*

Authentication System Based on Keystroke Dynamics

Project written by:

Name: Signature:

Sorin ZAHARIA ____________________________

June 7, 2018

Title:
Authentication System Based on Keystroke Dynamics

Theme:
Services and Platforms

Project period:
February - June 2018

Author:
Sorin Zaharia

Supervisors:
Per Lynggaard

Samant Khajuria

No. of Pages: 81

No. of Appendix Pages: 12

Total no. of pages: 103

Finished: June 7, 2018

Aalborg University Copenhagen
A. C Meyers Vænge 15
2450 København SV
Secretary: Maiken Keller
Telephone (+45) 9940 2471
mks@staff.aau.dk

Abstract:

Information is a very important asset for companies and users. With new types of attacks being developed at a quick pace, innovative technologies are required to keep information safe. In this project, a keystroke dynamics authentication system is proposed, which aims to solve the issues of classic one time log in systems. The problem with these systems is that, no matter how secure they are, once the authentication step is performed, there is no way for the system to authenticate the users continuously. In this project, the concept of keystroke dynamics is introduced, and a system that can continuously authenticate users based on freely-typed text is proposed. Different feature extraction methods are described that should capture the users' typing behavior.

When uploading this document to Digital Exam each group member confirms that all have participated equally in the project work and that they collectively are responsible for the content of the project report. Furthermore each group member is liable for that there is no plagiarism in the report.

Contents

Abbreviations

AAU	Aalborg University
AAU-CPH	Aalborg University - Copenhagen Campus
API	Application Program Interface
APP	Application
CIA	Confidentiality
DD	Down - Down
DU	Down - Up
EER	Equal Error Rate
FAR	False Acceptance Rate
FR	Functional Requirement
FRR	False Rejection Rate
GDPR	eneral Data Protection Regulation
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
I/O	Input/Output
ID	Identifier
IDP	Identity Provider
IT	information Technology
KNN	K Nearest Neighbors
NFR	Non-Functional Requirement
NIST	National Institute of Standards and Technology
OOP	Object Oriented Programming
OS	Operating System
PC	Personal Computer
SAML	Security Assertion Markup Language
SOTA	State of the Art
SSO	Single Sign On
STDEV	Standard Deviation
SVM	Suport Vector Machines
UD	Up - Down
UML	Unified Modelling Language
UU	Up - Up

1| Introduction

Due to rapid digitization of the industry, data has became one of the most valuable and critical assets of enterprises. Data leakage is a serious threat for companies, which can cause massive financial and reputational loses. This is why, increasing security in order to prevent data loss is one of the most pressing objectives for enterprises today. Furthermore, with new regulations being enforced, for example GDPR, it is a prerequisite for companies to provide safeguards and lawful ways of processing for personal data in order to be compliant.

There are several ways in which data can be leaked from inside a company. It is often said that end users are the weakest link in a security chain. After all, even if all data is encrypted, if an attacker manages to compromise devices belonging to internal employees they could get access inside the enterprise. There could also be cases where an internal employee intentionally leaks data for various reasons like sabotaging or revenge.

In order to minimize the risks, enterprises implement controls to make sure the users are who they claim to be and they only have access to what they are supposed to. Most commonly, users are asked to log in using either something they know, for example user names and passwords, something they have, for example tokens or smart cards or something they are, for example biometrics like fingerprint sensors or face detection. In order to strengthen security even more, combinations of these methods can be used.

However, once the authenticity of the users is confirmed by any of these methods, under the process called authentication, they are granted access to the systems. If the actual user changes while the log in is still active, for example, another person physically or remotely takes control of the computer, these authentication methods don't provide a straightforward way to sense the change.

This problem raises the need of an authentication system that can perform continuous authentication of the user. Such systems should be able to learn the behaviour of the users based on their interaction with a system, for example their typing or mouse movement patterns, and be able to differentiate legitimate users from intruders. Furthermore, as productivity in an enterprise is very important, these systems should be transparent and non-intrusive with the user's work and should not require extra hardware added to the systems.

With the latest developments in the machine learning field, the possibility of developing such systems that would provide good performance is possible. Called, behavioral biometrics, these authentication technologies promise to offer continuous authentication of users based on their computer usage patterns. In this paper, a concept of implementing such technologies, in order to enhance security is presented. The system should be transparent in order to not affect productivity while ensuring the privacy of the user.

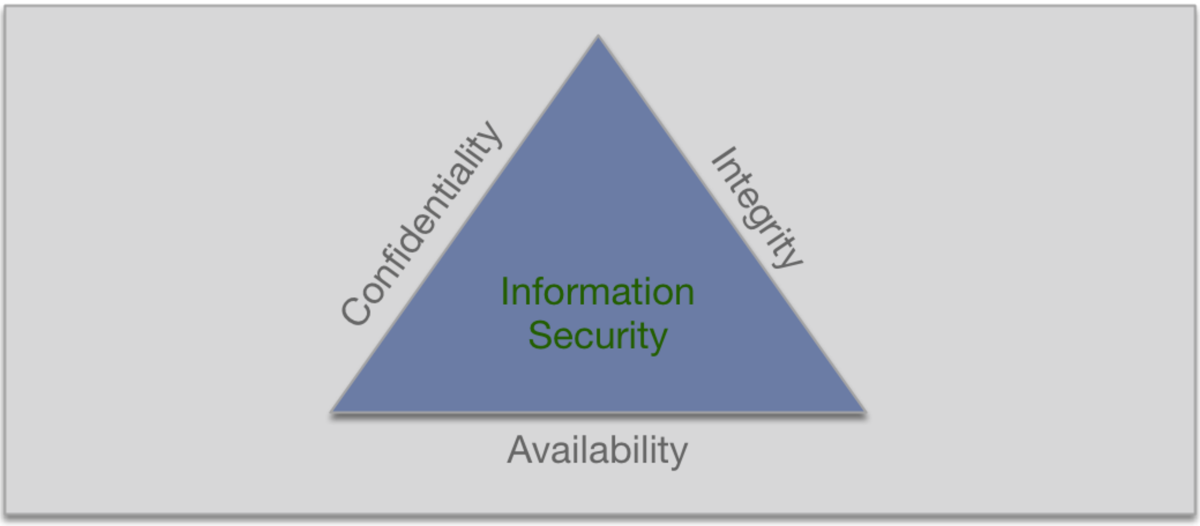


Figure 1.1: Illustration of the main objectives of Information Security, Confidentiality, Availability and Integrity, forming the CIA triad.

1.1 Background

1.1.1 Information Security Concepts

Information is one important asset for individuals and enterprises [1]. In the previous section, the term data was used. In principle, information is data that is processed or organized in a way that is meaningful or useful. For example, a word or a character can be considered data, however when put in a context, a combination of more characters or words becomes information as it becomes useful to a person.

Since it is an asset, disclosure, modification or unavailability of information can cause expenses or loses to a company or an individual. In order to prevent this kind of actions, security protection mechanisms are being deployed. There are three main security objectives that need to be fulfilled by these mechanisms [1]. These objectives are considered to be at the core of information security and they are forming the so called CIA triad as illustrated in figure 1.1.

- Confidentiality - refers to protecting the information from being disclosed to unauthorized entities.
- Integrity - refers to protecting information from accidental or intentional modification which would affect the data validity.
- Availability - refers to the fact that information and services must be available when an organization or individual needs them.

In an ideal case, all these objectives would be completely covered, with state of the art technologies. However, often in cybersecurity everyone knows what should be done, but most likely resources to do it are not sufficient [2]. Most of the times, companies would not put a great effort in protecting an asset that brings small direct benefits. Furthermore, sometimes even when defenders get what they want, attackers are very effective in exploiting gaps in those defences[2].

As it is further described [2], cyberattacks have evolved greatly. Back in time, cyberthreats were viruses, worms, and Trojan horses. These kind of threats were usually infecting random computers on the internet while posing small threat to enterprises. Enterprises were mainly protected by firewalls and anti viruses, which were more than enough to keep threats away. However, today hackers are targeting enterprises more and more and with mobility on the rise, employees are not only operating from a main network that can be easily protected. By infecting machines that have access inside the network, hackers can take control of them. This would allow them to connect to various systems, steal credentials, and even move laterally across the network, infecting more and more machines.

One of the most effective way of infecting computers inside an organization is by using phishing attacks [2]. This kind of attacks are usually carried by sending scam e-mails to employees, that contain infected attachments or links. When they are opened, the hacker takes control of the victim's computer. Sometimes users also tend to reuse usernames and passwords in multiple services. This way, if a hacker manages to steal credentials from one service, they can get access to all the other services that the user is registered to. One way to mitigate these impacts is by using strong identification, authentication and authorization controls, in order to determine and validate the user's identity and ensure that only authorized entities can access resources, while compromised users can be immediately cut off.

1.2 Authentication

When a person is performing transactions online, they are using a digital identity. While a subject has one identity in real life, they can have multiple digital identities, depending on the context. For example, a user could have a digital identity for an online shopping store, while having another digital identity for connecting to their company. Furthermore, these digital identities do not necessarily directly point back to the real life subject. For this matter, in order to make sure the subject is actually who they claim to be, identity proofing is required. As introduced by NIST [3], digital authentication is the process that determines the validity of a subject's claim for a digital identity. It establishes that the physical subject that is trying to access certain resources, is actually in control of the technologies that are used to authenticate.

In order to be authenticated, a user should provide some sort of evidence. This evidence is called a credential [4]. These credentials can be of different types and can be presented under the form of something they know, something they have or something they are. These types of authentication together with examples are illustrated in figure 1.2. Using multiple types of

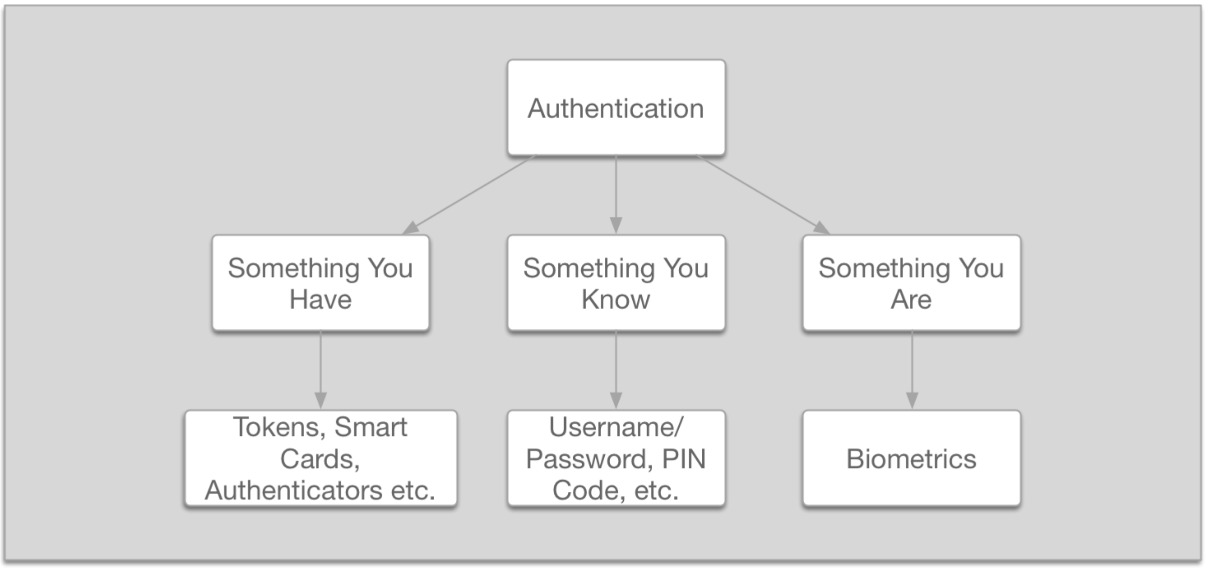


Figure 1.2: Illustration of the types of authentication and examples of implementations for each type.

authentication is used to increase security under the process called multi-factor authentication.

Two important concepts should be described in this this context, identification and authentication. The process of identification refers to determining who the user is. Typically, the security system searches through all the abstract objects, usually user IDs, that it controls and finds the specific one for the privileges that the user is applying for [1]. On the other hand, authentication is used to confirm the validity of a user. When a user claims that they are represented by an abstract object, they must provide evidence to prove their identity. If this is successful, the user is granted rights and permissions specific to the abstract user object (user ID) [1].

In order to avoid the scattering of a subject's identities around systems, the concept of federated identities has been introduced. This concept and different uses cases are presented in the Ping Identity whitepaper [5]. The main idea relies on the fact that the identity of subjects is managed by one entity called IdP (Identity Provider). IdPs in one federation should be able to communicate and have a complete picture of a subject's identity. When a subject tries to access a resource, the IdP is able to guarantee to the server holding that resource, that the user is authenticated and they are who they claim to be.

While the federated identity concept is broad, one of the concepts it introduces is SSO (Single Sign-On). Protocols like OpenID Connect [6] or SAML [7], have been introduced in order to support SSO. This way, companies that are using multiple systems, inside or outside their premises, can allow their employees to authenticate in one place and to use all the systems that they have access to without having to authenticate again. After the authentication process is performed, the authorization process ensures that a user should or should not have access to a resource based on policies.

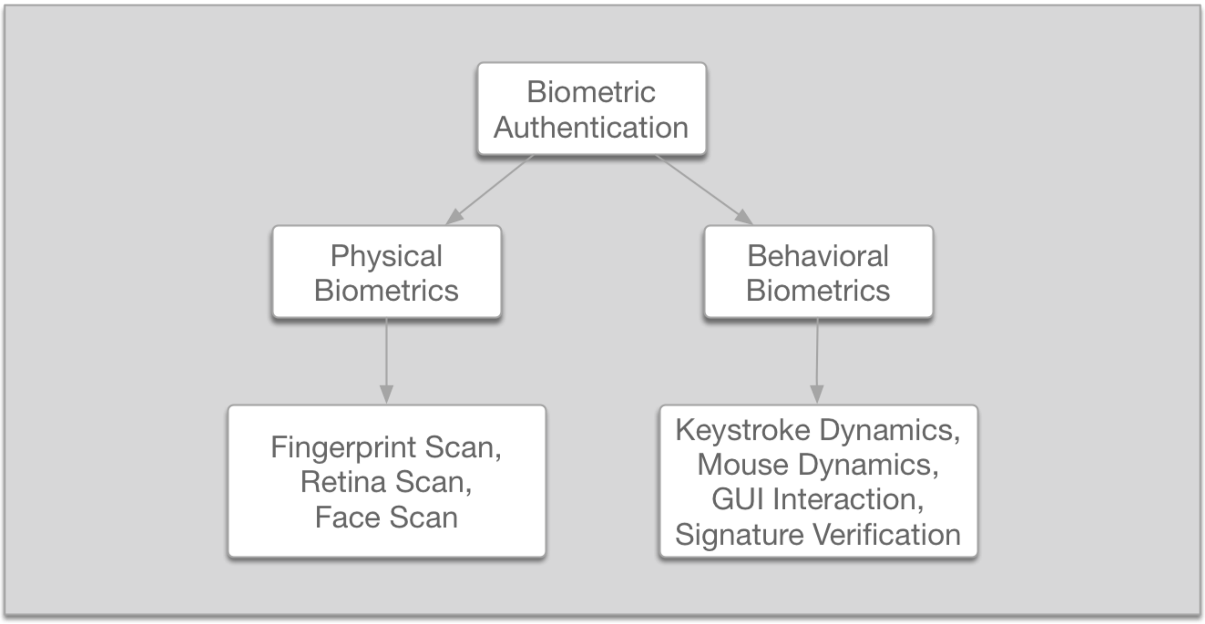

Biometric Authentication
Physical Biometrics
Behavioral Biometrics
Fingerprint Scan, Retina Scan, Face Scan
Keystroke Dynamics, Mouse Dynamics, GUI Interaction, Signature Verification

Figure 1.3: Illustration of the types of biometric authentication and examples of implementations for each type.

1.2.1 Biometric Authentication

When compared to other traditional authentication methods, like user names and passwords, biometric authentication has an advantage, as it is based on something you are which is harder to copy or steal [8]. Furthermore, biometrics should offer an inextricable link between the authenticator and its owner which can offer the property of nonrepudiation, something that can't be fully achieved through passwords and tokens since they can be lent or stolen [9]. This property provides proof of a transaction, so that the involved parties cannot negate its authorization or their involvement.

Biometric authentication can be classified under two different categories, "physical biometrics" and "behavioral biometrics" [9]. The two types of biometric authentication, together with a few examples of implementations, are illustrated in figure 1.3. Physical biometrics refer to authentication of the user using stable body parts like fingerprints, face, iris and so on. Generally, these methods are more known to the public, having a higher implementation rate as well as being considered to be more reliable [10]. However, one drawback for this type of authentication is that it requires extra pieces of hardware, which add an extra layer of complexity to the login process, as well as additional costs [8].

Behavioral biometrics, on the other hand measure behavioral tendencies that identify a person, like keyboard typing (dynamics), mouse movement, handwritten signature and so on. While this kind of authentication does not require any extra hardware, it has a lower adoption rate compared to physical biometrics mainly because of the variability of human body and mind over time [8]. However, a big benefit of behavioral biometrics is that authentication can occur actively throughout a user's session. This prevents cases when the user session is hijacked after the initial logon [8].

One way presented in the literature for implementing behavioral biometrics is keystroke dynamics [11][12][13][14]. Other approaches like mouse dynamics, voice recognition, signature verification and GUI (graphical user interface) interaction analysis are, also, presented in the literature. These technologies can also be combined in fusion systems [15].

1.3 Problem Definition

This section introduces the main research question that this report is based on as well as subquestions which will be used to fully investigate the problem.

How to implement a machine learning system that uses keystroke dynamics to continuously authenticate the users?

1. *What are the cyber security improvements that such a solution would bring?*
2. *How to continuously track a person's typing for finding patterns?*
3. *Which features should be measured in order to uniquely identify a person?*
4. *Which machine learning algorithms should be used in order to adapt to and detect the person's typing pattern?*
5. *How to calculate the performance of the system?*

The objective of this project is to develop a concept for an authentication system based on keystroke dynamics that uses machine learning to take decisions. The five subquestions provide more granularity to the problem, by identifying more detailed steps that need to be fulfilled towards solving the problem. The first subquestion suggests that it should be analyzed if such a system would provide solutions to any of the cyber security issues that enterprises and users are facing today, in order to understand if such a system is necessary.

The subquestions 2, 3 and 4 refer to the technical parts of developing such a solution. Since keystroke dynamics involves authenticating a user based on the way they type, second question indicates that a way of tracking the user's typing continuously, in a way that can lead to finding typing patterns, should be researched. Furthermore, question three aims at organizing the tracked data in unique sets of features for users that would allow a machine learning algorithm to make distinctions between them. The fourth question introduces the discussion about which machine learning algorithms are appropriate in order to fulfill the objective of the project. Finally, the last subquestion adds the idea that a way of calculating the performance of the system, in order to understand its applicability in the real world, should be introduced.

1.4 Structure of the Project

The next chapter, "Methodology" introduces the methods and approaches taken throughout the project from a research and development point of view. Furthermore, the project continues with the "State of The Art" chapter where previous work as well as current state of the topics and technologies used for this project are introduced. Three main topics are discussed, Cyber Security Concepts, Keystroke Dynamics as a persistent authentication option and Machine Learning Algorithms. The Cyber Security section discusses the current issues that users and enterprises are facing today and which controls are applied to tackle these issues.

When researching Keystroke Dynamics, different approaches for implementing authentication as well as different choices on which features are used for differentiating users and how they are organized will be are introduced. In the Machine Learning part, the different types of algorithms are introduced, together with which algorithms are used in the current Keystroke Dynamics algorithms.

The "Analysis" chapter discusses the elements introduced in the "State of the Art" chapter from a business and technological point of view, in order to come up with functional and non-functional requirements for the system. It is discussed which of the approaches introduced in the "State of the Art" chapter are relevant to the problem introduced in section 1.3. Also, it is discussed how the researched approaches are useful and how they can be adapted for reaching the project goal.

The requirements resulted from the "Analysis" chapter are then used as input for the system design, introduced in the "System Design" chapter. A short description of the system and it's functionality are presented. Furthermore, a high level architecture is introduced, followed by detailed description of each entity and the interaction between them.

The "Implementation" chapter describes the implementation steps of the project, describing the technical choices for implementing the system introduced in the "Design". A "System Testing" chapter is also included where the implemented system is tested with different settings and results are presented. The project ends with "Discussion" and "Conclusion" where the findings in this project are summarized, the applicability of the system in real life is discussed and what future work can be performed in order to improve the system.

1.5 Limitations

In order to limit the scope of the project, limitations are set for the project. They are presented in the following list:

- Legislation and compliance aspects are briefly introduced but will not be analyzed in depth.

- A limited proof of concept will be implemented, additional functionality being proposed for future works.
- Samples for testing the system have been collected in an university environment, by students, so the background of a participants is similar. Furthermore, since the software used for collecting features worked only on Windows OS and could be manually started and stopped by the participants due to privacy concerns, the samples sizes are limited.
- The proof of concept in this project is built for the Windows OS and it's functionality is only guaranteed for the English language, with a Latin alphabet.

2| Methodology

This chapter introduces the approach taken and the methods used in order to gather data, analyze it and ultimately design and develop a system that tackles the problem introduced in section 1.3. The research techniques used for gathering data are detailed, together with the techniques used to conduct the design and the implementation, as well as how they relate to the research questions. The reasons behind the methodology choices are described. The research strategy covers objectives, issues, limitations and data collection resources. This strategy is described for every stage of the project in this chapter.

2.1 Project Phases

As described in the chapter introduction, the project is built over three phases: Research, Analysis and Design and System Development. As the purpose of each phase is very different, the methods used vary accordingly. In this section, each phase is described, together with the methods used for each of them.

2.1.1 Research

The research stage of the project includes the initial investigation that is present in the introduction and background knowledge. The findings at this stage result in defining the problem formulation and sub-questions. The research is continued in the State of the Art chapter where current contributions and developments in the scope of the project are introduced, in order gain knowledge in the field, as well as, getting familiar with already existing technologies. The gained knowledge is further analyzed and should be used to shape the final solution used to answer the problem. The areas included in the research stage form the theoretical framework of this project. The research framework is introduced in the upcoming section 2.2.

2.1.2 Analysis and Design

Following the State of the Art, the Analysis chapter aims at analyzing the theoretical findings in report to the main problem and the attached sub-questions. Different approaches proposed in the literature for implementing keystroke dynamics systems are described and their usefulness in achieving the goal is discussed. The output of the Analysis chapter is to provide a list of requirements that describe the functional and non-functional behaviour of the system. The requirements are prioritized using the MoSCoW method and a rationale is provided. MoSCoW method is further described later in this chapter in section 2.3.1.

Using the requirements, the system design is performed. A description of the system functionality is provided, together with a high level design followed by low level descriptions of the

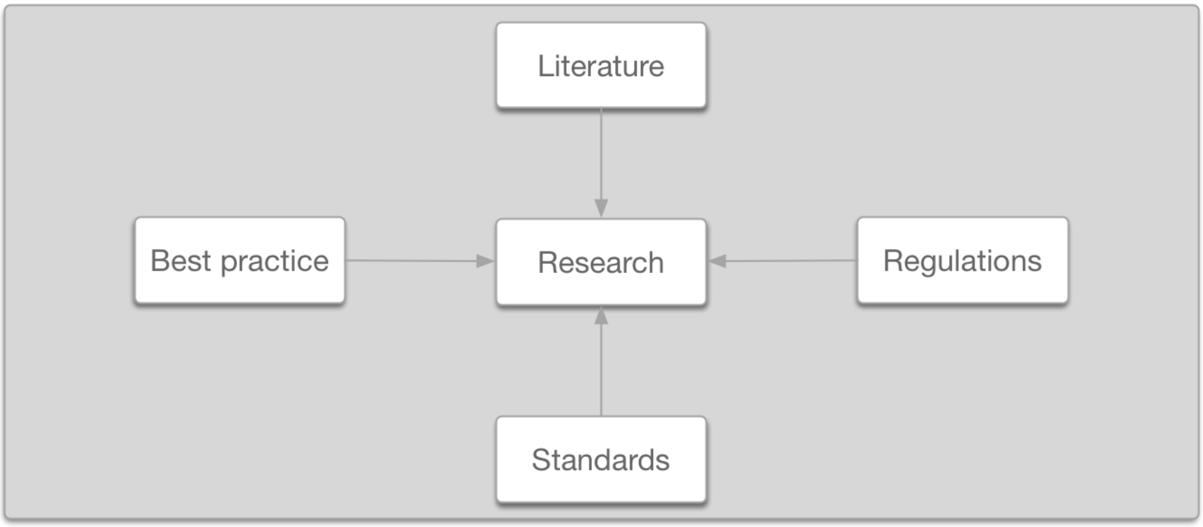


Figure 2.1: Illustration of the chosen research methods. Inputs from literature, regulations, standards and best practices are used in order to perform the research part of the project. This figure is based on the Methodology chapter in project [16]

.

entities and modules interactions. In order to make it easier to visualize the functionality and interactions in the system, UML (Unified Modelling Language) diagrams will be used. UML will be further described later in this chapter in section 2.3.2.

2.1.3 Implementation

By the end of the project, it is expected that a functioning prototype can be demonstrated. Low level implementation details are introduced in this chapter, for example coding language, libraries, resources and so on. The development is built based on the outputs from the Analysis and Design phases. An agile approach is taken for the software development, with the requirements being implemented in different sprints, following the SCRUM methodology. This methodology is described in the upcoming section 2.3.3.

2.2 Research framework

The research framework used for this project is introduced in figure 2.1. The sources for the project's research are illustrated as the squares that are pointing towards the research process.

- **Literature** - The literature research refers to books, articles, research papers and other publications. Since the purpose of the project is to propose a system to perform the required functionality, papers running experiments on the matter will be also studied, in order to understand which of the approaches are providing better results. The materials are gathered from online libraries, for example but not limited to, Aalborg University online library or other universities libraries, IEEE Explore, ScienceDirect, ACM Digital Library, also using tools like Google Scholar and Mendeley to help search for materials.

- **Standards** - Standards, as described by ISO [17], are documents or publications that are built by a consensus of subject matter experts and are approved by a recognized body that also provides guidance on the design, use or performance of materials, products, processes, services, systems or persons. By using standards in the project, allows for compatibility, interoperability and consistency across the system, as well as future expandability.
- **Regulations** - Beginning of May 2018, the General Data Protection Regulation comes into force. It is meant to strengthen the individual's right to privacy. Due to it's legal nature, some concepts introduced by GDPR are discussed throughout the project. However, an in depth study regarding regulations will not be performed.
- **Best practice** - This part of research contains published materials from individuals and organizations with relation to the best practices that should applied when tackling a problem in our area. This kind of data sources is considered highly biased due to their interest in the area.

2.3 Methods and Tools

In this section the tools and methods that are used in this project are introduced. The MoSCoW method is used for requirement prioritization. The Unified Modelling Language is used for drawing diagrams for describing the system functionality. The SCRUM method is used for splitting the implementation phase into multiple sprints based on requirements prioritization.

2.3.1 Moscow

In order to prioritize requirements, based on the importance of the business needs, the MoSCoW method introduces a criteria of prioritization of requirements [18]. This method can be also used to prioritize tasks, products, use cases, user stories, acceptance criteria and tests [19] but for this project it will be applied just to requirements. As the MoSCoW is an acronym, excepting the two Os that were added to make the word more readable, the letters stand for:

- **Must Have** - These requirements are vital for the delivery of a product. Not fulfilling these requirements may bring functionality, safety or even legal issues to the product. In order to meet the goals, all the requirements under this category must be implemented.
- **Should Have** - The requirements included in this category are important but not vital for the project. The system is still able to run and perform it's objective, but not including these requirements may degrade performance or cause missing features.

- **Could Have** - The requirements in this category have a much less impact on the project. These features are usually niche features that are not essentials but may provide benefits. If resources allow these features can be implemented, even at later stages but are not mandatory.
- **Won't Have this time** - These requirements are agreed to not be delivered in the agreed time frame or with the current resources. These entities are listed to help clarify the scope of the project. If the project progresses quicker than expected or the scope changes these requirements might be implemented.

2.3.2 Unified Modeling Language

Unified Modelling Language helps in specifying, visualizing and documenting models of software including their structure and design. It is used to provide system architects, software engineers and software developers with the tools for analysis, design, and implementation of software based systems [20]. The following UML diagrams will be used for the purpose this project:

- **Activity diagram** - A UML structure diagram describing the flow of information in the system from start to finish. I will be used in the introduction to the system design in order to give an overview of the system functionality.
- **Sequence diagram** - They are artifacts that model the flow of logic in the system in a visual manner. The messages are ordered in a sequential manner.They are used for both the design and analysis phases.
- **Component diagram** - A UML structure diagram that shows components, provided and required interfaces, ports, and relationships between them.
- **Class diagram** - A UML structure diagram that describes the structure of a system by showing the system's classes, their attributes, methods and the relationships among objects.

2.3.3 SCRUM

Compared to the step by step approach in the traditional waterfall method, where the product is developed in clear steps, with verification running at the last step, this methodology takes a more adaptive approach. SCRUM applies an incremental design, breaking down the system in smaller parts. This way, the smaller parts are implemented, with the complexity of the product gradually increasing [21].

As illustrated in figure 2.2, at the beginning of each sprint, a planning is performed where the requirements with the highest priority are moved to a sprint backlog. The tasks are then divided

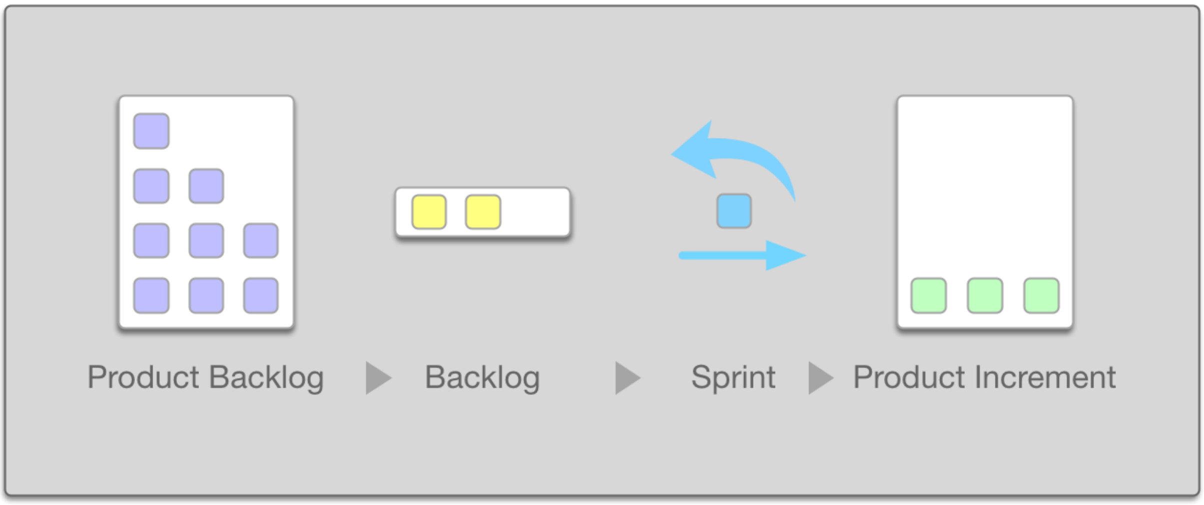


Figure 2.2: Illustration of SCRUM, used as implementation method for this project. In the product log the requirements are collected, which are then prioritized using MoSCoW and assigned to different backlogs based on priority. The requirements are then implemented one by one in the sprints. This figure is based on the Methodology chapter in project [16].

in smaller sections. The progress should be periodically discussed within the implementation team and goals should be adapted. In order to manage the time effectively, daily and weekly milestones are adopted [21].

3| State of the Art

3.1 Cyber Security Concepts

This section introduces concepts about how cyber security is treated by enterprises and individuals and what measures are taken in order to improve the security of their systems. The concept of persistent authentication is introduced, as well as, what kind of attacks are popular when it comes to exploiting the authentication process.

3.1.1 Risk Management

As introduced in section 1.1.1, companies and individuals hold information that is bringing them value. In order to protect this information, three main security objectives have to be fulfilled, confidentiality, integrity and availability (CIA). However, ensuring that these objectives are fully covered may require a lot of resources. For example, buying a fully featured antivirus may be very expensive for one user, as well as an overkill, as the risks that they are exposed to may not justify the investment. Same principle applies for enterprises, as they could buy expensive protection equipment to protect information that may not have such big value to the company or protecting against attacks that are unlikely to happen.

In order to understand what risks a company is exposed to, where compromises can be made, what the effect of the compromises would be, the consequences of these compromises, and how to reduce the impact or likelihood of of these consequences, the risk management process is introduced [2]. As shown in figure 3.1, the risk management process goes through six main steps.

The six risk management steps are presented as follows [2]. The first step involves identifying the assets that need to be protected. For a company, such assets could be personnel, facilities, processes and information. The next step, "vulnerabilities", introduces ways in which the assets can be compromised from the CIA point of view. The "threats" step involves identifying ways in which the vulnerabilities can be exploited in order to attack the asset. In the fourth step, threats and vulnerabilities are combined in order to identify the risks. Naturally, a risk is higher for threats and vulnerabilities that affect an area where the enterprise is less protected. After the risks have been identified, the way they should be treated is studied in the next step. The risks could be avoided, by eliminating the vulnerability or the threat. The risk could also be mitigated, reducing the chance that this risk will occur. The third way of handling the risk is to share it, which refers to outsourcing it to a third party, for example an insurance company. Furthermore, a risk could be retained, which refers to accepting the risk and accepting the consequences.

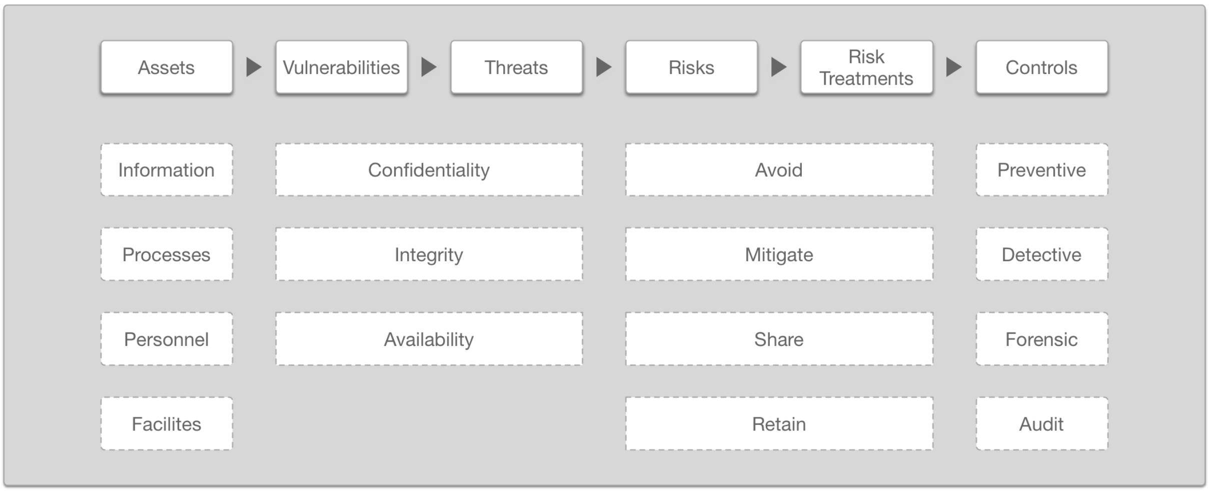


Figure 3.1: Illustration of the six steps of the risk management process. Underneath the steps, the elements that are involved at each step are described.

In the last step, in the case where companies decide to reduce a risk, security controls in order to achieve this objective are introduced. There are four types of controls [2]:

- Preventive Controls - This kind of controls block the threat in order to prevent the risk to occur.
- Detective Controls - This kind of controls should detect when a risk happens and generate alerts which can be acted upon.
- Forensic Controls - This kind of controls monitor and collect records of activities related to a risk and can be used to produce artifacts that support the operation of detective controls, as well as investigating incidents and auditing controls to verify their operation and effectiveness.
- Audit Controls - This kind of controls are used for investigating the presence of a risk and incidents associated with that risk and the operation of the controls that are related to that risk.

3.1.2 Persistent Authentication

As described in section 1.2, authentication refers to the process of ensuring that a subject is who they claim they are. Multiple proofs may be required in a process called multi-factor authentication, where the user should fulfill two or more criterion, based on something they have, something they know or something they are. However, while it is possible to increase the security level of the authentication process, it is a one time process. In this case, if the user performs the authentication steps but leaves their computer unlocked, another person could easily start using their computer, without the system noticing that the authenticated person is not using the computer anymore. Same goes for the cases when someone may take control of the computer remotely. As it will be presented in the next section 3.1.3, there are many types of attacks that may make use of this vulnerability.

Several approaches have been taken in order to attempt to solve this issue. One approach is presented in a patent, published by Google [22], where a watch that collects biometric information about the wearer is introduced. The collected information is sent to an authentication module where the biometric information is checked against a baseline information. If the user is recognized, they are authenticated until the watch senses any changes, for example it is removed from the hand. Similarly, in another patent [23], the author introduces a wearable that gets activated by an action, for example putting it in the hand, and gets deactivated by another action, for example removing it from the hand. Such device could communicate with a server in order to authenticate a user.

A persistent authentication system is introduced for smart environments [24]. A system is proposed that is able to authenticate and track users using 3D cameras. This way, the users are authenticated once, and authentication then persists since the system can track the position of the users at all times. A similar approach is taken in [25], where the user is authenticated once, when entering an access controlled area, and then relies on sensors to track individuals and keep then authenticated. Other papers introduce similar ideas as well, making persistent authentication a popular concept for smart environments.

However, as it can be concluded from the paragraphs above, these kind of implementations require extra hardware and are related to particular use cases. Another way, proposed in the literature, for implementing persistent authentication is by using behavioral biometrics, as introduced in 1.2.1. This concept proposes a system that understands the behavior of the user based on the way they interact with their computer by following keystrokes, mouse movements, GUI interactions and so on, without requiring any extra hardware or any modification in the user's normal daily behavior. This concept will be further analyzed in this project.

3.1.3 Types of Attacks

Since it is still in a research phase, not many studies have been made about security of keystroke dynamics. As a resultt, there are no reports about cases when such systems have been breached or attacked [26].

When it comes to vulnerabilities, a general idea is that the end users are the weakest link when it comes to enterprise security. As introduced in section 1.1.1, an attacker can make use of end user terminals to get access inside the networks.

Paper [26] introduces the traditional types of attacks when it comes authentication. Furthermore, in the book [2], common cyberattacks are described. Later in the project it will be analyzed how keystroke dynamics could improve security when it comes to these types of attacks.

Traditional types of attacks are categorized as [26]:

- Shoulder Surfing - Refers to the fact that a person can steal a user's credentials by watching them insert the information during authentication.
- Spyware - Refers to malicious software that is installed on the user's computer, that collects information about them without their knowledge.
- Social Engineering - Refers to the practice of obtaining confidential information about users by manipulation, generally through e-mail or telephone. This kind of attack is not an attack that takes advantage of vulnerabilities in the IT systems but rather exploits people's trust.
- Guessing - People tend to use easy passwords for their accounts, especially in the cases where they have to remember a multitude of them. This makes it easy for an attacker to guess the password or crack it with specialized software.
- Brute Force - This kind of attack refers to an attacker trying all the possible combinations until they reach the user's password. This is more of an extension from the "guessing" type of attack, except that usually specialized software is used. On the other hand, same as in the guessing case, more complex passwords would take a considerable longer time to crack than very simple passwords.
- Dictionary Attack - This kind of attack is similar to the brute force attack, in the way that it searches through different combinations. However, instead of looking through all the possible combination like a brute force attack, it searches to a number of possibilities, for example the most used passwords online.

An example of social engineering attack, which is one of the most effective ways of getting into an enterprise network, is the phishing attack [2]. Attackers send mails with infected content that, once accessed, gives them access to the victim's computer. Keylogging, when used in malicious ways, it is considered a spyware type of attack where an attacker can install software on a user's computer which logs all their inputs, stealing passwords or sensitive information [2].

Another attack can be credential harvesting, where attackers can compromise systems that users use and steal their credentials [2]. This kind of attack is very effective in the case where users tend to reuse their passwords. This way, a hacker can steal user's credentials from systems with weak security and use them to get access into systems with much stronger security.

3.2 Keystroke Dynamics

This section introduces notions about the concept of keystroke dynamics. The historical evolution of this concept as well as ways of implementation proposed in the literature are presented.

Ways of testing the performance of such systems are also introduced.

3.2.1 Evolution of Keystroke Dynamics

The idea of using keystroke dynamics for authentication was introduced way before in 1980 [11]. The authors [11] carried an experiment to confirm if people are typing in timing patterns and if they can be distinguished on the basis of typing "signatures". A group of people was asked to type three predefined texts and then repeat the task four months later. The time between each two successive letters, a combination which is referred to as digraph, was measured. The results were that each person had similar typing behavior for both sessions and that it is reasonable to consider authentication procedures based on keystroke timing. The method of interpreting the data was purely statistical, with graphs and distances between two sets defining the similarity between them. This statistical approach is taken, also, in more recent papers where a reference set is trained in the system for each user. Each time that user is authenticated, a test set is created is then compared to the reference set.

Another popular approach is using methods that are taking advantage of the neural networks paradigm [13][26]. A neural network approach, uses automated machine learning techniques in order to build a prediction model based on historical data. This prediction model is then used to classify a new observation for a user that is being authenticated [26]. With the development of machine learning algorithms, the gap between these two approaches becomes less clear, as several algorithms are using different statistical methods in order to build a model and take classification decisions automatically.

More recent papers, are also using trigraphs or quadgraphs which are combinations of three or four successive letters in order to offer more context to keystrokes, increasing the accuracy of the measurements.

While initial work focused on predefined text samples, more recent studies have been looking at ways of analyzing long free text samples. When analyzing predefined text samples, keystroke dynamics authentication works in a similar way to the dual-factor authentication techniques that are widely available today. A system either asks the user to type a predefined text in a text box or apply measurements when the user enters their user names or passwords.

3.2.2 Types of Authentication

There are two security methodologies to which keystroke dynamics can be applied, static and dynamic authentication [13]. This categorization is illustrated in figure 3.2. In static authentication, keystroke analysis is performed by a system at the time of login. The timing patterns are calculated based on the user's entry of username and password and they are compared with a profile that has been explicitly trained previously. Based on the degree of similarity a decision

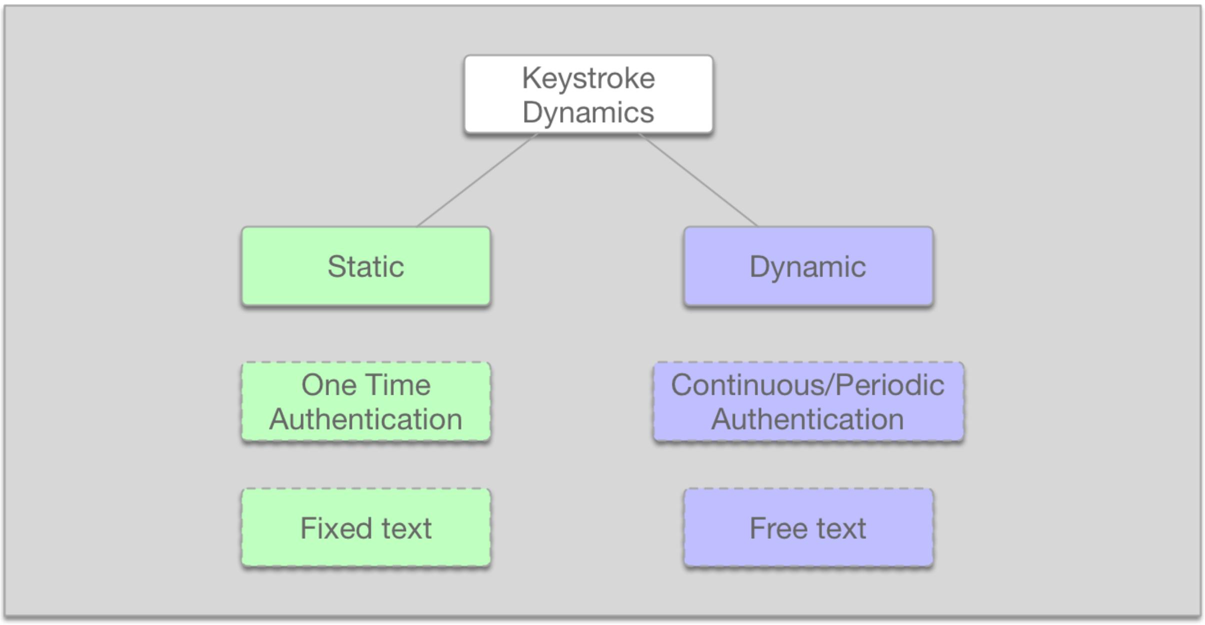


Figure 3.2: Categories of Keystroke Dynamics Authentication based on the input to the system. The specifics of each category are illustrated.

is taken either to accept or reject the authentication. This kind of systems have been proposed in literature in various publications [14][27] , and are considered to be the simpler of the two types as the sampled data is fixed. It is also possible to incorporate impostors patterns in the training of the algorithm in order to strengthen the classification model [22].

Furthermore, when using static context, the process of authentication will become more effective over time through training. Also, people tend to form a habit after typing their user names and passwords multiple times, developing a constant rhythm [13]. However, this also introduces certain disadvantages, as the rhythm of the user's typing changes over time, the algorithm has to either have tolerance towards this issue or adapt to it. Moreover, when a user changes their password, the algorithm will need to be retrained from scratch. Given these inconveniences, it can be concluded that while static context keeps the algorithms simple and provide good theoretical results, mainly because of the fixed-text nature, it is hardly suitable for practical systems that are designed for long term use [13].

On the other hand, dynamic authentication is a security paradigm that tries to create a user profile based on their typing patterns using an unknown stream of input data [28]. Such a system would monitor a user's keyboard and generate reports in real time or near real time about the confidence of the users' authenticity [13]. This is an advantage over traditional authentication methods as they mostly provide confidence at the time of log in while they don't check continuously if the person using the computer is still the person that logged in. However, because of the varying nature of the data, the system must be very carefully designed as there may be a lot of noise. Also, since long strings of data are analyzed, there is a likelihood that

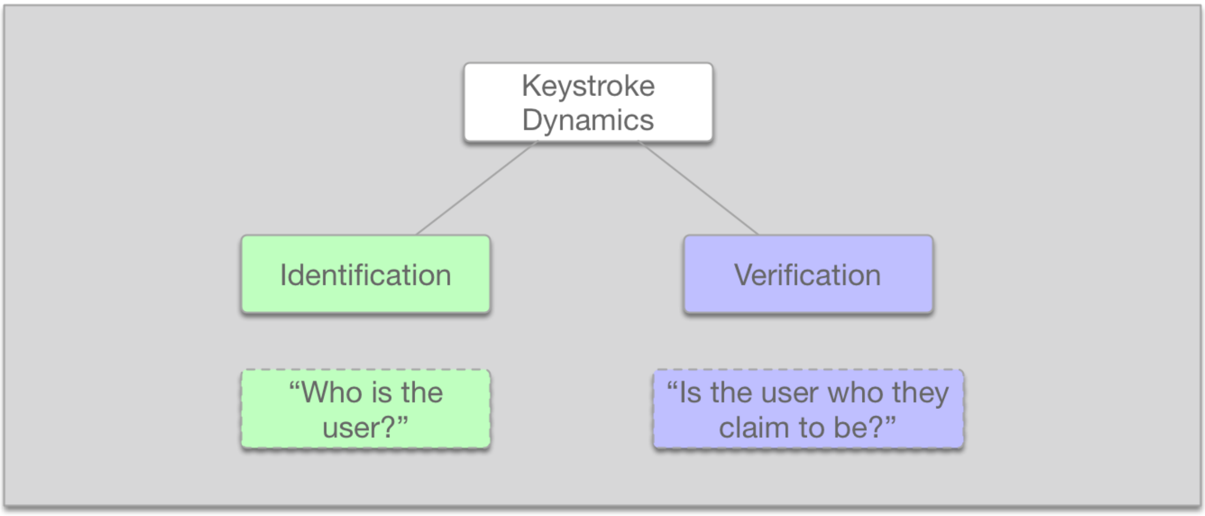


Figure 3.3: Categories of Keystroke Dynamics Authentication based on the type of decisions the system takes.

there may be pauses in the typing, like interruptions or distractions, or users may go back and make corrections in the cases of typing mistakes and so on.

These two ways of authentication are also summarized and the dynamic approach is further split into two categories, periodic and continuous [26]. In a periodic fashion, data is logged throughout the session and authentication is performed periodically at different time frames. In a continuous approach, data is captured during the entire duration of the session and continuously checked.

Other approaches introduced in [26], are keyword-specific, which extends the continuous or periodic monitoring to consider the metrics related to specific keywords. Another approach is application specific which also extends the continuous or periodic monitoring to develop separate keystroke patterns for different applications.

The functionality of the keystroke dynamics can be also be categorized as illustrated in figure 3.3, based on the way the system takes decisions, namely identity identification and verification [26]. Identification mode, also known as multi-class or one-to-many matching, helps answer the question "who is the user?". It occurs when the identity of the user is unknown. In this case, the system compares the biometric data provided by the user with all the entries in a database and returns the most likely one [29].

In identity verification mode, also known as binary or one-to-one matching, the question "is the user who they claim to be?" answered. The identity of a user claiming to access a system is known. This way, the system compares the biometric measurements provided by the user to the records belonging to that user.

3.2.3 Feature Selection and Performance Metrics

As introduced in [26], a variety of keystroke metrics are widely used in literature in order to perform free text, dynamic authentication. The following metrics are introdcued:

- Digraph latency - Typically measures the delay between two letters being pressed. In literature, some papers consider this metric as being the time between two key down events while others consider it to be the delay between the key-up and the subsequent key-down events.
- Trigraph latency - Extends the digraph latency to consider the timing for three consecutive keystrokes
- Keyword latency - Considers the overall duration of complete words or of unique combinations of digraphs and trigraphs.

Also, key hold timing is also one metric used in some implementations [13]. This kind of metric measures the duration of each consecutive key being held down. However, rather than having to choose between the inter key timing or key hold timing, it is rather preferable to fuse the metrics together into a more robust algorithm as proposed in [12][13].

Additionally, in paper [12], the authors suggest that standard deviation of each digraph can be used as features, when digraphs appear multiple times. They suggest calculating the mean values and standard deviations for these digraphs.

The typical metrics in order to measure the quality of a system that implements behavioral authentication are FAR (False Accept Rate), False Reject Rate (FRR) and Equal Error Rate (EER) [30]. FAR refers to the percentage of times when an imposter is accepted, while the FRR refers to the percentage of times when a legitimate user is denied. The EER is the error rate achieved when the detection threshold of the system is set in order for the FAR and FRR to be equal.

The results in the literature for keystroke dynamics authentication systems ranged from a FAR of 5% [31], to 2.24% [8], or even 1.24% [15]. However, these papers, together with other papers presented in the literature have obtained these results in a laboratory environment and they agree that further testing and improvement is required before this authentication method can be used in the real world.

3.3 Machine Learning

Artificial intelligence is a field that was started in 1956, at Dartmouth College, by John McCarthy [32]. It was created to automate many functions of business. Initially it was used in factory line automation, but with the development of computers and increase of processing power, human

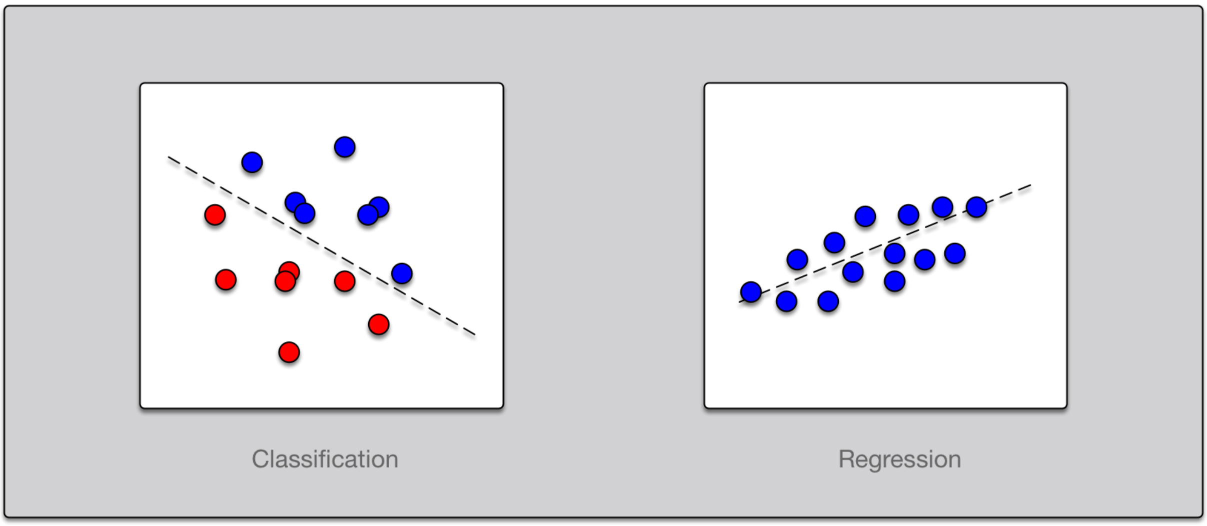


Figure 3.4: Example of supervised learning methods. On the left, a binary geometrical classification example is illustrated, where a linear border is found between the two classes. On the right, a regression example is shown where the algorithm tries to find a relationship between variables.

factors were added based on psychology, sociology, and neuroscience. It lead to the development of human-like solutions, for example designing machines based on human brain patterns called neural networks [32].

Machine learning is a subfield artificial intelligence. It refers to "the ability of machines to learn and work on new problem dimensions without being explicitly programmed to do so" [33]. There are three types in which a machine can learn, supervised learning, unsupervised learning and reinforcement learning [32].

3.3.1 Supervised Learning

In supervised learning, the data that is used to train the system has a known label [34]. This way, the inputs and the desired outputs are known for the training data. The system uses this data to build a model, that is later used for taking decisions. Regression and classification are types of outputs that a supervised learning algorithm can produce [34].

Classification refers to the operation of predicting classes based on observations [33]. The classes are known beforehand and the algorithm is trained using supervised learning. The training is done by providing multiple observations together with the classes that they belong to the algorithm. The goal of the algorithm is to find to which class are new observations most likely to belong. Classification algorithms are of different types, based on the way the decision model is built. The types are probabilistic methods, geometric methods or entropy methods. In figure 3.4, a liniar binary classification is illustrated.

Regression is similar to classification because the target values are known. However, while

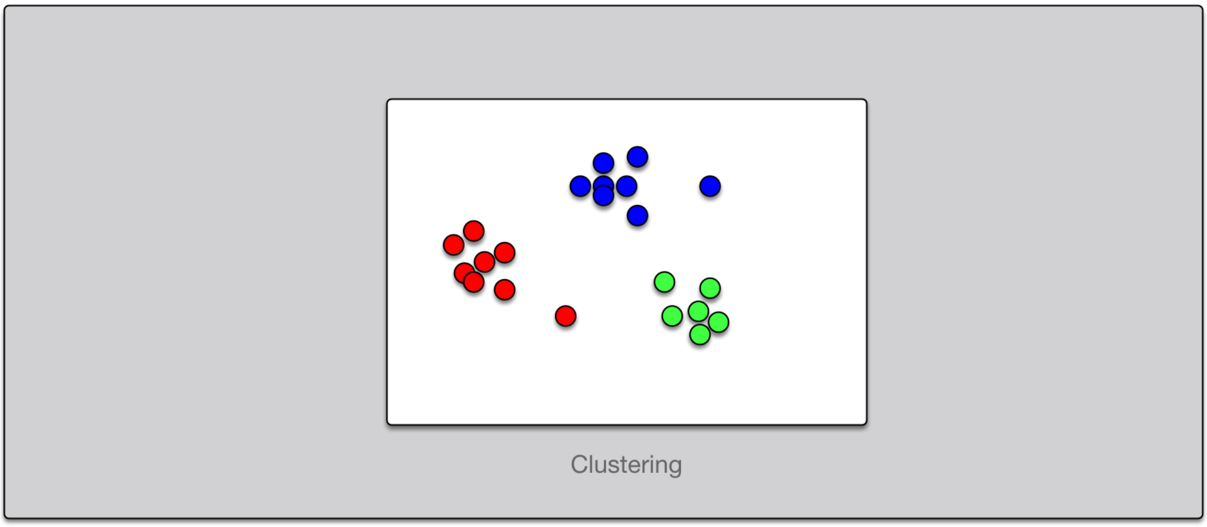


Figure 3.5: Unsupervised Clustering example where different measurements are arranged in clusters based to their distance from the closest mean.

in classification methods, the target values are nominal, in regression methods target values are numerical. The algorithms are concerned of modeling a relationship between variables, for example fitting a polynomial, as illustrated in figure 3.4. It is iteratively refined using a measure of error in the predictions, and it is considered that the algorithm is trained when the smallest error is reached [34].

3.3.2 Unsupervised and Reinforcement Learning

In the case of unsupervised learning, the input data used for the training is not labeled and a desired result is not known. The machine will try to find patterns and structures in the input data and prepares a model. This may be done through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity. Clustering or dimensionality reduction are approaches that make use of unsupervised learning [34].

Clustering algorithms' goal is trying to group together similar instances. After the inputs are grouped together, they are arranged in clusters by using methods like geometrical distances or probabilities. This process is illustrated in figure 3.5.

The reinforcement learning model helps the machine learn from interactions with the environment. It is based on the concept on how the output should be changed based on how the input changes. The machine receives positive or negative feedback for every action taken and will try to maximize the rewards that it receives [33]. An example would be a machine trying to learn chess where every good move will be rewarded, while bad moves would receive a punishment.

3.3.3 Machine Learning Algorithms

The machine learning algorithms that are going to be used later in the this project are briefly introduced in this section. Since features are being collected from known users, supervised learning method is being applied.

Naive Bayes is a Bayesian algorithm that explicitly applies Bayes' Theorem for problems such as classification and regression [34]. However, it uses the "naive" assumption of independence between every pair of features [35]. Naive Bayes learners require a small amount of training data to estimate the necessary parameters. However, although naive Bayes is known as a decent classifier, it is known to be a bad estimator [35].

K Nearest Neighbors is an algorithm functioning under the simple idea of "finding a predefined number of training samples closest in distance to the new point, and predict the label from these" [36]. The number of nearest neighbors can be configured by the user to fit their requirements.

Decision trees are using supervised learning method to create a model of decisions by using the actual values of the attributes in the data [34]. Entropy may be used in order to build decision trees.

Logistic regression, despite its name is a regression model and linear classifier, where the output is categorical [37]. It is estimating probabilities using a logistic function that can take any input and will always output a value between 0 and 1.

Similarly, SVM (Support vector machines) derives a linear decision boundary. It works based on the requirement that the distance between instances and the boundary, which is a hyperplane is as large as possible.

3.3.4 Algorithm Performance Testing Using K-Folds Cross Validation

Before using a machine learning in a production environment, its performance on the available datasets should be tested. One approach could be to train an algorithm and then run the tests on the same data. However, according to [38], this situation could lead to overfitting of an algorithm where it could have very good results on the test data but it would have bad performance on any new yet-unseen data.

K folds is a way of testing performance of machine learning algorithms that tries to avoid this problem. It works by splitting the dataset, comprised of features and labels, into k parts (thus the name K-Folds) and using a number of folds for training and a number of folds as validation to test the performance of the training. Nonetheless, in this case, by partitioning the dataset, the number of features available for training is lowered and the result may depend on the choice of the training and validation dataset [38].

Cross-validation comes to solve this issue [38], by running a loop where k-1 folds are used for training and the resulted model is validated against the remaining part of the data. The results from each loop are then averaged to obtain the final performance score.

4| Analysis

This chapter aims to analyze the concepts and technologies introduced in the State of the Art chapter 3, in order understand the benefits and clarify the requirements for building a solution that addresses the problem introduced in the Problem formulation section 1.3. The chapter is structured based on the project's subquestions, each section corresponding to one subquestion.

4.1 Cyber Security

This section describes the security benefits that a keystroke dynamics authentication system would bring to an enterprise or a user. Furthermore, the types of authentication that can be performed with such a system are introduced and it is concluded which concepts are most likely to answer the first sub-question (*"What are the cyber security improvements that such a solution would bring?"*), introduced in section 1.3. Subsequently, architectural choices in order to enhance security of the system are discussed.

4.1.1 Security Controls

Keystroke dynamics authentication is a concept that could provide a solution to multiple vulnerabilities that exist today in enterprises' or individuals' cyber security. In the section 3.1, popular types of attacks together with types of controls to mitigate these attacks have been introduced. Keystroke dynamics can be used to provide controls in order to mitigate these threats.

However, as described in the State of the Art chapter 3, this kind of authentication systems are more prone to errors than other traditional authentication systems so they should be carefully tuned and be used in conjunction with a primary authentication method. Depending on various factors like the application being used, or a user risk score, different actions can be taken when an alert is triggered. For example, if a user is accessing a critical resource, they can get locked out immediately when an intruder is detected, preventing the access, while for more non-critical resources that are used often, a decision could be delayed until more samples are collected or inputs from other systems are compared. However, in order to provide flexibility, as well as a separation of functional duties, the keystroke dynamics authentication system should send authentication results to a security center, where a decision can be taken based on additional factors, in a process similar to the one illustrated in figure 4.1.

Furthermore, the need of a primary authentication system is accentuated by the fact that a keystroke dynamics system requires an initial number of keystrokes in order to take the authentication decision. Without having an initial authentication step, the user's computer would be

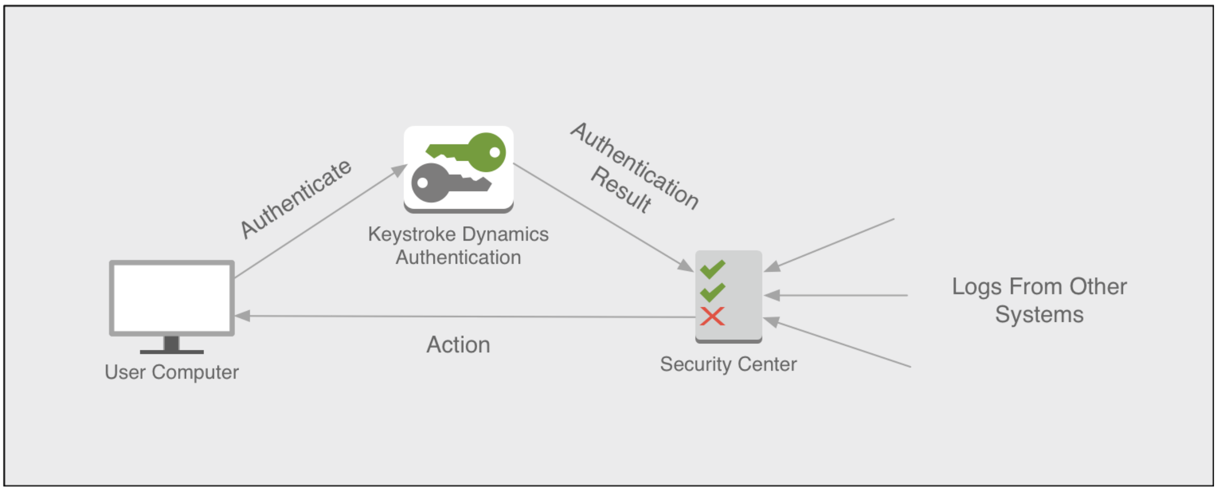


Figure 4.1: Keystroke dynamics authentications system used to provide detective and forensic controls to a security center. By analyzing inputs from multiple systems, the security center can take action upon a user, if necessary, or provide historical logs in case of breaches.

freely accessible until the first sample is analyzed. Furthermore, if an attacker is interacting with the computer in a way that does not include typing, they could have uninterrupted access.

That being said, a keystroke dynamics authentication system could prove very useful at providing detective controls and forensic controls, as described in section 3.1. This way, when the system detects an intruder, it can notify a security center, where input from multiple systems can be processed and a decision like locking the user out can be made. Furthermore, as a forensic control, such a system could provide valuable logs about the activity of users from authentication point of view. In the case where a breach or an attack happens, logs from the system could be checked for suspicious activity, as they provide a continuous report about the certainty of a user's authenticity.

4.1.2 Attack Mitigation

Different possible attacks have been introduced in section 3.1.3. These kind of attacks focus mainly on user authentication, finding vulnerabilities in this process. If an attacker manages to take control of a user's computer they may use this access for an even bigger scope, for example getting access inside a company.

Applying keystroke dynamics authentication can be useful against some of the most popular attacks performed today. As introduced in section 3.1.3, a shoulder surfing type of attack can occur at any moment when a user introduces their user name and password. However, using a keystroke dynamics authentication system, as second factor authentication, eliminates this problem, as even if an attacker could steal the credentials that a user needs in order to access their system, they would also need to match the exact typing pattern of a user.

In the case of spyware being installed by an attacker on the victim's computer, keystroke dynamics authentication may not prove very efficient in some cases, depending on how the spyware is operating. While it may be able to identify an attacker that remotely controls the machine, a spyware program could get access to the whole system of a user, closing the process or intercepting the user's typing patterns, allowing the attacker to reproduce this patterns with specialized software in order to trick the system into believing a valid user is performing actions. On the opposite side, there are social engineering attacks which don't exploit vulnerabilities in the IT systems, more getting information directly from the users by using manipulation techniques. As described in section 3.1.3, phising is a very popular attack that goes under this category. A keystroke dynamics authentication system may greatly help in this case, as in most of the situations attackers steal credentials from the users. However, it might be possible for hackers to ask users to write information while their keystroke pattern is registered. Nonetheless, it would require users to write long lines of text in order to get a meaningful keystroke pattern that can be used to replicate a user's typing.

When it comes to guessing, brute force or dictionary attacks, the situation is similar to the shoulder surfing case. Since dual factor authentication is used, even if the password of a user is cracked, an attacker would have to replicate the user's typing pattern in order to pose as a legitimate user.

In conclusion, it is safe to say that a properly implemented keystroke dynamics system can provide protection against popular types of attacks where an attacker can steal the user's credentials. It could provide detective controls as well as valuable forensics information about users' activity.

4.1.3 Authentication Type

As introduced in section 3.2.2, two main types of authentication can be performed using keystroke dynamics, static and dynamic authentication. Static authentication is using a fixed, known string as input. On the other hand, dynamic authentication happens during the user session, either continuously or at certain points in time, and it is applied on free-text strings.

Static authentication, can be easily combined with username and password authentication, allowing the system to perform keystroke analysis while the user types in their credentials. Using fixed sized strings of text, it has a performance that is very close to the most commonly used physical biometrics [15]. Applying this kind of authentication would be an improvement to physical biometrics as it doesn't require extra hardware and is transparent to the user. However, it does not solve the issue of continuous authentication. This is mainly because it only monitors particular strings of text at certain steps of using a system, so once the user performs the authentication, it is impossible for the system to monitor the free-text that the user types.

On the other hand, dynamic authentication using keystroke dynamics, is able to continuously or periodically analyze the user's typing activity and authenticate them, using any text that

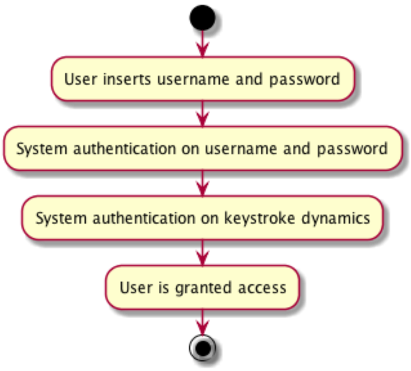

User inserts username and password
System authentication on username and password
System authentication on keystroke dynamics
User is granted access

Figure 4.2: Static keystroke dynamics authentication in a multi-factor authentication process. Keystroke information is collected during username and password entry.

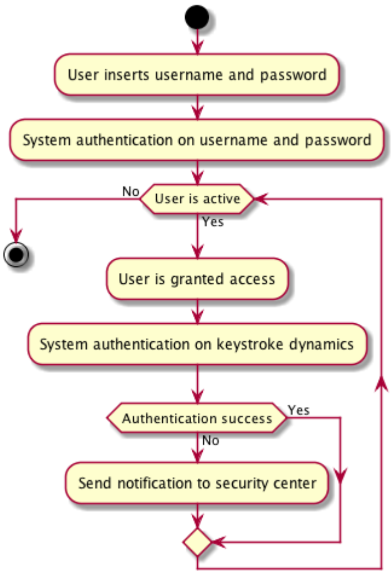

User inserts username and password
System authentication on username and password
No
User is active
Yes
User is granted access
System authentication on keystroke dynamics
Authentication success
Yes
No
Send notification to security center

Figure 4.3: Dynamic keystroke dynamics authentication in a multi-factor authentication process where events are sent to a security center. Keystroke information is collected from free-text typing of the user during their session.

the user inputs. This kind of authentication is able to identify if the session is hijacked after the user is initially authenticated. Furthermore, it doesn't require any extra hardware and it doesn't need any user intervention.

An example of the functional steps of the two authentication types is illustrated in figures 4.2 and 4.3. As it can be observed in figure 4.2, the static process is a simple step by step two factor authentication process. Firstly, the user introduces their username and password. While this action is performed, the system also collects keystroke information. The system then checks the username and password and, if they are correct, it checks the user's keystrokes to see if they match their pattern. If both steps are positive, the user is granted access to the system until they choose to log out.

In the case of dynamic authentication, illustrated in figure 4.3, the user is granted access immediately after their username and password are checked. The system starts collecting keystroke information and performs authentication when enough keystrokes have been recorded. If the authentication fails. a notification is sent to a security center. The keystroke authentication process is repeated continuously while the user is active.

Furthermore, as all the keystrokes of the user are being monitored, privacy is a big factor to take into consideration. If the keystrokes information is sent to a server where it is processed, it is very important that sensitive data does not leave the local device. Processing of the keystrokes information is required on the local device in order for the data to lose contextual information and be irreversible. This process is described in more details in the feature extraction section. Nonetheless, using a centralized server could greatly increase security as the user's data that is stored for modeling and retraining the machine learning algorithm, could be better protected, compared to storing it on the user's computer, ensuring confidentiality. In this case it should be noted that also the communication between the server and the client must be secure.

Using a client server approach could also improve security in the case of mallware taking control of the user's computer. The client could notify the server when a user becomes active. This way, timers could be activated on the server and the client. The client should send a message at every few minutes while the server should raise an alarm if a client stops transmitting. This way if the process is forcefully stopped on the client, by mallware for example, the server will raise an alarm, indicating that something is wrong with the client. The client should also inform the server when the user becomes inactive, in order to cancel the timers.

In conclusion, in order to solve the objective of performing, transparent, continuous authentication, a dynamic authentication keystroke dynamics solution will be considered for development. However, it is important to take into consideration the requirements that ensure user's privacy.

4.2 Benefits and Drawbacks of Keystroke Dynamics

This section introduces a short discussion related to the benefits and drawbacks of using keystroke dynamics authentication over other authentication methods, mainly physical biometric methods.

4.2.1 Drawbacks

As described earlier, in section 1.2.1, compared to physical biometrics, the behavioral biometrics approach has to deal with features that may change over time, even between two consecutive samples [39]. Furthermore, variability between two samples occurs even if the user strives to maintain a uniform typing pattern[39].

Some attempts were made to collect other features other than time, like pressure [40]. A system for smart cards is proposed, where smart cards and on card sensor can sense the pressure of the user's presses. However, most of the keyboards used today only send signals when a button is pressed or released. This way, the only measurement that can be performed is differences between events' time stamps. This measurements have to be heavily filtered however since they may be very noisy and, at parts, unreliable.

Some papers, like [12], introduce the idea of emotion recognition using keystroke dynamics. This suggests that the users may have a different typing pattern based on their emotional status or environment changes. For example, if a user is angry they might type differently. Also, if there are a lot of distractions in the environment, the user may type differently compared to when they are focused. Another example may depend on the application that the users are using. There may be different typing behavior between the case when one is typing a document compared to when they are writing code.

Furthermore, it is fair to suggest that the keyboard may influence the typing patterns of a user. If a user changes their keyboard, there may be some adaptation time. Also, as mobility is an important factor in companies today, people might travel to different countries, having to use different keyboard layouts. All these aspects bring noise into the measurements.

Moreover, while experiments ran in a laboratory environment might provide good results, it is very important to test these aspects in real world scenarios. While these factors may have a smaller impact when analyzing text in a static way, where the input is known in advance, when continuously authenticating the user by monitoring real time typing, the presented issues may greatly impact the performance, resulting in false accepts or false rejections of users.

4.2.2 Benefits

Despite the described drawbacks, there may be some use cases for keystroke dynamics authentication in certain cases, where it can be used as a multi-factor authentication method. One main advantage of it, however, is that it doesn't require any extra hardware. This authentication method could run in the background with little to no intervention required from the user. Furthermore, as it will be introduced in the research part of the project, tuning the algorithm in order to adapt to certain situations may provide good authentication results.

One way of tuning the algorithm is to set thresholds at values that would never reject legitimate users. Even though, this approach would reduce security, it would save the trouble of users complaints. The actions taken when an intruder is detected can also influence user's experience, as the system could be used as an informative system, where in case an intruder is detected an alert can be sent which is then analyzed further. On the other hand, for more sensitive systems, where the accesses are infrequent, the system could automatically lock the user out.

Another benefit which is a consequence of the facts that no extra hardware is required and no user intervention is needed, is that authentication can be performed continuously and transparently. This brings a huge plus to the authentication procedure, as the user can be authenticated even after the initial authentication step. This way, if the user session is hijacked after the initial authentication, the intruder should still be detected. Using this continuous authentication method, the system could report a confidence score for the user's authenticity to a security center regularly.

4.2.3 Conclusion

In conclusion, studies show that the typing pattern of a user may be influenced by external factors or emotional factors, and generally, recording the typing of a user may provide measurements with a lot of noise. However, with proper filtering and algorithm tuning, it has been demonstrated that useful results can be obtained (section 3.2.3). Furthermore, the fact that this kind of authentication does not require any extra hardware or user interaction, makes it an interesting candidate for future authentication methods.

4.3 User Tracking

This section compares ways through which a user's behavior can be tracked in order to perform persistent authentication, after the initial authentication step. Moreover, it is discussed what data sources can be used in order to extract information about the physical interaction between a user and their keyboard. This section aims to find the best approaches in order to tackle the second subquestion as introduced in section 1.3 (*"How to continuously track to a person's typing pattern"?*).

4.3.1 User Tracking Methods

As it can be inferred from the different methods introduced in section 3.1.2, in order to implement a persistent authentication system, a way of tracking the user is required. Usually, as a first step, the user is authenticated by various methods, and then the system keeps tracking the user in order to continuously confirm their identity. In order to perform this, some methods have been proposed, that include following the user around an area by using cameras, or using physical devices like smart cards or smart wearables that the user should carry and the system could sense their presence.

Such systems could greatly increase security as users are being authenticated and monitored constantly. However, since additional hardware is required, such systems may be expensive to implement. Also, users must always carry their authentication devices which may cause interruptions in their productivity in cases where the users forget their devices or the devices malfunction. Furthermore, privacy concerns may be raised, as users are being constantly monitored, and different biometric attributes may be measured.

On the other hand, keystroke dynamics authentication could provide a way to continuously authenticate users, while eliminating the need of additional hardware or any intervention from the users. Such a system tracks the keystrokes of a user, and uses them for extracting different features that help it learn the user's typing pattern. This system also provides flexibility, since it is a software implementation, it can be integrated directly into operating systems or applications.

4.3.2 Data Sources

As described earlier in the project, in section 4.2, measuring time differences is the main approach taken when collecting information about the interaction between the user and a keyboard. Even

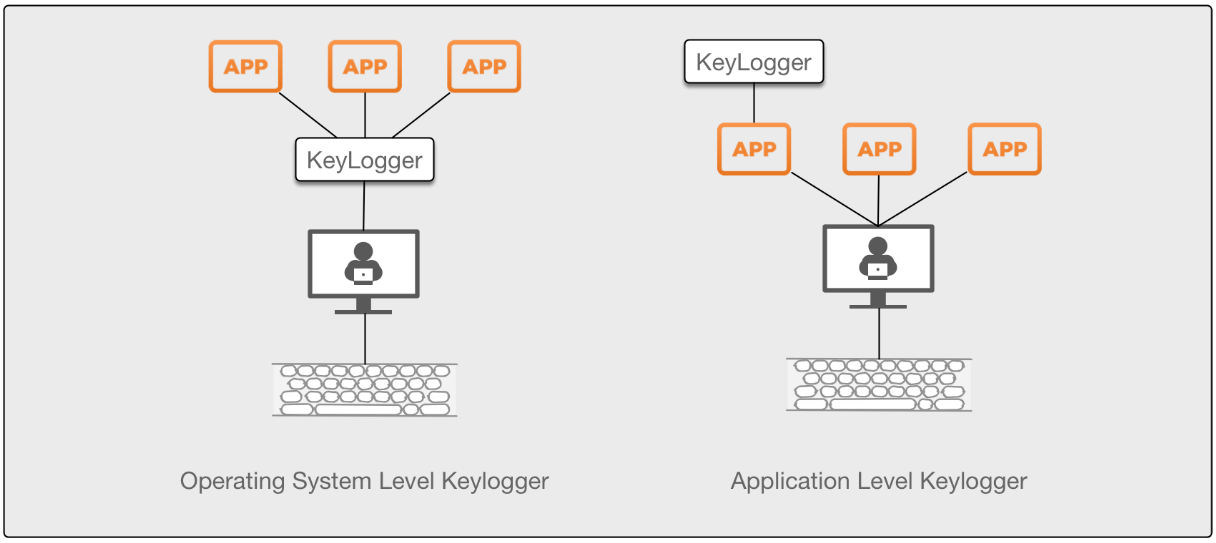


Figure 4.4: Different ways of capturing user keystrokes. An OS level keylogger intercepts all the keystrokes before they reach the active application, while the application level keylogger records only keystrokes performed inside the application.

though studies have been made for measuring pressure, modern day keyboards don't have this possibility. Mainly, the only events that can be registered by a keyboard are the pressing and depressing of a key. However, as it will be discussed in section 4.4.2, measuring the time difference between keystrokes may provide features that are sufficient enough to be able to identify a user.

In order to capture the keystrokes information, a way of intercepting the user's keystrokes must be implemented. This action is also refereed to as keylogging. For example, a text box on a website, that collects keystroke information for the text that a user types inside it. This approach could be very useful in the case of static authentication, as website administrators can implement keystroke dynamics as a second means of authentication. They could collect keystroke statistics when the user introduces the user name and password. Also, applications can be developed that, whenever they receive a keystroke from the user, they can register it as an event. However, while this approaches may be efficient, they are not satisfying the goal of having continuous and transparent authentication. In this case, keystrokes should be continuously recorded, on OS level, independent of the applications being used. This functionality is illustrated in figure 4.4, where both cases of an application level and OS level keyloggers are shown.

A keylogger is a piece of software that can implement a hook into the operating system, transparently intercepting events that are sent by the computer's I/O devices, the keyboard in this case. It could provide useful to satisfy the objective of this project. When a number of keystrokes are registered, the keylogger can send the collected data to the next step for analysis. Since it is an operating system level keylogger, implementations may vary depending on the OS, so before implementing such a software, it should be clarified on which operating systems it will run.

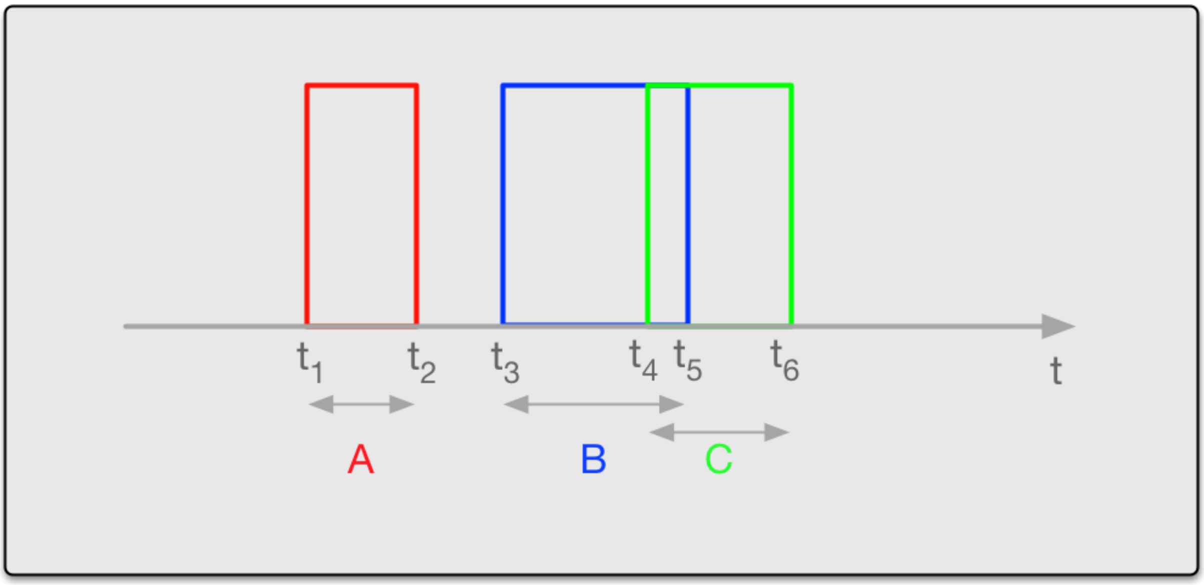


Figure 4.5: Recording information about keystrokes from the moment a key is pressed until it is depressed.

In order to provide useful information regarding a user's typing patter, such a keylogger should be able to record when a key is pressed or depressed. Furthermore, it should register the name of the key that is pressed and add a timestamp to the event. This information can be used later to extract features. Figure 4.5 illustrates the temporal sequence of keystrokes being pressed by a user. Two timestamps can be registered, the press and depress time, while the time between them can be considered the dwell time. As in can be observed, it is also possible that one key is pressed before the previous key is depressed, causing an overlap.

However, since it does record all the inputs of the user, privacy and security are important factors, especially that keyloggers are usually seen as threats by several security systems, as they can be used to steal passwords, bank details or other sensitive information. Users are generally reluctant when it comes key logging software. It is because of these facts that a keylogger should be carefully designed and the user properly informed of it's functionality.

4.4 Features

After the human-keyboard interaction is converted from a physical action to data that can be processed by a computer, features have to be extracted from this data that would uniquely describe each user. Since the system should be able to recognize free text, meaning that there is little chance that the user will type the same sequences twice, flexible features that are applicable to any text should be used. One solution, as presented in most of the literature, summarized in section 3.2.3, is to group multiple letters and calculate the time differences between them as illustrated in figure 4.6. A group of two letters is referred to digraph while groups of three or four letters are referred to as trigraphs or quadgraphs. In the following sections, ways of filtering

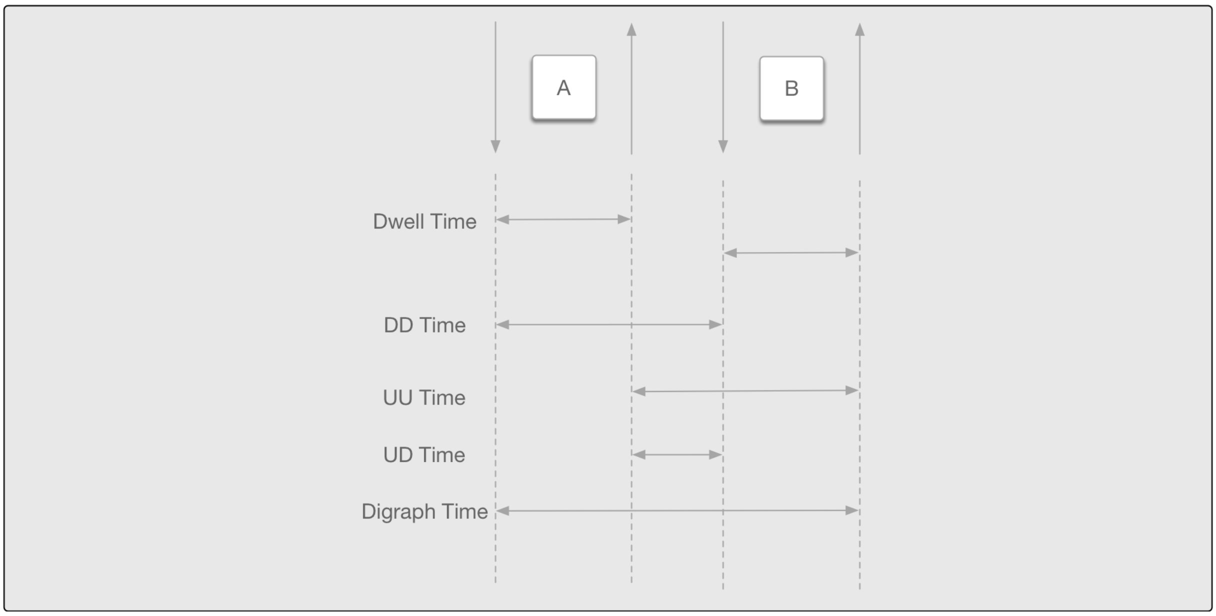


Figure 4.6: Digraph Measurements

the data collected by the keylogger in order to eliminate noise, as well as how to organize it in features are described. The temporal sequence of the tracked characters has to be removed by processing, as well, for ensuring privacy. This section aims to provide answers for the third subquestion introduced in section 1.3 (*"Which features should be measured in order to uniquely identify a person?"*).

4.4.1 Information Filtering

As described in section 4.2, behavioral biometrics measurements are subject to lots of noise. In order to try to minimize the noise and it's impact on the authentication decision, filtering has to be applied to the dataset before features can be extracted.

It is normal that when typing long lines of text, users will take pauses at random points in time. This will result in a long time difference between keystrokes that can affect the performance of the classification algorithm. A ceiling should be introduced in order to eliminate long pauses. Different papers propose filters for solving this issue. Digraph measurements are introduced in figure 4.6. The values that are proposed are varying around 600 to 800 ms for digraph latency (UD time) [31]. These values are used as a reference, however tests on the final product should be run in order to find the ideal threshold. A minimal value of 10 ms is also proposed in [31], as it eliminates the cases where users might press two buttons at the same time by mistake. From real life testing, however, it was deducted that when measuring the digraph latency (UD time), it happens several times that a key is released after the next key is pressed. This would result in a negative digraph latency which would be eliminated if a low filter is applied. This filter can be applied after the information is received from the keylogger and pre processed. This may result in samples being dropped and has to be taken into account when setting the number at which the keylogger triggers an analysis.

Furthermore, when a key is pressed for a long time, the operating system will start sending multiple repetitions of the corresponding key. This will result in digraphs containing the same letters to be recorded for multiple times. According to English Oxford Dictionary, there are no words in English that contain the same letter more than twice in a row. In this case, whenever a letter appears for more than two times in a row, the corresponding digraphs will be ignored.

Moreover, as keys that don't represent alphabetical characters, like delete, control, alt, backspace, arrows, etc. may be used on a random basis, a filter should be applied in order to collect only letters. It would make sense to implement this kind of filter directly into the keylogger, as it would reduce the number of discarded samples in the pre processing phase as well as using the memory in a more efficient way.

4.4.2 Feature Extraction

The data collected by the key logger is a stream of events which includes the action, the key and the time stamp of each event. As described in the previous sections, the data needs to be pre-processed in order to be useful for the classification algorithm as well as ensure the privacy of the user. As introduced in the State of the Art chapter 3, several features, that can be extracted, are introduced in the literature.

Some of the literature suggests collecting digraph data, together with trigraph data for better results [31][13]. However, in paper [13], experiments showed that the user may type the same digraph differently, depending on the context. It was proven that there is a considerable difference in the speed with which a user types a digraph, based on how familiar the phrase that they are writing is. As a result, the author suggests extending the data capture to trigraphs and quadgraphs in order to include contextual information.

When using trigraphs or quadgraphs, however, there is a risk that the processing of data will affect the performance of the user's computer, because of memory and cpu usage. The necessity of performing additional measurements, as well as storing and transferring bigger amounts of data might impact the system.

On the other hand, other than providing more context to the samples, using trigraphs and quadgraphs increases the number of possible measurements to be performed, which may result in multiple possible features to be extracted. Using the combinations formula

$$C_n^k = \frac{n!}{k! * (n-k)!} \tag{4.1}$$

it can be calculated how many measurements can be performed. The n in the formula represents double the number of keystrokes taken into consideration (as two events are recorded per keystroke), while the k will always be 2 as the measurements are performed between a start time and an end time. Applying the formula, it can be concluded that using digraphs, 6 measurements can be performed, as illustrated in figure 4.6, while using trigraphs 15 measurements are possible and in the case of quadgraphs 28 combinations are possible.

In the paper [31], the authors compare the precision of a keystroke dynamics authentication system when using digraphs, trigraphs and words. Although the conclusion is that the effectiveness of the techniques depends a lot on the user, the overall results showed that measuring digraphs provided the smallest FAR, of around %5. However, the paper concludes that using digraph data alone is not enough for a live system and multiple metrics from digraphs and trigraphs, as well as other sources, should be combined together.

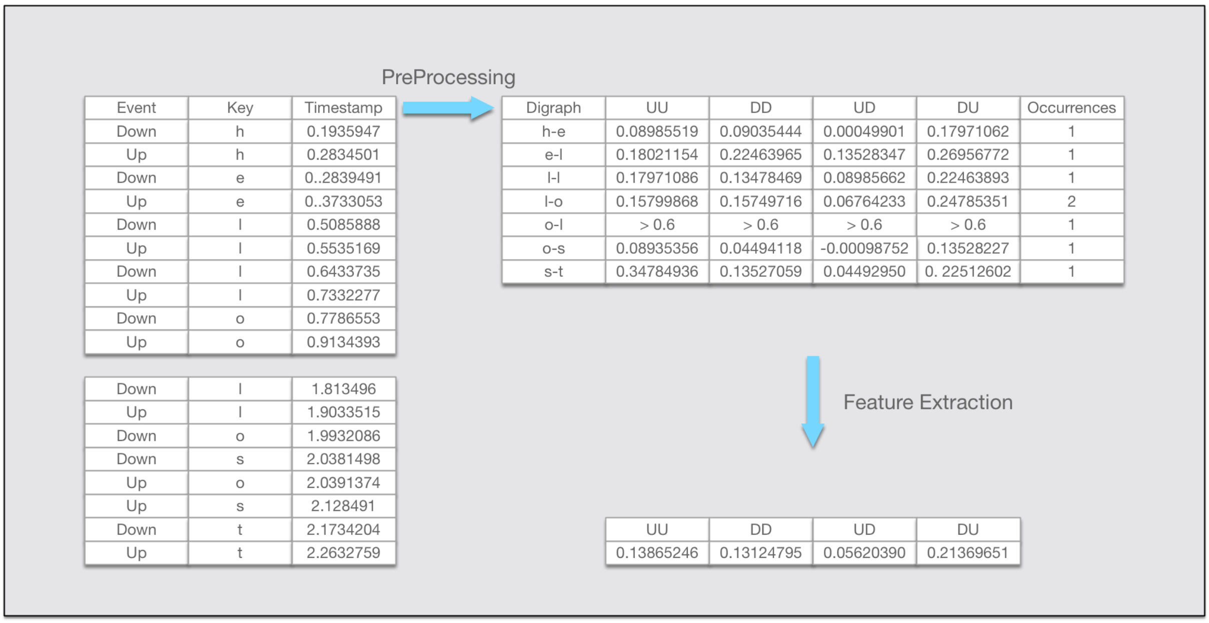
Such an idea is approached in [8], where the author uses the features described in the [31] paper and combines with mouse dynamics and GUI interaction features in order to obtain FAR of 2.24% and FRR of 2.10%. Also, in the paper [15], a user adaptive approach is taken where the features are given different weights for each user depending on the frequency that they appear and the speed they are typed at obtaining results as good as 1.24%. Paper [13], proposed features to be collected from trigraphs for inter-key times, as well as quadgraphs for hold times.

After measuring the data, it has to be processed in a form that can be useful to the classification algorithm. A keyboard with 104 keys, results in 10816 possible digraph combinations, most of which are never used [8]. One approach, applied in paper [12], is to use the most frequent digraphs appearing for a language, 20 digraphs in Polish for this case as features. In paper [31], all the digraphs and trigraphs are logged, and a limit variance is configured. The digraphs and trigraphs with a big variance or that don't have any values are removed.

The paper [15] takes a user adaptive approach to processing digraph data. The digraph data is sorted based on the frequency of appearance, speed and variance. Then, the data is split into eight features. The first 1/8 of the data going in the first feature, by calculating the mean value, next 1/8 to the second feature and so on. Each feature is, then, assigned a weight.

In [13], multi-dimensional matrices are used for storing the statistics. For example, in the case of trigraphs, a three dimensional matrix is used, with each axis corresponding to the letters. The data for the trigraph is stored at the intersection of the three letters.

By using any these approaches, the possibility of recovering a chronological log of keystrokes is lost, thereby improving privacy.



Event	Key	Timestamp
Down	h	0.1935947
Up	h	0.2834501
Down	e	0..2839491
Up	e	0..3733053
Down	l	0.5085888
Up	l	0.5535169
Down	l	0.6433735
Up	l	0.7332277
Down	o	0.7786553
Up	o	0.9134393
Down	l	1.813496
Up	l	1.9033515
Down	o	1.9932086
Down	s	2.0381498
Up	o	2.0391374
Up	s	2.128491
Down	t	2.1734204
Up	t	2.2632759

Digraph	UU	DD	UD	DU	Occurrences
h-e	0.08985519	0.09035444	0.00049901	0.17971062	1
e-l	0.18021154	0.22463965	0.13528347	0.26956772	1
l-l	0.17971086	0.13478469	0.08985662	0.22463893	1
l-o	0.15799868	0.15749716	0.06764233	0.24785351	2
o-l	> 0.6	> 0.6	> 0.6	> 0.6	1
o-s	0.08935356	0.04494118	-0.00098752	0.13528227	1
s-t	0.34784936	0.13527059	0.04492950	0. 22512602	1

UU	DD	UD	DU
0.13865246	0.13124795	0.05620390	0.21369651

Figure 4.7: Feature Extraction Process

4.4.3 Feature Extraction Considerations

By using an agile approach on the system development, incorporation of features can be done in several stages. Measurements are performed after incorporating new features in order to evaluate the impact on the performance and decide whether a feature should be kept in the product for future tests or be removed. For this matter, the system should be built in such a manner that it offers the flexibility to add and remove features without impacting the rest of the functionality in the system, using the OOP approach. Moreover, since the keystrokes of the users are logged continuously and features are extracted when enough keys are logged, impact on system resources should also be considered. Depending on the method chosen, feature extraction can be a resource demanding process so a balance between simplicity, resource usage and machine learning algorithm performance should be considered. Furthermore, in order not to lose samples, since the feature extraction process could be time consuming, these two processes should run in parallel and not sequentially.

The process considered for this project is displayed in figure 4.7. in this example, the words "hello lost" are written by the user. At the first step, information for each keystroke is captured by the keylogger, indicating the action "Up" or "Down", the actual key that was pressed and the timestamp. As discussed in section 4.4.1, only alphabetical characters are being recorder while all other keystrokes are ignored. The timestamp in this example is simplified for illustration purposes. In a normal case, the timestamp is returned by the operating system, usually in Unix Epoch time. Unix time is a system that describes a point of time by the number of seconds elapsed since 1 January 1970. For example, the actual timestamp for "Down", "h" is

1527423987.1935947 seconds, which would translate to Sunday, May 27, 2018 12:26:27.193 PM GMT.

As it is implied, such a timestamp may provide a lot of information, so in order to ensure privacy, during the pre-processing step, the data is arranged in digraphs and only the time differences between the characters are stored. This way, it is not possible to trace back the exact moment of typing each letter. In the case where a digraph appears more than once, the average value is calculated for the timing information and the number of occurrences is recorded. As this is a very small example dataset, there is only one digraph that occurs twice ("l-o"), but in a real case scenario, repeated occurrences are very probable. There may be cases where one digraph appears a number of times but the timing data has a high dispersion, resulting in a mean value that is not very representative. For this reason, together with the mean value for each digraph, the standard deviation should also be calculated. This standard deviation can be used for eliminating digraphs that have high sparsity, or use it as a feature for the machine learning algorithm.

As it can be observed in figure 4.4.2, it is still possible to decode what the user was typing since the digraphs are in order. However, ordering the digraphs by the occurrence times or just randomly, during pre processing, would strengthen privacy of the user even more, as the temporal order would be lost. Furthermore, as discussed in section, 4.4.1, values higher than 600 ms are considered as breaks in typing and are ignored. In this case, there is a break made by the writer between the two words. Also, as it can be observed in the "o-s" digraph case, there is also possible to have negative values in the UD case. This appears when the user presses a key before releasing the previous one.

As a first step of implementation, four features are considered for the machine learning algorithm. In this way, the mean values of the DD time, UU time, UD time and digraph time are calculated, and the results are used as features. As it can be observed in the last step in figure 4.4.2, the pre-processed data is used for feature extraction. Four features are extracted, corresponding to UU, DD, UD, DU timings for each user. These features are then used to train the machine learning algorithm and later to authenticate users.

4.5 Machine Learning

In this section the machine learning approaches, discussed in State of the Art chapter, section 3.3, are examined in accordance to the objective of the project. Different machine learning approaches and ways to train algorithms are introduced, as well as which algorithms may fit the keystroke dynamics authentication system. This sections aims to answer the fourth subquestion, introduced in section 1.3 (*"How to use machine learning algorithms in order to adapt to and detect the person's typing pattern?"*).

4.5.1 Machine Learning Approach

While collecting the right features and organizing them in the right way impacts the performance of the keystroke dynamics authentication system, choosing the right classification algorithm can also provide better results. As described in the State of the Art chapter 3, two main approaches have been used as a starting point in the literature. Some approaches rely on techniques based on neural networks, while others are statistical in nature. However, further categories are introduced like pattern recognition techniques or hybrid techniques [41].

Generally, statistical classifiers are described as comparing distances between two sets of data, one reference set that is trained beforehand, and a new test set. If the distance falls within a threshold, the user is recognized as legitimate. Initial works on this field, like in paper [11] from the year 1980, had a purely statistical approach with calculating distances on plots and manual inspection. Neural networks approach is using the historical data to build a model, that is then used to predict the outcome of new samples. While there is no consensus in literature on which method provides better results, plenty of experiments were performed using classification algorithms that use machine learning techniques for automated learning. These techniques soften the gap between the two categories as classification methods are using statistical models to some degree.

Furthermore, depending on the way algorithms function, there are two main approaches presented in the literature [13]. One approach is to train a classifier in a one versus all fashion, which discriminates between a valid user and all others. Such algorithms require samples from both a valid user as well as negative samples and are also known as binary algorithms. Another approach, is to use anomaly detection algorithms which require only samples from a particular user and should detect changes that may appear. Since, in this project, the system should be dealing with multiple users the first approach could be easily implemented where the active user is considered valid while all the others can be considered negative samples.

These two approaches are very similar to the concepts of identification and authentication, as presented in section 1.2. While it can be inferred that the first method, where classification us performed on all users is more of an identification approach, compared with anomaly detection algorithms which are only focused on the authenticated user, the ultimate goal of the system to provide authentication can still be achieved with both methods. For the classification method, the system can check if the predicted user matches with the active user and with what degree of certainty.

Moreover, since an algorithm should handle data from multiple users, it makes sense to use a centralized server. It would not only make the maintenance of the algorithm easier but it could allow companies to properly protect the stored data, away from the user's computers. A secured server inside an enterprise would ensure proper confidentiality of the data. Furthermore, since the machine learning algorithm may need to be retrained periodically and it may require high amounts of memory and processing power, having a dedicated machine for handling this kind of tasks becomes a requirement.

4.5.2 Machine Learning Algorithms

As introduced in section 3.3, different types of algorithms are used for solving different kind of problems. The first step into choosing the right machine learning algorithm is to understand what the inputs and the outputs of the algorithm should be. In the case of keystroke dynamics authentication, the input would be the features collected from a user that is known. This way, the features used for training the algorithm are accompanied with the label of the user that they belong to. The presence of a label, together with the samples, points to a supervised learning problem.

The next step is to look at the output that the system should provide and decide whether the problem is a regression problem or a classification one. In the case of the current project, the output of the system should be the name of the user that is most probable the provided features belong to. This way, the system is dealing with a number of known labels, in which case using a classification algorithm makes sense.

As a next step, the available algorithms that fulfill these requirements should be identified. There are multiple types of algorithms that could be used, as introduced in section 3.3.1. Probabilistic methods like Logistic Regression or Naive Bayes, geometric methods like K Nearest Neighbor or Support Vector Machines, or entropy methods like Decision Trees can be used for the case.

There doesn't seem to be a standard accepted way of choosing one type of algorithm as performance may greatly vary depending on the input data as well as the implementation of the system. Best practice shows that multiple algorithms should be implemented and their performance should be compared. Algorithms can be scored using a method like K-Folds cross validation, by using a part of the training dataset as test dataset. The algorithm that provides the best results can then be used in the system. Furthermore, depending on the algorithm, it can be fine tuned in order to provide better results.

Furthermore, in order to improve the performance of the algorithm as well as adapt it to the changing behavior of users, a way should be considered to periodically retrain the algorithm. A sliding window can be used where only the newest features are kept. Still, since mean values are used as features, the oldest values could be averaged from multiple feature vectors into one feature vector. Since the algorithm has to be trained periodically for new users, new features for already registered users can be used as well.

4.6 Performance of the System

In this section the testing approaches in order to calculate the performance of the system are discussed. This section aims to answer the fifth subquestion, introduced in section 1.3 ("*How to*

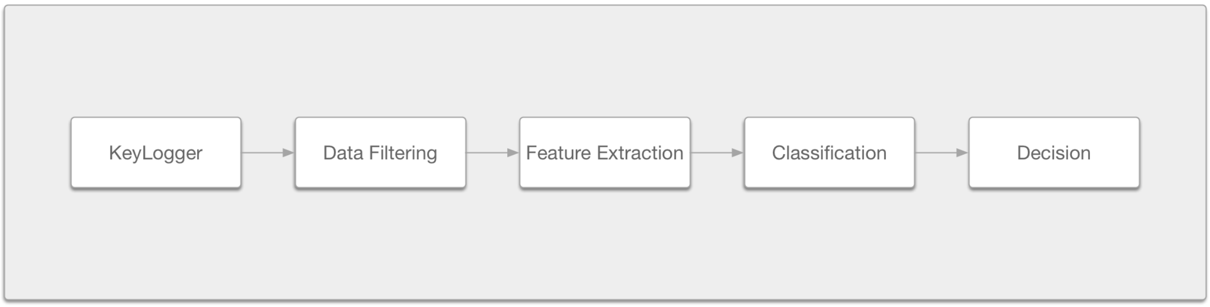


Figure 4.8: Keystroke Dynamics Process Steps

calculate the performance of the system?").

As introduced in section 3.3.4, K folds cross validation is a recommended approach for calculating the performance of the machine learning algorithm. Since choosing the best machine learning algorithm is a question of trial and error, as discussed in section 4.5, a test system should be implemented that runs K-Folds cross validation on different machine learning algorithms using the available user data. The algorithm with the best score, that satisfies the requirements of the system should be implemented in a production scenario.

Moreover, as introduced in the State of the Art chapter 3, section 3.2.3, in order to measure the quality of a keystroke dynamics authentication system, the FAR (False Acceptance Rate) and FRR (False Rejection Rate) should be calculated on the final product. These tests should be run in a real environment, where users are tracked while doing normal daily work in normal conditions, in order to provide meaningful results.

4.7 Summary

It can be concluded from the analysis section that a keystroke dynamics authentication system can bring security benefits to users and enterprises if tuned and designed appropriately. Five major elements have been identified that build such a system. These elements are illustrated in figure 4.8.

The first element is a keylogger that transforms the user's physical interaction with their keyboard into electronic data that can be used by computers. This module collects information about each keystroke and filters out the non-alphabetical keys. This information is then sent to the second element, data filtering or pre-processing phase, which involves cleaning up the data in order to minimize noise. This module should eliminate measurements resulted from long user breaks in typing or repetitions of keys resulted from keeping a key pressed for too long.

In the third step, after the information is filtered, features are being extracted. The initial proposal is to extract four features by calculating the mean values of UU, DD, UD and DU times. If the results are not satisfactory, integration of other features can be considered. The features are then sent to the fourth module, which performs classification. This module is firstly trained for each user, and should then provide probabilities for each sample, corresponding to each user. In the last step, the prediction of the algorithm is presented to an entity that can take a decision of regarding the user. All these modules should be lightly coupled in order to allow changes to any of the modules without affecting the others. Based on the discussions presented in this chapter, requirements are going to be built and are introduced in the next section.

4.8 Requirements

Based on the analysis performed in this chapter, a list of requirements is built and presented in table 4.1. Each of the requirements is described and a rationale is provided. Additionally, the requirements are prioritized using the MosCoW model. As requirements can be either functional or non-functional, the name of each requirement will be prefixed with FR (functional requirement) or NFR (non-functional requirement). The section of the analysis where the discussion leading to the requirement is performed is introduced in the last column.

Table 4.1: Requirements Table

Req	Function	Description	Rationale	Priority	Section
FR_01	System Architecture	In order to avoid processing and memory intensive tasks and have a centralized database of users, the system should function in a server client model, where the client collects the required data and sends it to a server for machine learning processing.	It should be implemented in order to provide full functionality.	Could	4.1

Table 4.1 continued from previous page

FR_02	Authentication	The system should perform continuous authentication of the users by analyzing keystroke samples and predicting with a certainty score to which users the keystrokes belong. the authentication is considered successful if the prediction corresponds to the active user, with a score above a set threshold.	It must be implemented in order to provide input for the system.	Must	4.1
FR_03	Key Logging	The system should be able to continuously record the keystrokes of the user, regardless of the application that they are using. Only alphabetical characters should be recorded. The action "Up" or "Down", letter pressed and timestamp should be recorded.	It must be implemented in order to provide input for the system.	Must	4.3
FR_04	Digraph Measurement	The system should process the recorded data to organize captured events in digraphs and collect digraph statistics from the recorded data under the form of UU, DD, UD, DU times as shown in figure 4.6. When a digraph appears multiple times, the mean value and standard deviation of the times should be calculated.	It must be implemented in order to provide input for the features to be measured.	Must	4.4

Table 4.1 continued from previous page

FR_05	Trigraph Measurement	The system should process the recorded data to organize captured events in trigraphs and collect trigraph statistics from the recorded data.	It could be implemented in order to improve the accuracy of the classification algorithm.	Could	4.4
FR_06	Quadgraph Measurement	The system should process the recorded data to organize captured events in quadgraphs and collect quadgraph statistics from the recorded data.	It could be implemented in order to improve the accuracy of the classification algorithm.	Won’t	4.4
FR_07	Data Filtering	Digraphs that contain time differences higher than 600 ms should be filtered out. Trigraphs and quadgraphs corresponding to the digraph should be also filtered out.	It must be implemented to eliminate noise and improve the accuracy of the classification algorithm.	Must	4.4
FR_08	Feature Extraction	The system should extract features from the collected statistics that can be used by the classification algorithm. Initially, average values for each UU, DD, UD and DU measurements are used to build features but the system should allow easy expansion of features.	It must be implemented in order to provide features for the classification algorithm.	Must	4.4
FR_09	User Creation	The system should check if the active user is a new or existing user and create a new profile in the case of new users.	It should be implemented in order to register new users in the system.	Should	4.4

Table 4.1 continued from previous page

FR_10	Classifier Training	The classification algorithm should be trained periodically at low traffic times. When a new user is detected, the machine learning algorithm should be trained for the new user. The system should save features for this user until the next training session.	It must be implemented in order to be able to train the classification algorithm.	Must	4.5
FR_11	Classification	The system should be able to classify users based on the samples that it receives from the client and return a confidence score for the user. The number of keystrokes required to build features should be decided through testing.	It must be implemented in order to provide the classifier with features for authentication.	Must	4.5
FR_12	Alert Trigger	The system should send an alert to a security center when an intruder is detected. The actions to be performed on the client machine when an intruder is detected are environment dependent so they are out of scope.	It could be implemented in order to prove that the system can raise alerts when intruders are found.	Could	4.1
FR_13	Parallel Processing	The keylogger should be recording at all times. Data pre processing and feature extraction should be done in parallel on the client.	It could be implemented in order to avoid missing keystrokes.	Could	4.4

Table 4.1 continued from previous page

FR_14	Server Listening	The server should be always listening for requests for the client and act upon them as soon as they are received.	It cloud be implemented in order to embrace the client-server architecture.	Could	4.1
FR_15	Active Timers	When a user comes online the server should start an active timer of 15 minutes and reset it whenever it receives a valid feature vector. If the timer reaches 0 an alarm should be raised.	It won't be implemented as it can be considered an extra security feature. The main purpose of the current stage of the project is to demonstrate the functionality and performance of the proposed system..	Won't	4.1
FR_16	Send Timers	A timer of 10 minutes should be implemented on the client. If not enough keystrokes have been collected, a feature vector should be built using the available collected keystrokes and this fact should be signaled to the server.	It won't be implemented as it can be considered an extra security feature. The main purpose of the current stage of the project is to demonstrate the functionality and performance of the proposed system.	Won't	4.1

Table 4.1 continued from previous page

FR_17	User Logout	The client should inform the server when the user finishes their activity and log out from their operating system. The server will remove any timers and sessions related to the client.	It could be implemented if the client server approach is implemented.	Could	4.1
FR_18	Adaptive Training	The system should save the features of users in a sliding window mode and user them for training in the upcoming training session of the machine learning algorithm.	It could be implemented in order to be able to adapt to the changing behavior of the users.	Could	4.5
NFR_01	Confidentiality	The data flowing in the system is considered sensitive data so the confidentiality of the data should be ensured for throughout the whole lifecycle. Appropiate security measures should be applied on client, server and communication between them.	It should be implemented to comply with the regulations enforced on the market. It won't be implemented in the beginning stages of the development project.	Won't	4.1
NFR_02	Privacy	The system should ensure the privacy of the user. All personal data on the user's computer as well as data sent to the server should be sufficiently protected and only the personal data necessary for fulfilling the authentication process is collected.	It should be implemented in order to comply with regulations. Some privacy aspects are already discussed in this project.	Should	4.1

Table 4.1 continued from previous page

NFR_03	Transparent Process	The process should start automatically when the operating system boots up. It should not require any intervention from the user.	It could be implemented so the whole functioning of the system is transparent to the user.	Could	4.3
NFR_04	Data Processing	The data should be pre-processed on the client side before being sent to the server for classification. The data resulted from the pre-processing phase should have context removed so it would be impossible to trace back the exact text written by the user.	It could be implemented to provide a centralized server, with enough resources to run classification algorithms.	Could	4.4
NFR_05	OOP Approach	In order to allow for easy upgrade of any each module, the software should be build using OOP approach.	It must be implemented to allow of upgradeability in the future.	Must	4.4
NFR_06	OS and Language	The system should be created for Windows OS and tested on English language samples.	It must be implemented to allow functionality and testing.	Must	4.3
NFR_07	Testing Module	The system should contain a testing module where the performance of different machine learning algorithms can be tested. The impact of using different features can also be tested with this module	It should be implemented to allow for system testing.	Should	4.6

5| System Design

In this chapter, the design of the system will be introduced based on the requirements gathered in table 4.1. The overall functionality of the system will be presented, followed by a high level design, a detailed description for each entity's functionality and sequence diagrams that describe the interactions between entities.

5.1 System Description

The main purpose of the system is to provide continuous authentication in a transparent manner while protecting the user's privacy. The system learns the typing pattern of the users by gathering initial typing samples and builds a profile for each user. The profiles are used to evaluate upcoming samples from typing activities and provide a certainty score on the user's claim validity.

The process automatically starts on boot and is running in the background of the operating system. Its functionality is transparent to the user and it does not interfere with their work. The user should, however, be aware that their typing activity is being logged and the purpose for it be clearly stated in order to align with the privacy by design principles.

The data is collected and pre-processed locally on the user's PC in a way that it loses context. The data is then sent to a server for further processing. In the case the classification algorithm detects an intruder, an event will be triggered. The event can be used to perform different actions like locking the user out or sending an alert.

5.1.1 Preconditions

In order for the system to function correctly, certain preconditions need to be fulfilled. It is mandatory that a primary authentication method is used in addition to the keystroke dynamics authentication. The keystroke authentication system automatically starts running when the user successfully authenticates in the operating system.

The users should not stop the process that is running in the background, in order for the continuous authentication to work. Furthermore, for the initial version, the system is only developed and tuned for the English language and running on Windows OS as per requirement NFR_06.

When using a server client scenario, it is mandatory that the communication between the entities is working. Furthermore, both of them should meet the minimal system requirements.

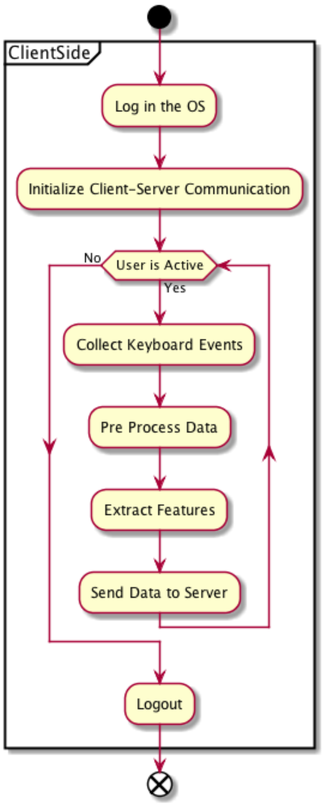


Figure 5.1: Activities performed on the user's client from the moment the user comes online until they logout. The activities ran when the user is active are running in a loop.

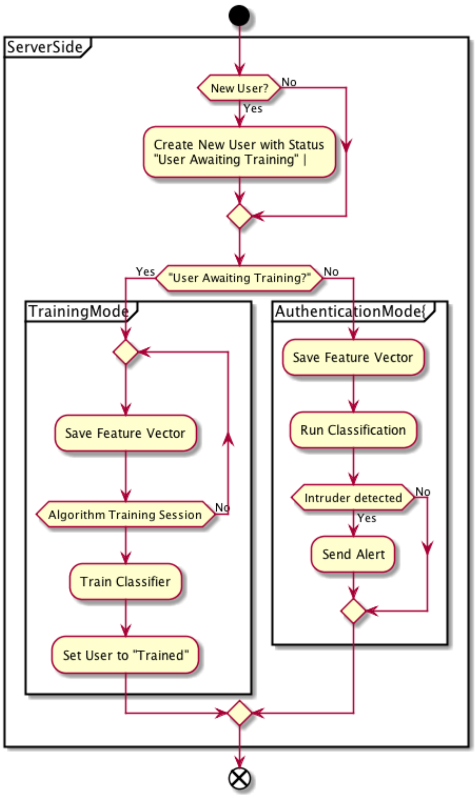


Figure 5.2: Activities ran on the server when it receives a user's feature vector. Depending on the user status, the server can run in two modes, training mode or authentication mode.

5.1.2 Postconditions

The system has successfully finish its job when it checks whether the user predicted by the machine learning algorithm is the right user and the probability for this prediction is over 80%. The system can then be used to trigger an alarm. The action taken upon the user when an alarm is raised is not implemented within the system and a target system that performs the corresponding action should be able to use the keystroke authentication system's output as a trigger for actions.

5.2 System Activity Flow

Based on the analysis performed in chapter 4, the functionality of the system can be described. Using UML activity diagrams, the series of actions that the client and the server must perform are introduced. The diagrams are illustrated in figures 5.1 and 5.2. This section includes a high level description of the system functionality. In the upcoming sections, each entity performing these functional steps is described in detail.

The system starts running as a background process when the user logs into the operating system, according to NFR_03. According to requirement FR_01, the system is built in a client server architecture. The user's computer, acting as a client, initializes a communication with the server. This action should inform the server that the user became active.

5.2.1 Activities on the Client

The main objective of the client is to collect information about the user's keystrokes, and process this information into feature vectors. The feature vectors should be then sent to a server for processing.

In the first activity, the system uses a keylogger that provides a hook into Windows OS which intercepts keyboard events. The number of events that should be tracked is 400, which corresponds to 200 keystrokes as both up and down events are recorded. The data is recorded under the following format: [<action (UP/DOWN)>, <key>, <timestamp>]. This way, for each key that is pressed two events will be registered, one for pressing the key ("DOWN") and one for depressing the key ("UP"), together with the name of the key and the timestamp of the event. Furthermore, according to FR_03, only key presses corresponding to alphabetical keys will be recorded.

The second activity is to pre process the data, according to requirement FR_04, where each two consecutive letters are arranged in groups of digraphs. The UU (up-up), DD (down-down), UD (up-down) and DU (down-up) times, illustrated in figure 4.6, are calculated. Data filtering is also performed at this step, according to requirement FR_07, where digraphs that contain times higher than 600 ms are excluded. In the case where the same digraph occurs multiple times, the mean values are calculated for each measured time corresponding to the digraph, as well as the standard deviations for each digraph. The output from this activity will have the following format: [<digraph>, <mean value>, <standard deviation>, <occurences>], for each digraph four outputs will be produced, one for each measured time.

In the "Extract Features" activity, using the output from the pre processing step, features are extracted. According to requirement FR_08, initially the features will consist of the mean values for the UU, DD, UD, DU times from all the digraphs. However, according to requirement NFR_05, this step will be implemented completely independent from the previous extract features step, such that additional features can be added in the future without impacting the functionality of the system. The output after this step is a feature vector and has the following format: [<UU mean value>, <DD mean value>, <UD mean value>, <DU mean value>].

After the feature vector is built, it is sent to the server for further processing. The whole process is repeated for as long as the user is active.

5.2.2 Activities on the Server

The server is always listening for incoming feature vectors, as per requirement FR_14. The client should also inform the server about the username of the active user. Using this information, the server checks if a profile for the user exists. In the case it does not, a new user is created according to requirement FR_09, and the status of the user is set to "User awaiting training". In the next step, the server checks if the active user's status is "User awaiting training". If this condition is true, the server will run into training mode. Otherwise, it will run into authentication mode.

Training Mode

The purpose of the training mode is to train the machine learning algorithm for new users, according to requirement FR_10.

The algorithm should run periodical training sessions at times when the traffic is low. For example, the training sessions could be run every day at midnight. While the server is in training mode, the keystroke dynamics authentication process is suspended.

Since the system is running in training mode, several features should be collected from a user. While the status of the user is "Awaiting Training", the server saves all the feature vectors for the user. When the next training session is performed, if enough features have been collected for the user, they are included in the training session and the user status is set to "Trained". Otherwise, more features will be collected until the next training phase.

Authentication mode

If the user for which the authentication is performed has "Trained" status„ the system will go into authentication mode. This mode fulfills the requirements FR_11 and FR_18.

The server saves the received feature vector in the user's profile for future training of the algorithm, as per requirement FR_18, and runs the classification algorithm for the received feature. The output of the algorithm is the probability for each user that the sample belongs to them. If the user predicted with the highest probability is the same as the active user, with a probability higher than 80%, the user is considered valid. Otherwise they will be considered as an intruder and a trigger will be sent to a monitoring or a security system , as introduced in FR_12.

5.3 System Architecture

The architecture of the system is depicted in figure 5.3. The system is built in a client-server approach, with a centralized server being connected to multiple clients, as per requirement FR_01. This way customer data can be securely stored, as well as, processing and memory

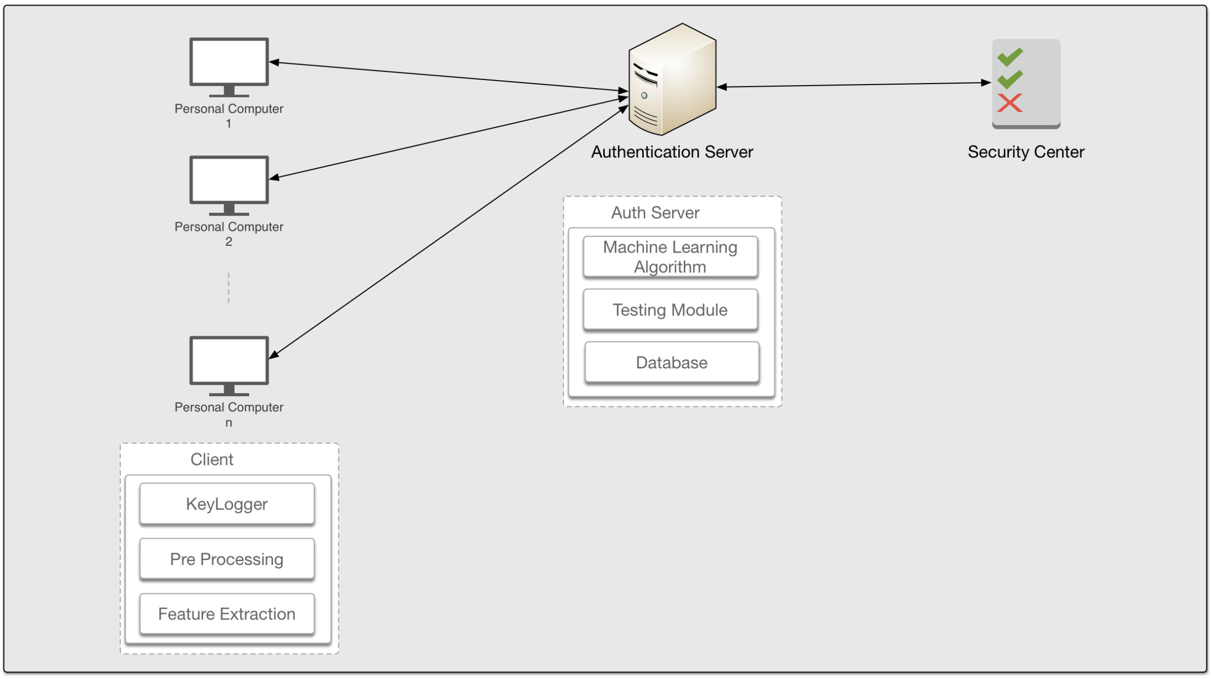


Figure 5.3: High level architecture of the system with the modules beloging to each entity illustrated underneath.

intensive tasks, like machine learning algorithms, are performed away from the users' computers. This client - server approach also provides easy maintenance and expandability in the future.

As depicted in figure 5.3, a number of computers are connecting to a centralized authentication server. For redundancy purposes, in order to avoid the single point of failure and increase the availability of the authentication service, backup servers can also be deployed. Each entity is described in detail below.

5.3.1 Client

As illustrated in figure 5.3, the client is represented by the user's computers. The objective of the client is to collect keystroke information about users and prepare features that can be sent to the server for authentication. There are three different modules running on the client, a keylogger, the pre processing module and the feature extraction module. Each module is built as loosely coupled as possible from the other modules in order to allow for easy modification and upgradeability, according to NFR_05.

Keylogger

As introduced in the requirement FR_03, the system should be able to record the user's typing activity. This function is implemented by using a keylogger. Since the implementation of a key

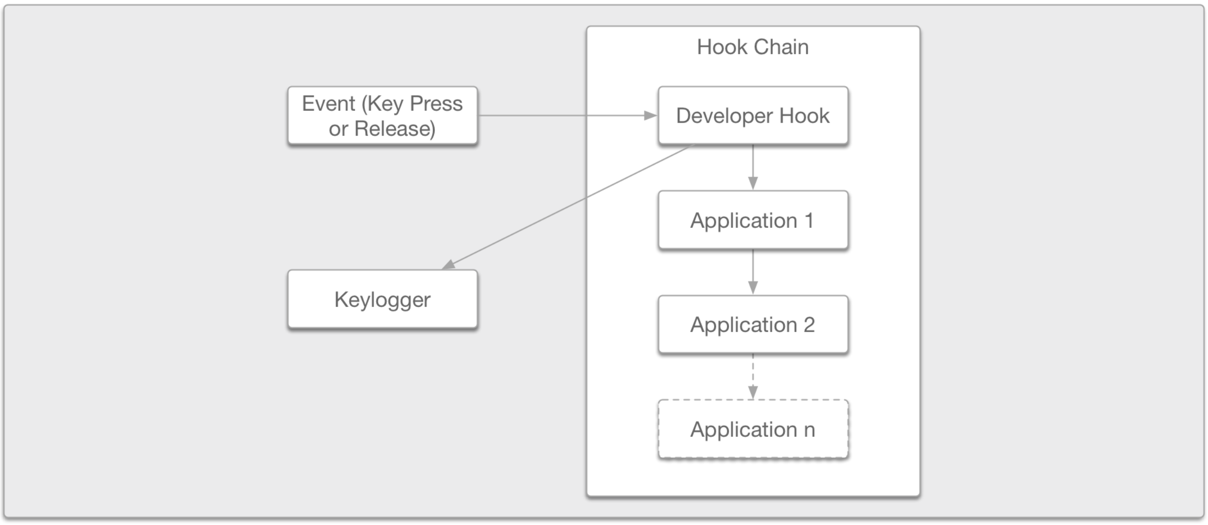


Figure 5.4: Windows hook mechanism illustrating a hook procedure, that intercepts keystrokes, being installed in a hook chain.

logger is dependant on the operating system, the system is mainly developed for Windows, given its popularity among organizations and the amount of support available.

The keylogger records the keystrokes of the users, marking down the action, which can be "UP" or "DOWN", the key that is pressed and the time stamp of the action. A down action is equivalent to the user pressing a key, while an up action refers to the user releasing the key.

According to the Microsoft Developer Network[42], The Windows operating system provides hooks in order to allow applications to intercept events such as messages, mouse actions, keystrokes, etc. Each type of hook has it's own hook chain which points to all the application level hook procedures. A hook procedure allows applications to act upon the event, allowing them to monitor, modify or even stop the advance of the event through the chain, preventing it from running the following hook events or reaching the target window.

Windows provides a hook procedure to developers which allows them to install it inside a chain. The procedure will take precedence over the other procedures in the chain so it will act first upon the event. This process is illustrated in figure 5.4. Windows offers the **WH_KEYBOARD** and **WH_KEYBOARD_LL** hooks which allow a developer or an application to monitor keyboard events.

The keylogger saves the keystrokes in a list, which is buffered until the required number of keystrokes is collected. When the number of keystrokes is collected, the system opens a new thread for the pre processing, and the list goes into pre processing. The keylogger restarts the recording, on an empty list, parallel to the pre processing and feature extraction process according to requirement FR_13. As introduced in section 5.2, the format of each element in the list is [<action (UP/DOWN)>, <button>, <time stamp>]. This process is illustrated in

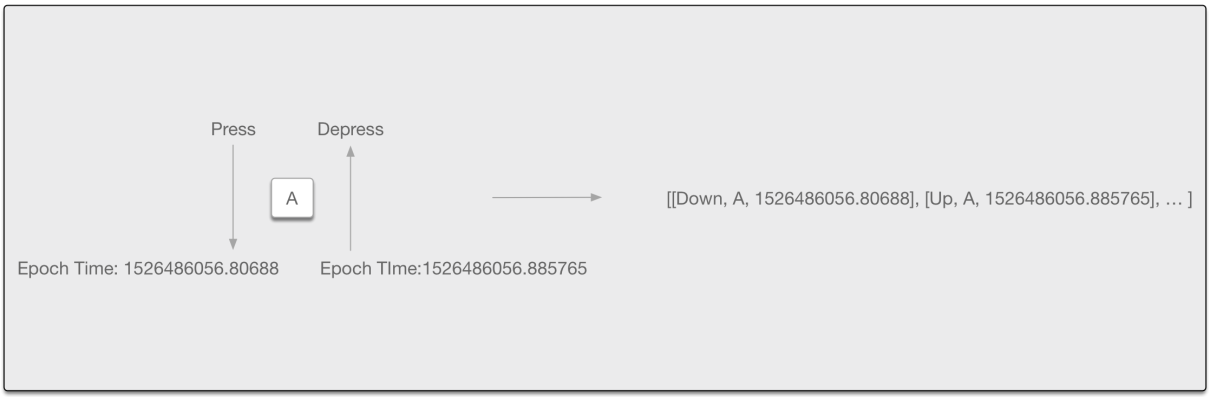

Press
Depress
A
Epoch Time: 1526486056.80688
Epoch TIme:1526486056.885765
[[Down, A, 1526486056.80688], [Up, A, 1526486056.885765], …]

Figure 5.5: Illustration of the keylogger sequence, which intercepts a key press or depress and it's timestamp, and saves them in a list.

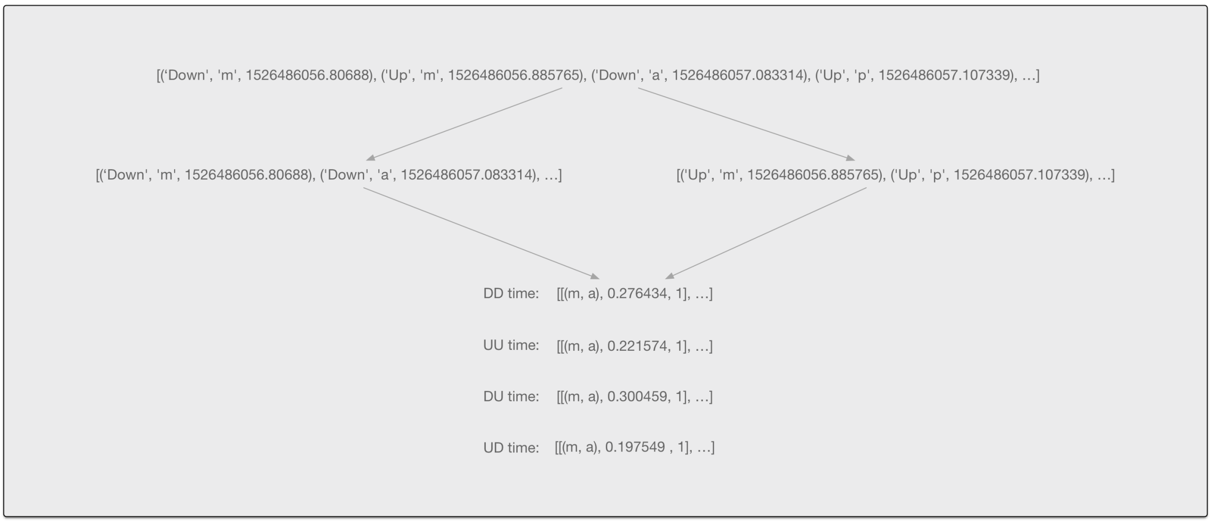

[('Down', 'm', 1526486056.80688), ('Up', 'm', 1526486056.885765), ('Down', 'a', 1526486057.083314), ('Up', 'p', 1526486057.107339), …]
[('Down', 'm', 1526486056.80688), ('Down', 'a', 1526486057.083314), …]
[('Up', 'm', 1526486056.885765), ('Up', 'p', 1526486057.107339), …]
DD time: [[(m, a), 0.276434, 1], …]
UU time: [[(m, a), 0.221574, 1], …]
DU time: [[(m, a), 0.300459, 1], …]
UD time: [[(m, a), 0.197549 , 1], …]

Figure 5.6: Illustration of the pre processing sequence which splits the events into a list of "UP" events and a list of "DOWN" events, which are then used to calculate digraph times.

figure 5.5. The timestamp is recorded in Unix Epoch time as introduced in section 4.3.

Pre Processing

In accordance to requirements FR_04 and FR_07, before extracting features from the recorded keystrokes, the data has to be filtered and pre processed. The keylogger module should continue running in parallel to the pre processing according to requirement FR_13. This module performs the actions introduced in requirements FR_04 and FR_07, as well as could perform FR_05 and FR_06.

The list of measurements is received from the keylogger module. Firstly, the keystrokes are separated in two lists, one for the "UP" action and one for the "DOWN" actions. These two lists will be used to calculate digraph times as illustrated in figure 4.6. Secondly, using the two lists,

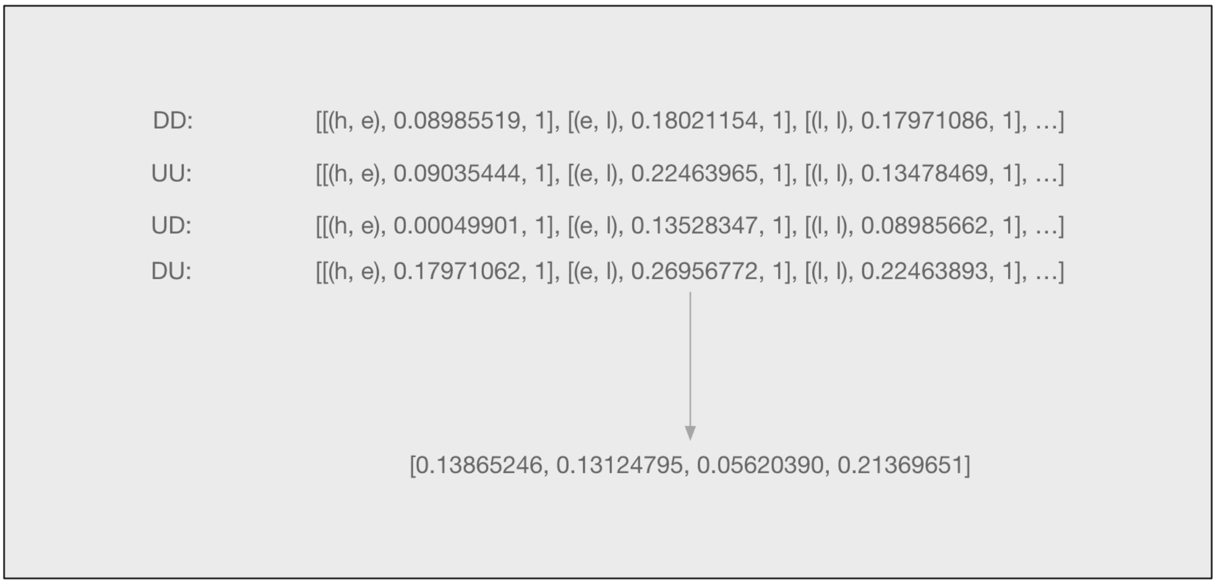

DD: [[(h, e), 0.08985519, 1], [(e, l), 0.18021154, 1], [(l, l), 0.17971086, 1], ...]
UU: [[(h, e), 0.09035444, 1], [(e, l), 0.22463965, 1], [(l, l), 0.13478469, 1], ...]
UD: [[(h, e), 0.00049901, 1], [(e, l), 0.13528347, 1], [(l, l), 0.08985662, 1], ...]
DU: [[(h, e), 0.17971062, 1], [(e, l), 0.26956772, 1], [(l, l), 0.22463893, 1], ...]
[0.13865246, 0.13124795, 0.05620390, 0.21369651]

Figure 5.7: Illustration of the feature extraction sequence, where the average values of the digraph times are calculated and saved into a feature vector.

the keys are arranged in digraphs and the UU, DD, UD and DU times are calculated for each digraph. Additionally, digraphs with times higher than 600 ms are ignored. In the next step, the mean time values for each digraph is calculated. Standard deviation for each digraph should also be calculated according to requirement FR_04. The calculated values, together with the number of occurrences are saved in four new lists, where each entry has the format [<digraph>, <mean value>, <standard deviation>, <occurrences>], where each list corresponds to either UU, DD, UD, DU times. These four lists are then used as input for the third module, the feature extraction module. These steps are illustrated in figure 5.6.

Feature Extraction

This module is responsible of satisfying the FR_08, NFR_02 and NFR_04 requirements. Regarding FR_08, the system uses the output from the Pre Processing module, and further processes the data in order to build feature vectors that can be understood used as input by the machine learning algorithm. As introduced in the requirement, the UU, DD, UD and DU times, from the digraphs calculated at the previous step, are averaged and the mean values are used to build four features. The system could also calculate the standard deviations in order to provide eight features which should improve the performance of the machine learning algorithms. The output of the module is one list that contains the four features. The process is illustrated in figure 5.7, by using the data from figure 4.7.

Furthermore, as the average values are calculated, the context of the data is removed, as it would be impossible to reverse the process and understand what the user initially typed, satisfying

NFR_02 and NFR_04.

5.3.2 Server

As illustrated in figure 5.3, the objective of the server is to perform user authentication, as introduced in requirement FR_02. The server receives the feature vectors from the client and saves it for training the machine learning algorithm and performs classification. The server also provides a module for testing the performance of the machine learning algorithm according to requirement NFR_07.

Furthermore, strong security should be implemented in order to protect the server resources, as well as the communication between the client and the server, according to requirement NFR_01. Also, the server will only store the necessary data for the keystroke dynamics authentication system functionality according to requirement NFR_03.

When the client goes online, an initialization step is performed. In this step, the client should authenticate with the server, and optionally, timers can be started for the user. The purpose of this timers is to raise an alarm if a user that should be active stops transmitting, as per requirement FR_15.

Testing Module

According to the requirement NFR_07, the server should allow testing of different machine learning algorithms, in order to compare results. Since this module has access to the server database, tests can be run on real user data in order to provide useful results. Furthermore, given the modular nature of the system, as introduced in NFR_05, the running machine learning algorithm can be modified without affecting the functionality of other parts of the system.

This module runs K-Folds cross validation, where the saved user features in the database are collected in one dataset. The dataset is then split into two parts, the training set and the test set. Using this method, different machine learning algorithms can be tested for the live dataset and the one performing the best can be chosen to be implemented in the live authentication process.

Machine Learning Algorithm

The purpose of this module is to fulfill requirements FR_02, FR_10, FR_11 and FR_12, FR_15, FR_18. The machine learning algorithm that performs the best in the testing phase should be implemented in the live system.

At the time of writing this project, with the UU, DD, UD and DU average values used as features, the Linear Regression algorithm performed best. For this reason, Linear Regression will be implemented. SVM will also be implemented for confirming the performance scores that

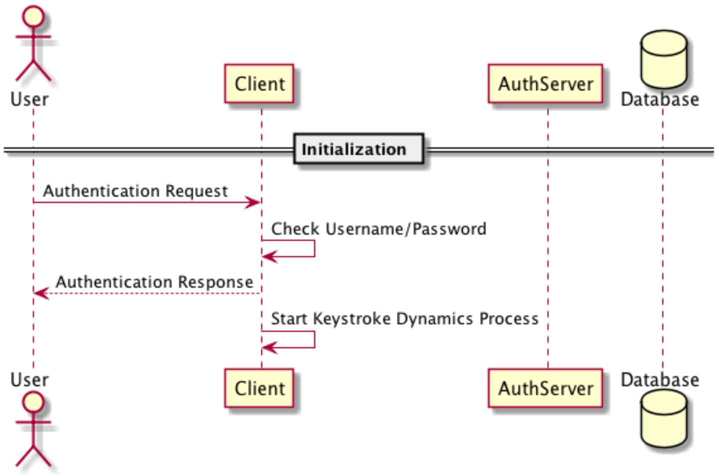


Figure 5.8: The initialization sequence, illustrating the initial system login of the user and the automatic start of the keystroke dynamics process in the background.

were tested. The output of the algorithm is the username and the probability for that user. The username is then compared to the active user's username and if they match the user is considered legitimate. Otherwise, the user may be considered as an intruder.

5.4 Sequence Diagrams

This section introduces the interactions between the system entities in a temporal sequence. The interactions are illustrated using UML sequence diagrams. Five cases are introduced in this section, depending in which mode the system is running. The system can be in Mode Selection mode, Training mode, Initialization Mode, Authentication mode or Logout mode. The complete system sequence diagram is attached in Appendix A, while snipets of the diagram will be used in this chapter.

5.4.1 Initialization

The purpose of the Initialization mode is to start the keystroke dynamics process automatically after the user is logged into their client, according to requirement NFR_03. The process is illustrated in figure 5.8. The first means of authentication is out of scope of this design. In this case it is considered that normal username and password combination is used for logging in the operating system. After the user credentials are checked, the keystroke dynamics process is started in the background upon OS initialization.

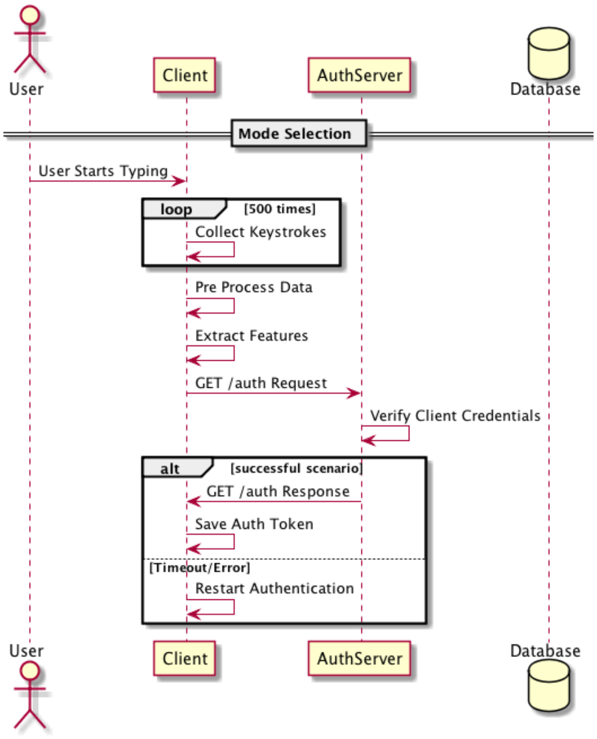


Figure 5.9: First part of the mode selection sequence. In this sequence, the feature vector of the user is created and before being sent to the server, the client gets authenticated with the AuthServer by providing appID and app secret or a refresh token in exchange for an access token.

5.4.2 Mode Selection

The main purpose of the Mode Selection sequence is to collect the data on the client and depending on the user status, ask the server to perform authentication for the user or save their features for later training. The process is illustrated in figures 5.9 and 5.10. This mode runs in a loop for as long as the user is active. Requirements FR_03, FR_04, FR_07, FR_08 are performed at this stage.

Initially, when the user starts typing, the system collects their keystrokes. When 200 keystrokes are collected, corresponding to 400 events, the data is pre processed and filtered and features are extracted. These functions are performed by modules on the client which are described in detail in the previous section 5.3.

The next step is for the client to authenticate with the server, in order to be able to access the protected server API. The client sends their APP ID and APP password to the server by sending a GET request to the /auth API. The server verifies the provided credentials and if the verification is successful, it sends an access token and a refresh token to the client. The access token has a validity of 10 minutes, while the refresh token can be used to obtain a new access token when it is expired. As this process is running in a loop, the client only has to provide APP ID and APP password in the first iteration, while in the next iterations, it can use only

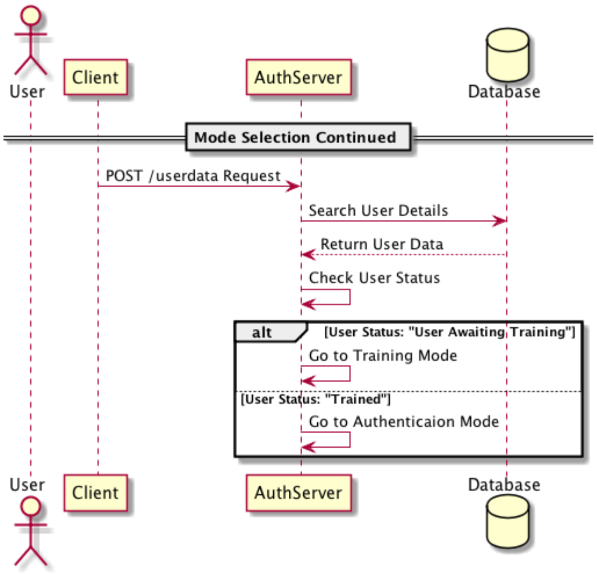


Figure 5.10: The second part of the mode selection sequence, where the feature vector is sent to the server and depending on the user status the server goes into training mode or Authentication mode. **???**

the refresh token to obtain new access tokens. In case the API call is unsuccessful, the client authentication process is restarted.

Subsequently, as illustrated in figure 5.10, the client saves the access token and then uses it in order to build a POST request to the /userdata API in order to send the user's feature vector. The access token is included in the header of the POST message, while the user features, togheter with the username are included in the body section.

Using this information, the server searches for the user details in the database. Depending on the output of this action, the system will continue to function either in training mode or authentication mode. If the status of the user is "User Awaiting Training" or the user does not exist at all, the system goes into training mode. Else, if the user status is "Trained", the system will go into authentication mode.

5.4.3 Training Mode

The sequence diagram for the case where the system is running in training mode is illustrated in figure 5.11. This mode is run for new users or users that the algorithm has not been trained for yet (with the status "Awaiting Training"). This mode collects user features for the next training session of the algorithm as introduced in requirement FR_10.

Firstly, the server saves the received user features and awaits for confirmation that the features are successfully saved. If it action is successful, it sends a response to the client for the POST

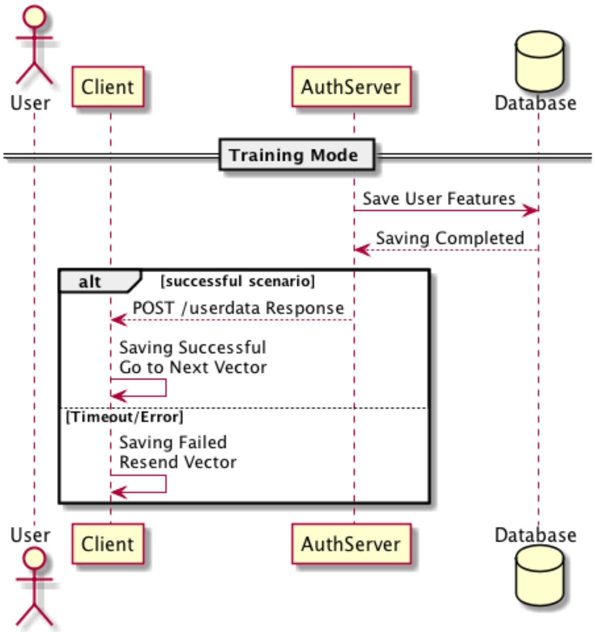


Figure 5.11: Illustration of the training mode sequence where the server saves the user feature vectors until the next algorithm training session is performed.

/userdata API request with a 200 HTTP code and a saving successful message, instructing the client to proceed on sending the next feature vector when ready.

In the case an error occurs on the server or the API call returns an error, the corresponding HTTP code is included in the response, as well as a message in the body instructing the client to resend the feature vector and informing it of the error nature.

5.4.4 Authentication Mode

The sequence diagram for the case where the system is running in authentication mode is illustrated in figure 5.12. This mode is run for returning users for which the status is set to "Trained". This mode saves the user feature vectors in order to be used for the next algorithm training session FR_10, and runs the classification algorithm for the received feature vector. The server than compares if the predicted user is the same with the active user and the probability is above the 80% threshold.

Both in the case where an intruded is detected or a legitimate user is detected, the system will send a successful response to the client's API call, with a 200 HTTP code and a successful message, instructing the user to send the next feature vector when ready. However, in the case when an intruder is detected, an alarm is also sent to a security or maintenance center. The functioning of this entity is out of scope of this project, but this entity can take an action based on the input from the AuthServer and send instructions to the client. In case the process fails

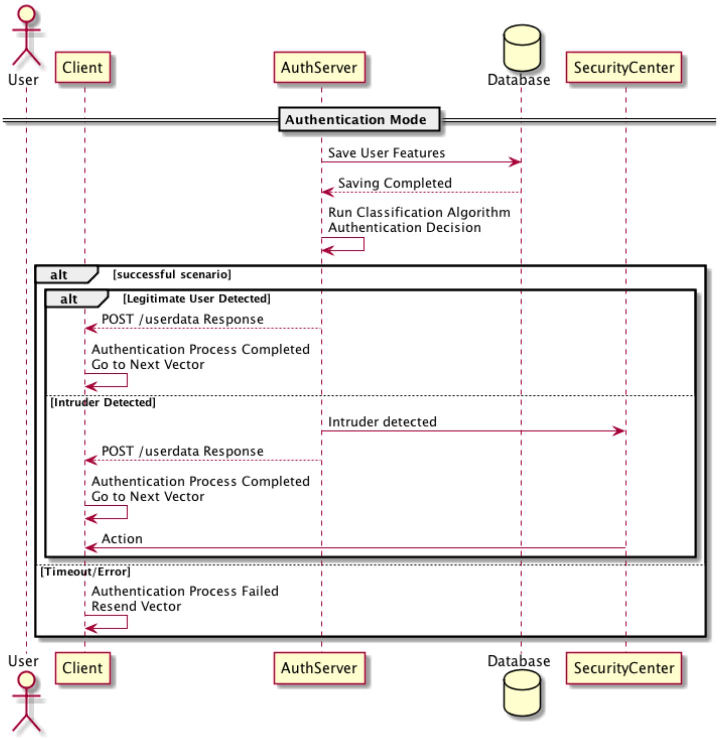


Figure 5.12: Illustration of the authentication mode sequence where the user feature vector is saved and the classification algorithm is saved. If an intruder is detected an alert is sent to an entity that is responsible with ensuring security within the network.

on the server or the API call returns a timeout, the corresponding HTTP message is received and the client is instructed to resend the feature vector.

5.4.5 Logout

The sequence diagram for the case where the system is running in logout mode is illustrated in figure 5.13. In this case, when the user logs out their operating system, according to FR_17, the client informs the server that the user is no longer active. The server should clear all the timers related to the user, if timers are in use, and cancel the refresh token for the client.

When the user requests to logout of the OS, the OS will send a message to the keystroke dynamics process to shut down. In this case, firstly the client informs the server that it is going offline by sending a POST request to the /logout API. The access token is included in the header of the POST message while the user name of the active user is included in the body. Based on this information, the client cancels the tokens corresponding to the client and, in the case timers are attached to the user as per requirement FR_16, it stops these timers as well. If these tasks are performed successfully, the server sends a POST response message, with the 200 HTTP code and tells the client that it can proceed with stopping the process. Otherwise, in case of errors or timeout, the corresponding HTTP error code is sent and the client is informed to resend

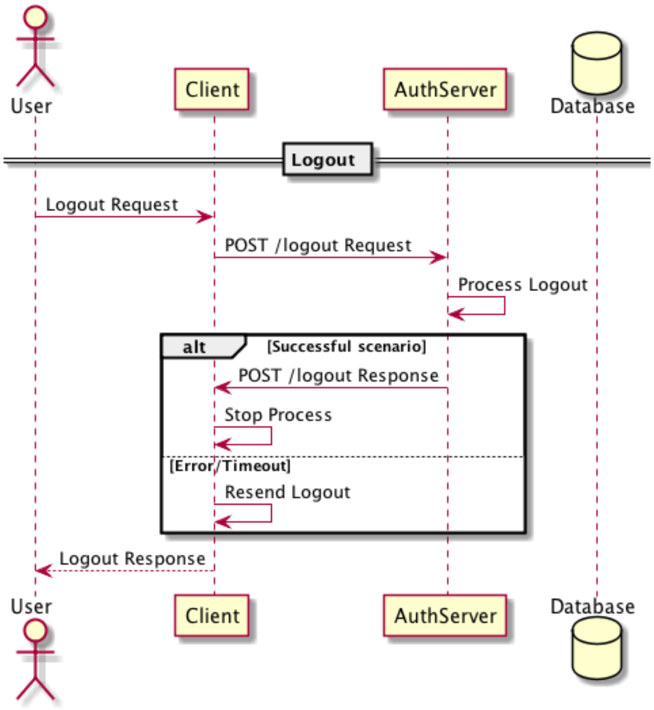


Figure 5.13: Illustration of the logout mode sequence. When the user logs out of their client, the client informs the server about this action.

the POST request, together with information about the error nature. If the call fails for more than three times, the process is automatically closed on the client and the logout sequence is performed.

6| Implementation

The aim of this chapter is to document the steps taken in the implementation of the solution discussed in this paper, according to the design from the previous chapter 5. Explanations, code snippets and illustrations are used in order to demonstrate the development steps. As introduced in the Methodology chapter 2, the chosen development methodology is SCRUM. The full code of the system is presented in Appendix B.

As introduced in 2, SCRUM methodology includes a product backlog, where all the tasks to be completed are stored. These tasks are derived from the analysis and are presented in the requirements table 4.1. The backlog could also include extra tasks like Software Installation, Software Configuration and Testing. Due to time constraints not all sprints are completed, yet given the prioritization of the requirements, the "Must" requirements are included in the first sprints and should be implemented. The development language used is Python 3.7. According to requirement NFR_05 the system is built in a modular approach, using the OOP programming paradigm in order to allow for easy modification and upgrading to different functionality of the system.

6.1 Sprint 1

The first sprint deals with implementing the basic functionality for the keylogger and pre processing modules. The following requirements are approached in this sprint: FR_03, FR_04, FR_07, FR_09.

The keylogger module is monitoring every keystroke of the user and records the keystrokes corresponding only to alphabetical letters. Each recorded keystroke is appended to a list. When the list size reaches 200 entries, the list is returned for further processing.

Using the list from the keylogger as input, the pre processing module arranges the data into digraphs with their corresponding time difference. When a digraph occurs more than once, the mean value and standard deviation are calculated. The pre processing module outputs a list of digraphs and corresponding mean value for each time measurement performed on the digraph, according to figure 4.6.

6.1.1 Keylogger Module

As described in section 5.3, this module is responsible with recording all the keystrokes all the user and save them in a list that can be processed by the upcoming modules. It is also responsible for filtering out the keystrokes that are not alphabetical letters. In order to implement this module, two external libraries are used, namely Keyboard and Pythoncom.

The Keyboard library, available on GitHub [43], is a Python library that allows an application to take control of the keyboard. It offers the possibility to hook global events, which means that the application is able to monitor keystrokes regardless which application is in focus of the OS. It can also capture the keystrokes and record information about them like ASCII code or timestamp and it can also simulate keystrokes. This library is used for recording the user's keystrokes and saving them in a list that can be further processed, as described in section 5.

The Pythoncom library encapsulates the OLE (Object Linking and Embedding) automation APIs. Object Linking and Embedding enables developers to create objects and then link or embed them in a second application. The methond PumpWaitingMessages() is used from this library which pumps all waiting messages for the current thread into the application. This is a non-blocking loop which will allow the program to collect all the keystroke events until a condition is satisfied.

This module contains the KeyLogger class. The class consists of four methods, KeyDownEvent, KeyUpEvent, mainLoop and storeEvent. The KeyDownEvent and KeyUpEvent methods are invoked when the when a key is pressed or released and an event object is passed to them. For this functionality, keyboard.on_press and keyboard.on_release methods, introduced by the keyboard library, are used as they are invoking a callback whenever a key is pressed or released passing the event information.

As it can be seen in the example code listing 6.1, when a keystroke is pressed, the KeyDownEvent is called in order to take actions. In this case, the method will create the activity name "Down" and call the storing method. In the storing method the .name and .time attributes of the event object are extracted.

```
keyboard.on_press(self.KeyDownEvent)

def KeyDownEvent(self, event):
    self.storeEvent("Down", event)

-- Section Omitted --

def storeEvent(self, activity, event):
        keystrokeName = event.name
        keystrokeTime = event.time

-- Section Omitted --
```

Listing 6.1: When a key is pressed a callback function is involved and an event object is passed. The event information is then stored.

As part of the storeEvent method, event information is also appended to a list, in the format

[<action (UP/DOWN)>, <key>, <timestamp>], as presented in section 5.2. In order to filter out keystrokes that are not represented by alphabetical letters, a condition that the event.name is alpha and that its length is equal to 1 is tested before appending the event. The implementation of this action is presented in the code listing 6.2.

```
if event.name.isalpha() and len(event.name) == 1:
           self.eventList.append((activity, keystrokeName, keystrokeTime))
```

Listing 6.2: Filtering of keystrokes that are not represented by alphabetical letters before being saved in a list.

In order to continuously listen to keyboard events, and return a value for the next module to process it, the pythoncom.PumpWaitingMessages method is used. When 200 keystrokes are registered, the list of event information is returned. The code listing performing this action is presented in listing 6.3.

```
pythoncom.PumpWaitingMessages()
    if len(self.eventList) == 400:
       return self.eventList
```

Listing 6.3: Non-blocking loop pumping all the messages from the current thread to the application.

6.1.2 Pre Processing Module

As presented in section 5.3, the purpose of this module is to process the data received from the keylogger and output four lists, each list corresponding of the timing information of each digraph. It also filters out keystrokes with times higher than 600 ms. Three libraries are used for this module, collections, ast and statistics.

The ast module "helps Python applications process trees of the Python abstract syntax grammar" [44]. In this case, this library is used for reading lists of keylogger outputs from file. Even though this functionality may not be required in a live scenario, it is implemented at this step for testing purposes. Since the modules are only implemented on one machine at this stage, samples from multiple users are required. For this reasons, several users ran a keylogger on their computers and the output of the keylogger was saved in a file.

The collections library provides specialized container datatypes for Python [45]. In this case, the functionality that is used is the defaultdict subclass which returns a dictionary like object. Using this functionality, a sequence of key-value pairs is grouped into a dictionary of lists. This is required in order to transform the list of digraphs and their corresponding times into a dictionary that uses the digraph as the key and the time as the value.

The statistics library "provides functions for calculating mathematical statistics of numeric (Real-valued) data" [46]. For this case, the average and standard deviation methods offered by this library are used when calculating the average and standard deviation time values for digraphs that are occuring more than once.

This module contains the PreProcessing class. It provides methods for reading and processing keylogger data received from a file (readFromFile, extractFromFile), process keylogger data received directly from the keylogger (preProcessThis) and methods for calculating the digraph statistics as illustrated in figure 4.6 (timeBetweenUPS, timeBetweenDOWNS, timeBetweenUP-DOWN, digraphTime and calcMean). As already outlined, reading and processing the data from files is implementing for testing purposes.

When receiving data from the keylogger, the preProcess. This method is called, passing the eventlist received from the keylogger. Firstly, the elements of the list are split in two lists, each corresponding to either the "UP" or "DOWN" actions. The code listing for implementing this functionality is illustrated in listing 6.4. Since the format of the eventlist is [<action (UP/DOWN)>, <key>, <timestamp>], list comprehension is used on order to filter based on the first element of each list (etype). The two lists are saved in two variables, ups and downs and then, the methods for calculating digraph measurements are called.

```
def preProcessThis(self, eventList):
    self.downs = [(etype, ename, etime) for etype, ename, etime in eventList
                    if etype == "Down"]
```

Listing 6.4: Extracting the elements corresponding to depresses from the keylogger eventlist.

The functionality of these methods is similar so the digraphTime method will be used for explanation. Digraph time method calculates the time spent between pressing the first key and depressing the second key. The first element of the ups list is pop-ed because it does not have a corresponding down value. Using a while loop, the elements of the ups and downs lists are pop-ed one by one and their time difference is calculated, while the letters are saved together in a list. Furthermore, as illustrated in listing 6.5, the two lists are combined, in order to obtain a unified list where every digraph and the corresponding time are saved.

```

digraph_duration = list(map(lambda x, y: [x, y],
                            digraph_list_du, time_between_down_up))
```

Listing 6.5: Arranging the keystrokes into digraphs and appending the corresponding times.

Even though the digraphs and their corresponding times have been calculated, at this step, if a digraph occurs multiple times, there will be multiple entries in the list corresponding to the digraph. The average value and the standard deviation have to be calculated in order to obtain a single entry for each digraph. Also, before calculating the average, the digraphs with times higher than 600 ms should be filtered out. The implementation for this process is listed in 6.6.

```
def calcMean(self, digraphslist):

        for key, value in digraphslist:
            if abs(value) < 0.6:
                c[key].append(value)

        result = [(t, stat.mean(v), len(v)) for t, v in c.items()]
        std = [(t, stat.stdev(v)) for t, v in c.items() if len(v) > 1]

        return result
```

Listing 6.6: Calculating the average and the standard deviation for digraphs that appear more than once.

The calcMean method obtains the digraph list from any of the methods that are calculationg the digraph times. As a first step, the program iterates through this list. The format of the list at this stage is [<digraph>, time]. The digraph is considered as a key. While iterating through the list, the times for the elements that have the same keys, corresponding to repeating digraphs, are appended together. Using list comprehension, the average values and the standard deviation for elements that occur more than once are calculated.

The result is then returned to the function that requested the mean (either of the timeBetweenUPS, timeBetweenDOWNS, timeBetweenUPDOWN, digraphTime) which then returns the answer to the preProcessThis function. This function returns the four lists that this module must output, being the UU, DD, UD and DU times of each digraph, under the format: [<digraph>, time, occurrences].

6.1.3 Summary

At the end of this sprint, two working modules have been created. The keylogger module is recording the keystrokes of the user and when 200 keystrokes are collected (400 events), the keylogger loop returns the eventlist. The preProcThis method of the pre processing module is called with this list and it outputs four lists with the UU, DD, UD and DU average times for each digraph. In order to test the functionality and communication between the two modules, a main module has been implemented.

As printed in the listing 6.7, firstly two objects are instantiated for each class, one for KeyLogger and one for Preprocessing. In the first case, the 200 keystrokes information is saved in the eventList when it is returned by the keylogger object mainLoop method. The eventList is then cleared so a new recording can start. The eventList is then pre processed by using the preProcThis method of the preproc object and saved into preprocdata variable.

Additionally, in the case where the data is read from a file, the readFromFile method of the preproc object is called. This method reads the list of keystrokes from a file and saves only the alphabetical keystrokes. In order to extract the preprocdata, the extractFromFile method of the preproc object is called. In the end, in both cases the preprocdata should have exactly the same format. The preprocdata is a list of lists which contains the four lists that the PreProcessing module should output.

```
keylogger = KeyLogger()
preproc = PreProcessing()

#Extract times from Keylogger
eventList = keylogger.mainLoop()
keylogger.eventList = []
preprocdata = preproc.preProcessThis(eventList)

#Extract times from file
preproc.readFromFile(filename)
preprocdata = preproc.extractFromFile()
```

Listing 6.7: The KeyLogger and PreProcessing modules working together. In the first case the keystroke data is received from the keylogger while in the second case the keystroke data is read from file.

6.2 Sprint 2

This sprint deals with the development of the feature extractor, the machine learning module and the test machine learning module. The following requirements are implemented in this sprint: FR_08, FR_10, FR_11, NFR_02, NFR_04, NFR_07.

The feature extractor module is using the four lists that are output by the PreProcessing modules for extracting features that will be later used by the machine learning algorithm. The test machine learning module, is using these features to test several machine learning algorithms using the K-Folds cross-validation method. On the other hand, the actual machine learning module is using the same features in order to train the algorithm and take classification decisions.

6.2.1 Feature Extraction Module

As introduced in section 5.3, the purpose of this module is to further process the data, with the objective of building feature vectors that can be used as input by the machine learning algorithms. The initial approach is to calculate the mean values and standard deviations for the UU, DD, DU and UD times provided from the previous steps. However, due to the modular design, the feature calculation method can easily the updated in the future if the classification results need to be improved. The numpy library is used in this module which allows the usage of N-dimensional array objects and offers various functions that can be applied on these arrays [47].

This module contains the FeatureExtractor class. This class contains the extractFeatures method which receives the four lists from the Pre Processing module. In the listing 6.8, it is exemplified how the uu_mean and uu_std features are extracted, using the UU digraph times calculated by the previous module.

The first step zips the UU list in order to separate the digraphs name from the digraphs timing. An example of the uu_zip format is as follows:[((’c’, ’d’), (’e’, ’f’), (’f’, ’a’)), (0.10929989814758301, 0.3404250144958496, 0.1856250762939453), (1, 4, 4)]. This way a list of tuples results, with the middle tuple containing all the UU digraph times calculated before. This middle tuple is then formatted into an numpy array and sorted. Using the .average and .std methods that can be applied on numpy arrays, the average of the digraph times and the standard deviation are calculated.

The same actions are repeated for the DD, UD and DU times. The feature vector is then build by inserting these calculated values to a list. The list is then returned by the function to be used by the machine learning modules.

```
def extractFeatures(self, UU, DD, UD, DU):

    uu_zip = (list(zip(*UU)))
    uu_times = np.array(uu_zip[1])
    uu_times.sort()
    uu_mean = np.average(uu_times)
    uu_std = np.std(uu_times)

    -- Section Omitted --

    features = [uu_mean, uu_std, dd_mean, dd_std,
                ud_mean, ud_std, du_mean, du_std]

    return features
```

Listing 6.8: Implementation of the extractFeatures method with the UU feature calculation example.

6.2.2 Machine Learning Testing Module

The purpose of the machine learning testing module is to test several machine learning algorithms and log their performance by using the same dataset that is used for the authentication.

The algorithm implements K-Folds cross validation on SVM, KNN, Decision Trees, Logistic Regression and Naive Bayes algorithms by using the features provided by the feature extraction module, together with a label that should accompany the features since supervised learning is performed. The sklearn library is used in this module which provides the algorithms as well as the cross validation scoring option. Sklearn is a library that offers tools for data mining and data analysis in Python [48].

```
def SVM_cross_validtaion(features, labels):
    clf = svm.SVC()
    scores = cross_val_score(clf, features, labels, cv=3)
    print(scores.mean(), scores.std() * 2, "SVM")


def KNN_cross_validation(features, labels):
    neigh = KNeighborsClassifier(n_neighbors=3)
    scores = cross_val_score(neigh, features, labels, cv=3)
    print(scores.mean(), scores.std() * 2, "KNN")
```

Listing 6.9: Implementation of the K-Folds cross validation for the SVM and KNN algorithms in the machine learning testing module.

The implementation of cross-validation for SVM and KNN is illustrated in listing 6.9. When performing SVM, an object is instantiated using SVC from sklearn. An object is then created by using the method cross_val_score from the sklearn library, as well. The cv = 3 indicates the number of cross validations to be performed, in this case 3. For the KNN the procedure is similar, except the fact that it should be indicated how many neighbors to include when running classifications in KNN. The testing for the other algorithms is very similar as well and has been omitted. The full code can be found in Annex B. When running a test, the same features must be used for all algorithms in order to ensure correct results. After collecting the features, each method will be called and the scores and standard deviations will be printed for each algorithm.

6.2.3 Machine Learning

The Machine Learning module is the part of the system that receives the feature vectors of the users and performs classification. Based on testing which is described in section 6.4, the SVM and Logistic regression algorithms were chosen for implementation. In order to implement these algorithms, the Sklearn library is used. It has been briefly introduced in the previous module.

This module contains the Classification class. Inside the class, two methods are introduced, train and predict. As the names suggest, one method is used for training the algorithm while the other one is used for predicting classification decisions. These methods are illustrated in the listing 6.10.

```
# SVM
def train(self, features, labels):

       self.model = svm.SVC(probability=True)
       self.model.fit(features, labels)
       print("Algorithm trained")
def predict(self, sample):

       predicted = self.model.predict(sample)
       score2 = self.model.predict_proba(sample)
       print(predicted, score)

#Logistic Regression
def train(self, features, labels):

       self.model = linear_model.LogisticRegression(C=1e5)
       self.model.fit(features, labels)
       print("Algorithm trained")
def predict(self, sample):

       predicted = self.model.predict(sample)
       score2 = self.model.predict_proba(sample)
       print(predicted, score)
```

Listing 6.10: Implementation of the training and prediction methods for the SVM classification algorithm.

As illustrated, the algorithm receives the features and labels from the previous modules and fits a model using them. In order to perform classification, the algorithm receives samples from previous modules and performs classification, as well as returning a probability for it. The implementations for the two algorithms are very similar. The main difference is the way the model is initialized.

6.2.4 Summary

At the end of this sprint, all modules of the system have basic functionality. FR_02 is also fulfilled as authentication can be performed. In this sprint, the feature extraction module was implemented. It receives the pre processed data and arranges it into a feature vector that is then passed to the machine learning algorithm. Two machine learning modules have been implemented, one for testing and one for performing classification.

The main module has also been extended to support these new modules. The code for this module is illustrated in listing 6.11. Firstly objects are being instantiated for the feature extractor and classification classes. In training mode a for loop is performed, calculating the features for one user. At this stage, keystrokes of the users are read from files. Depending on the size of the for loop, the number of features used is set, in this case 10. The features are appended to a list and for each feature, a label is also appended to a labels list. Both lists are converted to Numpy arrays. This process is repeated for every user that will be trained in the model. When all the features and labels for all users are processed, the data is sent to the classifier for training.

In classification mode, a similar process happens but the features used one by one. This way, after each feature vector that is calculated, it is converted into an Numpy array and then the classifier is calld for prediction.

```
featureext = FeatureExtractor()
classifier = Classification()

#Training part

for j in range(10):
    preprocdata = preproc.extractFromFile()
    singlefeatures = featureext.extractFeatures(preprocdata[0],
                                                preprocdata[1],
                                                preprocdata[2],
                                                preprocdata[3])
    features.append(singlefeatures)
    labels.append(user)
features = np.array(features)
labels = np.array(labels)
classifier.train(features, labels)

#Predict part
singlefeatures = featureext.extractFeatures(preprocdata[0],
                                            preprocdata[1],
                                            preprocdata[2],
                                            preprocdata[3])
singlefeatures = np.array([singlefeatures])
classifier.predict(singlefeatures)
```

Listing 6.11: Extension of the main module to accomodate the feature extraction and machine learning modules.

6.3 Sprint 3

By the start of this sprint, the requirements prioritized as "Must" have been implemented. Now the system offers keystroke dynamics authentication basic functionality and tests can be performed in order to decide what is the best way of extracting features and which algorithms perform best.

In this sprint, FR_01, FR_09, FR_14 should be implemented. These requirements refer to the client server architecture implementation. Given the OOP approach taken, classes can be easily moved to the client and to the server. The client server communication has to be implemented. The server should provide RESTful APIs as introduced in the Design chapter 5, while the client should be able to call these APIs. OAuth2.0 will be used for exchanging authentication tokens.

6.4 Functionality and Performance Testing

This section introduces the results of the tests discussed in section 4.6. Due to time constraints, by the time of writing this paper, tests are still being performed and more data is being collected.

The data was collected from different students of Aalborg University, which freely installed a keylogger on their laptop and recorded their keystrokes. One factor limiting the amount of data available was that the keylogger only worked on Windows OS. Also, since the keylogger had to be manually started and could be stopped at any time, in some cases, people stopped the keylogger and forgot to start it while some people did not use it at all due to tracking concerns. The logs from the keylogger were saved in a text file, saved locally on the students' computers. They could then send the file in order to be used for testing. Only seven samples were received from which only five had enough recorded keystrokes in order to be useful.

By using 200 keystrokes (400 events) to obtain features, K-Folds with a k=3 was run for SVM, KNN, Decision Trees, Logistic Regression and Naive Bayes. The mean values of UU, DD, UD and DU times were used for building the feature vectors of the users. 10 features per user have been used, as that was the maximum number of features that could have been extracted from the number of samples available in order to be able to use all the five users. There results are show below in listing 6.12. In listing 6.13, the standard deviation of these means is also included.

```
0.7352941176470589 0.0415945165403851 SVM
0.7075163398692811 0.06808060998692701 KNN
0.7630718954248366 0.10569255966533483 Trees
0.7908496732026143 0.12016193667222376 LogisticRegression
0.7712418300653594 0.2221741585854056 NaiveBayes
```

Listing 6.12: Test Results for K-Folds cross validation using mean values of UU DD UD and DU times as features. Score in percentage (left) and standard deviation (right) are printed.

```
0.7352941176470589 0.0415945165403851 SVM
0.7156862745098039 0.09705387192756532 KNN
0.7826797385620915 0.07263434892448654 Trees
0.7908496732026143 0.12016193667222376 LogisticRegression
0.6960784313725491 0.15251322731474554 NaiveBayes
```

Listing 6.13: Test Results for K-Folds cross validation using mean values and standard deviation of UU DD UD and DU times as features. Score in percentage (left) and standard deviation (right) are printed.

As it can be concluded from the above results, the Logistic Regression algorithm provided the best results given the available datasets. The number on the left is the algorithm score in percentage while the number on the right is the standard deviation of the results from the three folds. The logistic regression algorithm was then implemented and it was set to run in the background while writing parts of this project. SVM was also tested as it was implemented in the beginning of the project for testing. The results snippets can be found in Appendix C. The output probabilities, however, are not usable. A way of calculating the prediction accuracy has to be implemented in order to set thresholds.

While running in the machine learning in the background for the author, during the writing of this project, for SVM 20/122 predictions were wrong, giving a 16.39% FRR. For Logistic Regression 17/78 resulting in a 20% FRR.

It is planned in future tests to rerun the previous tests with higher amounts of data. More feature vectors will be used for training the algorithms. Furthermore, more persons should be trained in the algorithm with different typing proficiency. The users that have provided samples will be asked to test the system in order to calculate the FRR for them. One limiting factor is also the choice of features so it should be tested how the system performance changes when other features are considered. As it can be seen in figure 6.1, when calculating the mean value of all the digraphs, the distributions for two users are very similar. Also, since different digraphs will be typed with different speed, depending on their position on the keyboard, the distribution does not follow a bell curve. Unregistered users will also be asked to test the system in order to test FAR.

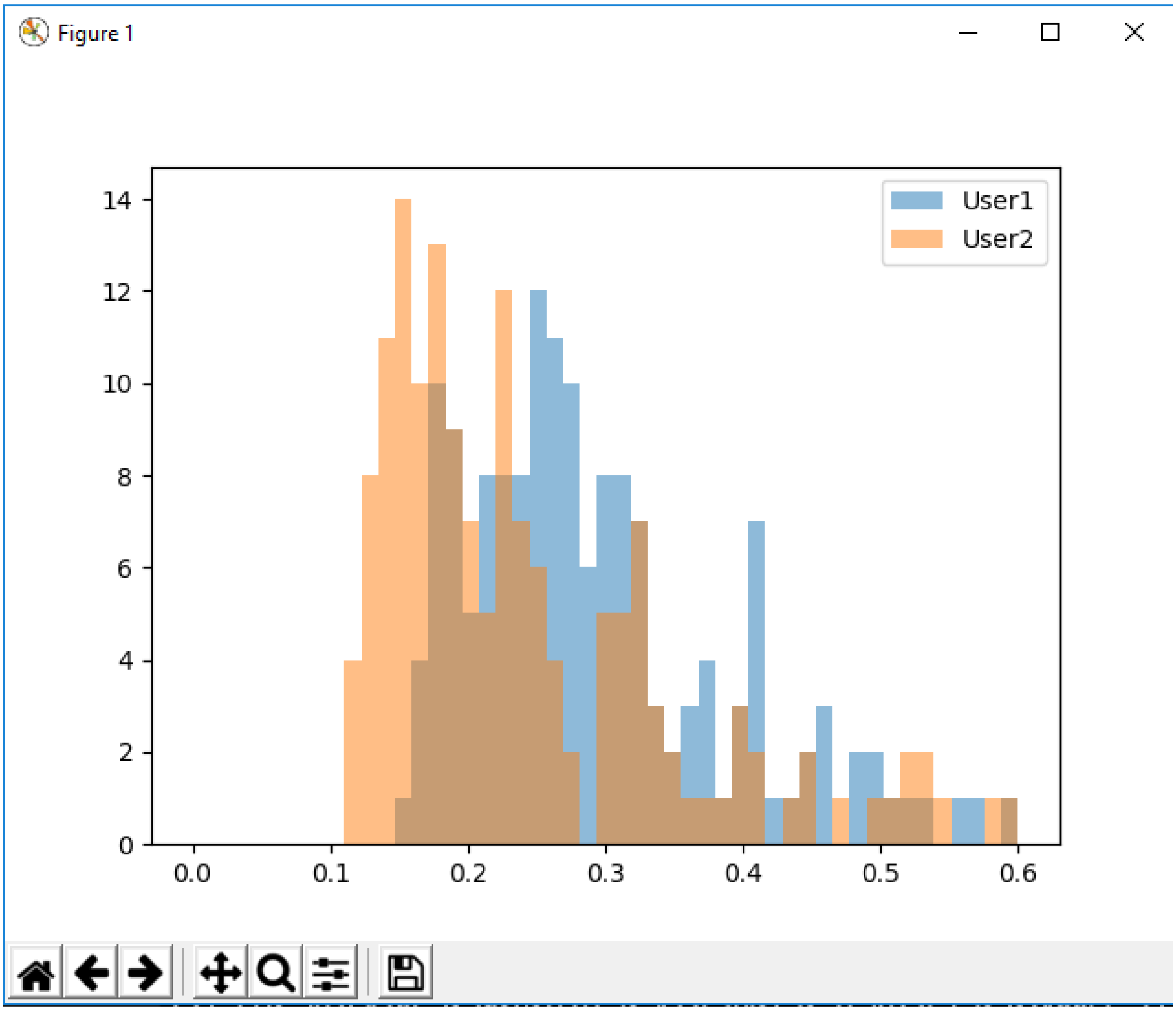


Figure 6.1: Distribution of the UD digraph times for two users.

7| Discussion

The goal of this project was to perform a study on the applicability of a keystroke dynamics system in the real world, by considering what advantages it would bring in terms of security, how such a system would function, as well as proposing an architecture that would make use of today's technologies to implement such a system in a company. However, due to time and resources constraints, certain limitations were set for the project. This section introduces a short discussion on other factors that should be taken into consideration when considering deploying such a system in a real world scenario.

As it was briefly presented in the Introduction chapter 1, General Data Protection Regulation (GDPR) is a regulation introduced by the European Union in order to protect and give users control over their personal data. This regulation came into force on 25th of May and, among others, introduces concepts like lawful processing of data, privacy by design and different rights for users like right to be forgotten, right to information or right to data portability [49]. Since the keystroke dynamics authentication system collects personal data about users, a study should be done on how GDPR would impact the implementation of such a system. Privacy by design principles introduce the idea that a system should be designed with privacy of the users in mind from the beginning of the process. While certain requirements have been presented in this project related to this matter, the main focus has been put on the functionality of the system. Furthermore, in order for the processing of data to be performed in a lawful way, a consent may be needed from the user before data can be collected. This consent should clearly state what data is collected, for which purposes and how it is processed. According to the right to be forgotten, the user should always have the option to delete their data on demand. These concepts should certainly be taken in consideration when considering implementing a keystroke dynamics system.

Furthermore, a market analysis should be performed in order to understand if such a solution would be worth the investment of time and money. While at a first glance, such a product may bring security benefits to users and companies, the willingness of them investing in such a system is a matter of debate. A case could be made for companies that are handling systems which contain very sensitive data where access has to be very strictly controlled. A risk assessment for such a company may reveal that investing in such a system may provide benefits and lower the number of steps required to take in order to be fully certain of a user's identity.

Another fact worth mentioning is people's reaction to such a system. Even if it is properly explained to them what the purpose of such a system is, the fact that everything they are typing is being logged may make some people uncomfortable. Furthermore, even though it should be impossible for anyone to trace back what a user initially typed, it can still be deducted if a user is typing and how much they are typing, which may raise the idea that people's activity can be tracked. As it was presented in this project, some papers suggest tracking people's emotions using keystroke dynamics as well. As discussed in the previous paragraph, it could be considered to use such a system just for controlling access to certain restricted, sensitive areas and only track the users when they are using this system, while not tracking them when performing usual, daily tasks.

Additionally, a keystroke dynamics system might not be enough on it's own to perform continuous authentication. As it only tracks keystrokes, a hacker taking control of a computer would not be detected until they start typing. Even then, a number of keystrokes is always required in order to be able to extract features. For this reason, tests should be performed on how this kind of systems could collaborate with existing technologies in order to provide better results.

8| Conclusion

The initial challenge for this project was finding new, innovative ways of performing authentication that would help with the cyber security issues that companies are facing today. One field that has been developing, together with the developments in machine learning, is the field of behavioral biometrics. After initial research in this field, the keystroke dynamics concept was analyzed. This way, the main problem of the project was set.

How to implement a machine learning system that uses keystroke dynamics to continuously authenticate the users?

The main problem of the project was further divided into five smaller sub questions in order to provide more granularity to the question. In order to understand what are the benefits of such a solution, the first question referred to identifying the cyber security improvements that it would bring. It was soon realized that most of the popular attacks today, that try to steal user's credentials by either exploiting the user's IT system or the user's trust, would easily be mitigated by such a solution. This relates to the fact that even if the user's credentials are compromised, a hacker would also have to imitate the user's typing pattern in order to stay authenticated in a system. Even though, the performance that keystroke dynamics authentication systems reached today may not be as good as other traditional authentication methods, the fact that it can be used to perform persistent authentication, without any additional hardware, makes it a good candidate for a second factor authentication method which can be used as a detective and forensic control.

The second question accentuates the fact that, in order to perform continuous authentication, a way of tracking the user continuously should be found. This way of tracking should provide inputs that translate the physical action of pressing a key on the keyboard into information that a computer can use to take decisions. The concept of keylogger was introduced for this reason, which is a piece of software that can intercept the information exchanged between the operating systems and other applications in order to record inputs from I/O devices. This way, a developer could build an application that constantly intercepts all the keystrokes of the users, even if the application is running in the background. However, being able to freely track user's activity rises privacy concerns. The user should be informed about this action and the data should be carefully secured, while all the stored data or data that leaves the user computer should have temporal context removed so it would be impossible to recreate whatever the user was typing.

The third question introduced is the problem of transforming the keystroke information into features that can be used to separate users. Since the timestamp of each keystroke and the actual key that was pressed are the only information that can be recorded from a user's interaction with their keyboard, measurements based on time are used for building features. It was also discussed at this step that transforming keystrokes information into features removes the contextual information and temporal order of letters. However, a balance between performance, resource usage and simplicity has to be found at this step, as using high dimensionality, for example, may require complex dimensionality reduction algorithms that could greatly impact the performance of the user's computer.

In the fourth question, a discussion was brought upon the concept of machine learning. At this step, based on the requirements of the system, given the features used as input and the required output, a decision was made on using supervised learning algorithms. Features are collected from users that are using the system, labeled and then used to train the machine learning algorithm. In order to decide which algorithm is the most recommended for the current project, according to best practice multiple algorithms satisfying the requirements have been tested and the ones that provided good results are used for further testing

The last question referred at finding ways of calculating the performance of the system in order to understand how it would work in a real life scenario. Some preliminary results have been presented in this paper. However, not enough data has been collected in order to obtain more precise results. As future works, more testing data will be gathered and more types of features will be collected and tested in order to understand the impact on the performance of the algorithm.

In conclusion, a keystroke dynamics authentication system has been designed and implemented as a proof of concept in this project. Further testing and tuning has to be performed in order to understand the real performance capabilities of such a system and test it in a real life scenario as a cyber security control.

Bibliography

[1] Robert Havighurst. User identification and authentication concepts, 2007.

[2] Scott Donaldson, Stanley Siegel, Chris K Williams, and Abdul Aslam. *Enterprise cybersecurity: how to build a successful cyberdefense program against advanced threats*. Apress, 2015.

[3] Paul A Grassi, James L Fenton, and Michael E Garcia. Digital identity guidelines [including updates as of 12-01-2017]. Technical report, 2017.

[4] Dobromir Todorov. *Mechanics of user identification and authentication: Fundamentals of identity management*. CRC Press, 2007.

[5] Ping Identity Corp. SAML 101 White Paper. 2016. URL `https://www.pingidentity.com/content/dam/pic/downloads/resources/white-papers/en/saml-101-white-paper.pdf`.

[6] J. Bradley Ping Identity M. Jones Microsoft B. de Medeiros Google C. Mortimore Salesforce Sakimura, NRI. OpenID Connect Core 1.0 incorporating errata set 1. *URL http://openid.net/specs/openid-connect-core-1_0.html*, 2014.

[7] Brian Campbell, Michael Jones, and Chuck Mortimore. Security assertion markup language (saml) 2.0 profile for oauth 2.0 client authentication and authorization grants. 2015.

[8] Kyle O Bailey, James S Okolica, and Gilbert L Peterson. User identification and authentication using multi-modal behavioral biometrics. *Computers & Security*, 43:77–89, 2014.

[9] L. O'Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91(12):2021–2040, Dec 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2003.819611.

[10] A. A. E. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, July 2007. ISSN 1545-5971. doi: 10.1109/TDSC.2007.70207.

[11] R Stockton Gaines, William Lisowski, S James Press, and Norman Shapiro. Authentication by keystroke timing: Some preliminary results. Technical report, Rand Corp Santa Monica CA, 1980.

[12] A. Kołakowska. *Usefulness of Keystroke Dynamics Features in User Authentication and Emotion Recognition*, pages 42–52. Springer International Publishing, Cham, 2018. ISBN 978-3-319-62120-3. doi: 10.1007/978-3-319-62120-3_4. URL `https://doi.org/10.1007/978-3-319-62120-3_4`.

[13] John-David Marsters. Keystroke dynamics as a biometric. June 2009. URL `https://eprints.soton.ac.uk/66795/`.

[14] Sungzoon Cho, Chigeun Han, Dae Hee Han, and Hyung-Il Kim. Web-based keystroke dynamics identity verification using neural network. *Journal of Organizational Computing and Electronic Commerce*, 10(4):295–307, 2000. doi: 10.1207/S15327744JOCE1004_07. URL `https://doi.org/10.1207/S15327744JOCE1004_07`.

[15] Junhong Kim, Haedong Kim, and Pilsung Kang. Keystroke dynamics-based user authentication using freely typed text based on user-adaptive feature extraction and novelty detection. *Applied Soft Computing*, 62:1077 – 1087, 2018. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2017.09.045. URL `http://www.sciencedirect.com/science/article/pii/S1568494617305847`.

[16] Stefan Reinholdt Jørgensen Catherine Njeri Gitau, Sorin Zaharia. Identity and access management system in a university context, 2017.

[17] 2018. URL `https://www.iso.org/sites/ConsumersStandards/1_standards.html`.

[18] Duncan Haughey. MoSCoW Method. *Project Smart*, 2011. URL `http://www.projectsmart.co.uk/moscow-method.php`.

[19] Dsdm atern handbook, 2008. URL `https://www.agilebusiness.org/content/moscow-prioritisation-0`.

[20] What is uml | unified modeling language. 2015. URL `http://www.uml.org/what-is-uml.htm`.

[21] Homepage | Scrum.org, 2018. URL `https://www.scrum.org/`.

[22] Hyoung-joo Lee and Sungzoon Cho. Retraining a keystroke dynamics-based authenticator with impostor patterns. *Computers & Security*, 26(4):300–310, 2007.

[23] N David BLEISCH. Method of persistent authentication with disablement upon removal of a wearable device, October 24 2017. US Patent 9,800,570.

[24] Martin Kirschmeyer and Mads Syska Hansen. Persistent authentication in smart environments. Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2008.

[25] Mads I. Ingwar and Christian D. Jensen. Remote biometrics for robust persistent authentication. In Joaquin Garcia-Alfaro, Georgios Lioudakis, Nora Cuppens-Boulahia, Simon Foley, and William M. Fitzgerald, editors, *Data Privacy Management and Autonomous Spontaneous Security*, pages 250–267, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-54568-9.

[26] D. Shanmugapriya and G. Padmavathi. A survey of biometric keystroke dynamics: Approaches, security and challenges. *CoRR*, abs/0910.0817, 2009. URL `http://arxiv.org/abs/0910.0817`.

[27] Fabian Monrose, Michael K Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security*, 1(2):69–83, 2002.

[28] John Leggett, Glen Williams, Mark Usnick, and Mike Longnecker. Dynamic identity verification via keystroke characteristics. *International Journal of Man-Machine Studies*, 35 (6):859 – 870, 1991. ISSN 0020-7373. doi: https://doi.org/10.1016/S0020-7373(05)80165-8. URL `http://www.sciencedirect.com/science/article/pii/S0020737305801658`.

[29] Vaclav Matyas and Zdenek Riha. Toward reliable user authentication through biometrics. *IEEE Security & Privacy*, 99(3):45–49, 2003.

[30] Simon Eberz, Kasper B Rasmussen, Vincent Lenders, and Ivan Martinovic. Evaluating behavioral biometrics for continuous authentication: Challenges and metrics. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 386–399. ACM, 2017.

[31] Paul S. Dowland and Steven M. Furnell. A long-term trial of keystroke profiling using digraph, trigraph and keyword latencies. In Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang, editors, *Security and Protection in Information Processing Systems*, pages 275–289, Boston, MA, 2004. Springer US. ISBN 978-1-4020-8143-9.

[32] Rohit Kumar. Artificial intelligence — basics. In *Machine Learning and Cognition in Enterprises*, pages 33–63. Springer, 2017.

[33] Rohit Kumar. Machine learning—basics. In *Machine Learning and Cognition in Enterprises*, pages 51–64. Springer, 2017.

[34] Jason Brownlee. A tour of machine learning algorithms, 2013. URL `https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/`.

[35] Naive bayes classification, 2017. URL `http://scikit-learn.org/stable/modules/naive_bayes.html`.

[36] K nearest neighbors, 2018. URL `http://scikit-learn.org/stable/modules/neighbors.html`.

[37] Logistic regression intro, 2018. URL `http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`.

[38] Cross-validation: evaluating estimator performance, 2018. URL `http://scikit-learn.org/stable/modules/cross_validation.html`.

[39] Francesco Bergadano, Daniele Gunetti, and Claudia Picardi. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security (TISSEC)*, 5 (4):367–397, 2002.

[40] Neil J Henderson, Neil M White, and Pieter H Hartel. ibutton enrolment and verification requirements for the pressure sequence smartcard biometric. In *Smart Card Programming and Security*, pages 124–134. Springer, 2001.

[41] Marcus Karnan, Muthuramalingam Akila, and Nishara Krishnaraj. Biometric personal authentication using keystroke dynamics: A review. *Applied Soft Computing*, 11(2):1565–1573, 2011.

[42] Msdn hooks overview, 2018. URL `https://msdn.microsoft.com/en-us/library/windows/desktop/ms644959(v=vs.85).aspx`.

[43] Keyboard library specifications, 2018. URL `https://github.com/boppreh/keyboard`.

[44] Ast library specifications, 2018. URL `https://docs.python.org/3/library/ast.html`.

[45] Collections library specifications, 2018. URL `https://docs.python.org/3/library/collections.html`.

[46] Statistics library specifications, 2018. URL `https://docs.python.org/3/library/statistics.html`.

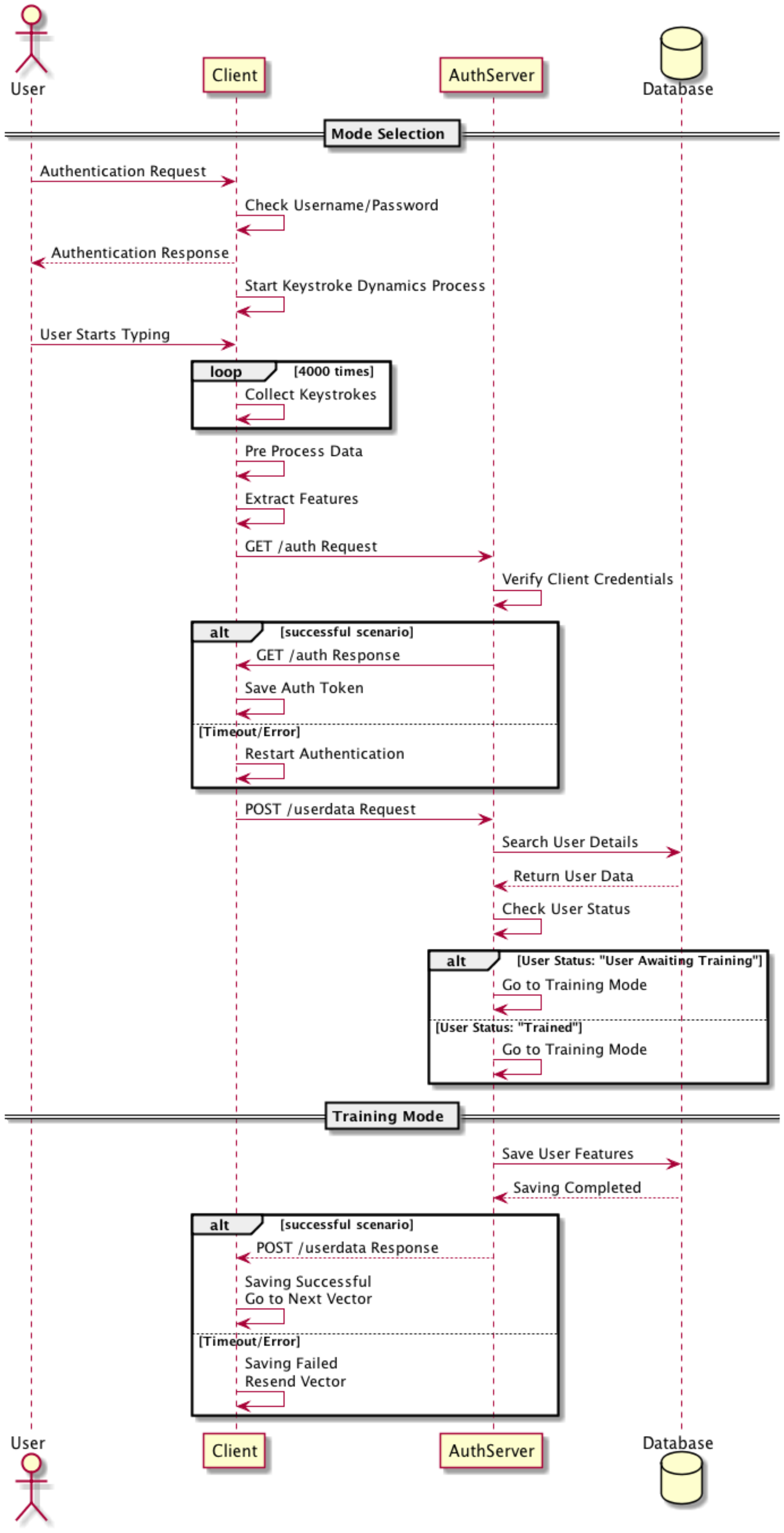
[47] Numpy library specifications, 2018. URL `http://www.numpy.org/`.

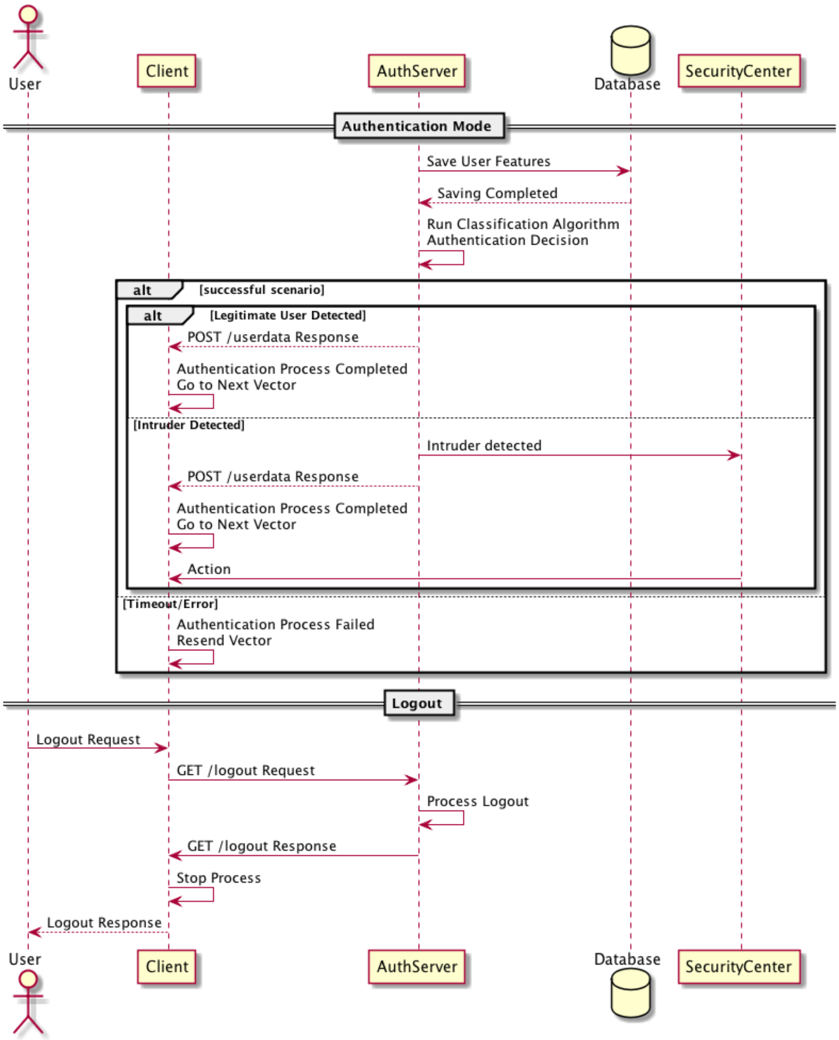
[48] Scikit library specifications, 2017. URL `http://scikit-learn.org/stable/index.html`.

[49] General data protection regulation, 2018. URL `https://gdpr-info.eu/`.

Appendices

A| Full System Diagram


User
Client
AuthServer
Database
Mode Selection
Authentication Request
Check Username/Password
Authentication Response
Start Keystroke Dynamics Process
User Starts Typing
loop
[4000 times]
Collect Keystrokes
Pre Process Data
Extract Features
GET /auth Request
Verify Client Credentials
alt
[successful scenario]
GET /auth Response
Save Auth Token
[Timeout/Error]
Restart Authentication
POST /userdata Request
Search User Details
Return User Data
Check User Status
alt
[User Status: "User Awaiting Training"]
Go to Training Mode
[User Status: "Trained"]
Go to Training Mode
Training Mode
Save User Features
Saving Completed
alt
[successful scenario]
POST /userdata Response
Saving Successful
Go to Next Vector
[Timeout/Error]
Saving Failed
Resend Vector
User
Client
AuthServer
Database


User
Client
AuthServer
Database
SecurityCenter
Authentication Mode
Save User Features
Saving Completed
Run Classification Algorithm
Authentication Decision
alt
[successful scenario]
alt
[Legitimate User Detected]
POST /userdata Response
Authentication Process Completed
Go to Next Vector
[Intruder Detected]
Intruder detected
POST /userdata Response
Authentication Process Completed
Go to Next Vector
Action
[Timeout/Error]
Authentication Process Failed
Resend Vector
Logout
Logout Request
GET /logout Request
Process Logout
GET /logout Response
Stop Process
Logout Response
User
Client
AuthServer
Database
SecurityCenter

B| Source Code

The source code can be found on GitHub at https://github.com/SRNZ91/Keystroke-Dynamics-Authentication. The repository is only public during examination period. Contact the author or collaborators for access.

C| Logistic Regression tests

```
[1] [[0.55500487 0.42550654 0.005994   0.00662926 0.00686532]]
[1] [[6.91498444e-01 3.04021027e-01 4.22812979e-04 3.97682023e-03
  8.08957494e-05]]
[1] [[6.87536517e-01 8.02819554e-03 3.86473102e-04 3.04002165e-01
  4.66497185e-05]]
[1] [[0.88415238 0.0020623  0.01899291 0.09334886 0.00144354]]
[1] [[9.90892239e-01 2.05734559e-04 1.32935021e-03 6.64842654e-03
  9.24249856e-04]]
[1] [[7.70584978e-01 1.97578622e-01 1.95413834e-04 2.31347128e-02
  8.50627377e-03]]
[1] [[6.79003940e-01 3.13727672e-01 8.15943443e-05 6.80718607e-03
  3.79607065e-04]]
[1] [[5.45001489e-01 2.43935064e-06 3.16862599e-01 1.37794412e-01
  3.39061067e-04]]
[1] [[5.38371381e-01 4.61058486e-01 1.00082728e-06 4.62794325e-04
  1.06337476e-04]]
[1] [[9.92216916e-01 4.83419511e-03 1.87925970e-04 2.26127842e-03
  4.99684974e-04]]
[1] [[0.92158404 0.00290685 0.00443293 0.06831271 0.00276346]]
[1] [[8.08438366e-01 1.02249115e-02 1.70770485e-05 1.81312866e-01
  6.78019185e-06]]
[1] [[5.57109501e-01 1.70155593e-04 2.89800346e-01 1.25934736e-01
  2.69852620e-02]]
[1] [[5.55291601e-01 4.25036442e-01 1.43671680e-08 1.96603798e-02
  1.15628268e-05]]
[1] [[9.38491120e-01 7.82879812e-03 1.10043913e-04 5.22461339e-02
  1.32390432e-03]]
[1] [[7.22523745e-01 2.58582763e-01 5.35234823e-05 1.87417926e-02
  9.81755368e-05]]
[1] [[4.99573726e-01 9.01462903e-55 4.99573726e-01 1.28402518e-07
  8.52420279e-04]]
[1] [[7.17093855e-01 8.68923552e-05 2.64743829e-01 1.69786313e-02
  1.09679203e-03]]
[2] [[4.66289636e-01 5.31176241e-01 3.15761515e-08 2.50196324e-03
  3.21283767e-05]]
[2] [[4.71723432e-01 4.82885089e-01 5.19774878e-06 4.53798457e-02
  6.43603928e-06]]
[1] [[8.71428260e-01 1.19470751e-01 7.92368624e-07 8.36713676e-03
  7.33060258e-04]]
[1] [[9.18762055e-01 2.02864377e-04 1.24387038e-02 6.65471630e-02
 2.04921355e-03]]
[1] [[9.42286322e-01 4.84880984e-02 3.36928107e-04 8.83751632e-03
  5.11353285e-05]]
[1] [[8.00335258e-01 9.90180985e-02 7.74865901e-04 9.98658941e-02
  5.88399257e-06]]
[1] [[8.96681336e-01 7.10832109e-03 3.24665297e-02 6.33149140e-02
  4.28899172e-04]]
[2] [[3.84065409e-01 5.99318698e-01 1.31993445e-03 1.51790611e-02
  1.16897052e-04]]
[1] [[8.00670604e-01 1.77913054e-01 1.50934801e-05 1.91429139e-02
  2.25833413e-03]]
[1] [[5.03625939e-01 4.88470486e-01 2.68144979e-07 7.13417364e-03
  7.69133766e-04]]
[1] [[4.99999369e-01 4.90716380e-51 4.99999369e-01 2.11488259e-07
  1.05135494e-06]]
[1] [[9.54070842e-01 4.27027787e-03 1.74339987e-04 1.68605220e-02
  2.46240183e-02]]
[2] [[4.92737061e-01 4.94616867e-01 4.54193340e-09 6.28363614e-04
```

```
1.20177046e-02]]
[1] [[4.99836791e-01 1.47574672e-51 4.99836791e-01 3.70539706e-08
  3.26380563e-04]]
[1] [[0.91044364 0.0362326  0.00339417 0.04113842 0.00879117]]
[1] [[4.99980080e-01 1.11987885e-49 4.99980080e-01 9.54033001e-09
  3.98296239e-05]]
[1] [[0.51425273 0.01625355 0.3405597  0.06737529 0.06155874]]
[1] [[4.97247150e-01 2.52785361e-52 4.97247150e-01 2.19505572e-09
  5.50569784e-03]]
[1] [[0.88401447 0.01092903 0.00162552 0.10096859 0.00246239]]
[1] [[4.99929126e-01 1.32899845e-55 4.99929126e-01 1.66606282e-09
  1.41747247e-04]]
[1] [[6.44181461e-01 2.98475415e-02 1.09633463e-01 2.16288602e-01
  4.89318587e-05]]
[1] [[4.99618627e-01 6.67438568e-47 4.99618627e-01 1.42274783e-08
  7.62732004e-04]]
 [1] [[4.99676373e-01 2.31265045e-49 4.99676373e-01 2.50292788e-08
  6.47229042e-04]]
[1] [[4.99998560e-01 6.63386716e-53 4.99998560e-01 2.94523199e-07
  2.58640152e-06]]
[1] [[7.99002531e-01 4.69855611e-03 4.78944800e-02 1.48362205e-01
  4.22284451e-05]]
[1] [[4.99687934e-01 9.58098262e-43 4.99687934e-01 8.76719292e-08
  6.24044570e-04]]
[2] [[4.55701351e-01 4.92990679e-01 3.14612806e-06 5.12661413e-02
  3.86821488e-05]]
[1] [[6.15718861e-01 3.77235044e-01 1.37547032e-06 4.96454829e-03
  2.08017164e-03]]
[1] [[4.99282051e-01 1.00528149e-47 4.99282051e-01 2.23174893e-09
  1.43589613e-03]]
[1] [[4.99941829e-01 5.70491789e-54 4.99941829e-01 1.34869907e-08
  1.16328733e-04]]
[4] [[0.01150804 0.00082022 0.27395424 0.62459653 0.08912097]]
[1] [[4.99990308e-01 4.16757731e-46 4.99990308e-01 1.36519973e-07
  1.92473115e-05]]
[2] [[2.45664636e-01 7.37746128e-01 3.47194454e-06 1.61255021e-02
  4.60262234e-04]]
[1] [[5.96388607e-01 3.72498371e-01 1.58510715e-04 3.02822961e-02
  6.72214771e-04]]
[1] [[4.99526233e-01 1.39503408e-49 4.99526233e-01 5.99865694e-07
  9.46933192e-04]]
[2] [[4.45683656e-01 5.46645408e-01 7.31814057e-06 7.47147337e-03
  1.92144190e-04]]
[2] [[3.42360548e-01 5.98378628e-01 1.11100310e-03 5.78700003e-02
  2.79820367e-04]]
[3] [[7.57517676e-02 1.40205192e-07 7.25657895e-01 1.92105255e-01
  6.48494245e-03]]
[1] [[5.06207967e-01 4.93004955e-01 2.71163742e-08 7.53523533e-04
  3.35279160e-05]]
[1] [[7.23075406e-01 2.62933992e-01 1.46390830e-05 1.26451385e-02
  1.33082512e-03]]
[1] [[4.99155757e-01 9.13739264e-51 4.99155757e-01 5.30450823e-09
  1.68848108e-03]]
[1] [[6.47263770e-01 1.72486884e-01 7.19048954e-06 1.80133072e-01
  1.09082864e-04]]
[2] [[4.84848004e-01 4.97248902e-01 1.22931770e-04 1.77773943e-02
  2.76737049e-06]]
[1] [[4.99867709e-01 6.44747054e-48 4.99867709e-01 7.63539557e-08
```

```
2.76737049e-06]]
[1] [[4.99867709e-01 6.44747054e-48 4.99867709e-01 7.63539557e-08
  2.64505624e-04]]
[1] [[0.60109359 0.2933519  0.03476367 0.06397713 0.00681371]]
[1] [[8.56387127e-01 9.90413086e-02 1.59167456e-04 4.37488778e-02
  6.63519484e-04]]
[1] [[9.49354217e-01 2.34625931e-02 2.49292906e-04 4.04569638e-03
  2.28882007e-02]]
[1] [[9.13053581e-01 7.51833785e-03 2.37004675e-04 1.76786568e-03
  7.74232112e-02]]
[1] [[8.24141035e-01 1.65589181e-01 4.12424109e-04 9.54653209e-03
  3.10827587e-04]]
[1] [[5.41305656e-01 4.24157358e-01 1.17930902e-04 3.42386137e-02
  1.80441873e-04]]
[2] [[2.80947033e-01 5.52811525e-01 1.11565975e-04 3.52426766e-02
  1.30887200e-01]]
[2] [[2.83959509e-01 6.93134201e-01 4.23053023e-07 2.27721571e-02
  1.33710540e-04]]
[2] [[4.31427134e-01 4.49483052e-01 4.40255180e-08 7.39473895e-04
  1.18350296e-01]]
[2] [[4.72405106e-01 5.21235805e-01 1.39769180e-07 6.15914075e-03
  1.99809061e-04]]
[1] [[4.99995836e-01 2.76823654e-45 4.99995836e-01 2.39993985e-07
  8.08791639e-06]]
[2] [[3.44804596e-01 5.89175569e-01 4.56328116e-06 6.59764250e-02
  3.88469381e-05]]
[2] [[1.11637269e-01 5.79136662e-01 1.51699607e-04 3.09073702e-01
  6.67265314e-07]]
[1] [[4.99995808e-01 5.21035307e-54 4.99995808e-01 7.25026790e-09
  8.37631334e-06]]
[4] [[1.55814802e-01 1.50673925e-01 8.54930006e-05 6.92560778e-01
  8.65001818e-04]]
[2] [[4.07135286e-01 5.77199952e-01 5.42500609e-08 1.56217536e-02
  4.29546552e-05]]
```

D| SVM Tests

```
[1] [[0.26282921 0.22705271 0.19956397 0.2428116  0.06774252]]
[1] [[0.22076823 0.18616209 0.17630506 0.19292159 0.22384304]]
[3] [[0.26216716 0.22574777 0.19825365 0.24040145 0.07342997]]
[1] [[0.26133223 0.22574965 0.1983419  0.24064229 0.07393392]]
[1] [[0.22429303 0.19200881 0.17981167 0.19819254 0.20569396]]
[1] [[0.22165815 0.18796506 0.17763494 0.19314791 0.21959395]]
[3] [[0.26618395 0.22325674 0.19725574 0.2346512  0.07865237]]
[1] [[0.2435823  0.2197947  0.19285286 0.23154922 0.11222092]]
[1] [[0.24491739 0.22513557 0.19718775 0.23894136 0.09381793]]
[1] [[0.24391077 0.21840537 0.19174994 0.23015714 0.11577678]]
[1] [[0.24237913 0.21702225 0.19127291 0.22826543 0.12106028]]
[1] [[0.24317698 0.21824279 0.19214648 0.22978145 0.1166523 ]]
[1] [[0.24220826 0.2179465  0.19214643 0.22912963 0.11856917]]
[1] [[0.24308226 0.21899699 0.19266133 0.23054441 0.11471501]]
[1] [[0.24306199 0.21887874 0.19240542 0.23038531 0.11526855]]
[1] [[0.24231503 0.21800662 0.19231845 0.22924518 0.11811473]]
[1] [[0.24116077 0.2144703  0.18985507 0.22546172 0.12905214]]
[1] [[0.24140595 0.21635998 0.19119563 0.22725245 0.12378599]]
[1] [[0.24352752 0.22060596 0.19335968 0.2324297  0.11007714]]
[1] [[0.24162933 0.21507917 0.1902807  0.22617887 0.12683194]]
[1] [[0.25807026 0.22681364 0.19851946 0.24279152 0.07380512]]
[3] [[0.25970996 0.22488383 0.19653807 0.23898285 0.07988529]]
[1] [[0.22253459 0.18823562 0.17746488 0.19617328 0.21559162]]
[1] [[0.2224372  0.18894529 0.17795845 0.19535391 0.21530515]]
[1] [[0.26501404 0.22779746 0.20204672 0.24493003 0.06021175]]
[1] [[0.22163338 0.18775784 0.17722014 0.19417048 0.21921816]]
[1] [[0.26452683 0.22780156 0.20103252 0.24438945 0.06224964]]
[3] [[0.26019359 0.22443607 0.19683008 0.23837383 0.08016643]]
[1] [[0.26287627 0.22675546 0.20037597 0.24285635 0.06713595]]
[1] [[0.26278191 0.22650551 0.1996436  0.24208093 0.06898805]]
[1] [[0.2257225  0.19476696 0.18144864 0.20059565 0.19746625]]
[1] [[0.26255438 0.22745595 0.20014352 0.24380394 0.06604221]]
[1] [[0.22214427 0.18849194 0.17764014 0.19505583 0.21666782]]
[1] [[0.26269846 0.22735025 0.19993557 0.24350593 0.06650978]]
[1] [[0.26559902 0.22746233 0.20150219 0.24389526 0.06154121]]
[1] [[0.26263689 0.22676854 0.19992429 0.2426597  0.06801058]]
[3] [[0.26029944 0.22418893 0.19727061 0.23819371 0.08004731]]
[3] [[0.26184298 0.22518899 0.19802875 0.23957347 0.07536581]]
[2] [[0.2693855  0.22317096 0.20001237 0.23467082 0.07276035]]
[1] [[0.24291372 0.21705651 0.19127053 0.22849087 0.12026837]]
[1] [[0.24364731 0.22029795 0.19361179 0.23218808 0.11025487]]
[1] [[0.245179   0.22405153 0.19561966 0.23723247 0.09791734]]
[1] [[0.24379124 0.22099305 0.19394947 0.23302488 0.10824136]]
[1] [[0.24240192 0.21802493 0.19217588 0.22928645 0.11811081]]
[1] [[0.24346842 0.21882937 0.19274325 0.23052089 0.11443807]]
[1] [[0.24363826 0.22143914 0.1945182  0.23354796 0.10685644]]
[1] [[0.24216479 0.21453422 0.18968164 0.2259078  0.12771155]]
[1] [[0.24382371 0.21852975 0.19195866 0.23026436 0.11542352]]
[1] [[0.24538876 0.22560703 0.19709549 0.23961845 0.09229027]]
[1] [[0.24351005 0.21800675 0.19192636 0.22965205 0.11690479]]
[1] [[0.24381313 0.2215727  0.19428659 0.2337071  0.10662048]]
[1] [[0.24395266 0.22406299 0.19653275 0.23711439 0.09833721]]
[1] [[0.24426786 0.22010738 0.19274304 0.2320835  0.11079821]]
[1] [[0.24320129 0.21975544 0.19356501 0.23146258 0.11201568]]
[1] [[0.24231191 0.21726864 0.19165559 0.22849041 0.12027346]]
[1] [[0.24400559 0.21985283 0.19324162 0.2317984  0.11110157]]
[1] [[0.24213481 0.2149142  0.18985809 0.22623423 0.12685866]]
[1] [[0.24301907 0.22004535 0.19320219 0.23164317 0.11209021]]
```

```
[1] [[0.24398114 0.22067252 0.19351627 0.23267431 0.10915575]]
[1] [[0.24426587 0.22067601 0.19339553 0.23275747 0.10890512]]
[1] [[0.24202912 0.21425563 0.18925386 0.22564305 0.12881833]]
[1] [[0.24298275 0.21847793 0.1923368  0.22996084 0.11624168]]
[1] [[0.25036407 0.22876198 0.19949873 0.24595873 0.07541649]]
[1] [[0.26217876 0.22673394 0.19933121 0.24236563 0.06939046]]
[1] [[0.22017195 0.18523302 0.17564746 0.19259624 0.22635133]]
[1] [[0.21866057 0.18312039 0.17447762 0.18947804 0.23426338]]
[1] [[0.26217055 0.22647559 0.19905166 0.24185322 0.07044898]]
[1] [[0.262857   0.22728702 0.19968922 0.24321645 0.06695032]]
[1] [[0.21577039 0.17908991 0.17183508 0.18641493 0.2468897 ]]
[1] [[0.26193055 0.22634873 0.19987126 0.24217794 0.06967152]]
[1] [[0.22232237 0.18828681 0.17740706 0.19599756 0.2159862 ]]
[3] [[0.26056076 0.2241518  0.19668749 0.23782233 0.08077762]]
[1] [[0.26208808 0.22583154 0.19851943 0.24067349 0.07288746]]
[3] [[0.25760325 0.22168147 0.19434962 0.23431424 0.09205142]]
[3] [[0.25867089 0.22364192 0.19634304 0.23743004 0.08391412]]
[1] [[0.2629289  0.22705765 0.19973095 0.2428849  0.0673976 ]]
[1] [[0.22075107 0.18621221 0.17652092 0.19205714 0.22445866]]
[1] [[0.26299551 0.22626694 0.19927398 0.24147425 0.06998933]]
[1] [[0.2643521  0.22790777 0.20150995 0.24489889 0.06133129]]
[3] [[0.18388724 0.20577018 0.23061344 0.2932405  0.08648863]]
[1] [[0.15014549 0.16238807 0.18733207 0.23291864 0.26721573]]
[1] [[0.18377912 0.20727045 0.23609696 0.30730385 0.06554963]]
[1] [[0.18346758 0.20674309 0.23420396 0.30319205 0.07239332]]
[1] [[0.18374179 0.20571465 0.2322496  0.29570073 0.08259323]]
[3] [[0.18438846 0.20587564 0.23195357 0.29373337 0.08404897]]
[1] [[0.1843861  0.20644667 0.23355263 0.29804301 0.07757159]]
[3] [[0.18480926 0.20669788 0.23330273 0.29715974 0.07803039]]
[3] [[0.18435434 0.20597158 0.23177366 0.293926   0.08397442]]
[3] [[0.18419368 0.20627974 0.23182494 0.29562804 0.0820736 ]]
[1] [[0.15487834 0.17253083 0.19711856 0.25199279 0.22347949]]
[1] [[0.15473815 0.17654837 0.20051567 0.2586116  0.20958621]]
[1] [[0.15263915 0.16709415 0.19180237 0.24118014 0.24728419]]
[3] [[0.18541495 0.20679571 0.23351109 0.29576517 0.07851308]]
[1] [[0.15257441 0.16869579 0.19298082 0.24331544 0.24243354]]
[1] [[0.15377858 0.17165718 0.195537   0.24853321 0.23049403]]
[1] [[0.18404616 0.20697964 0.23561252 0.30441941 0.06894227]]
[3] [[0.18412497 0.20593112 0.23099952 0.29350944 0.08543495]]
[1] [[0.18443189 0.20674039 0.23355769 0.29895839 0.07631165]]
[1] [[0.1832701  0.20672321 0.23343945 0.30257208 0.07399515]]
[1] [[0.15063698 0.16330792 0.18820157 0.23452372 0.26332981]]
[1] [[0.18514502 0.20692545 0.23404145 0.29794741 0.07594067]]
[1] [[0.18463745 0.20675832 0.23488283 0.30037369 0.0733477 ]]
[1] [[0.18562766 0.20730071 0.23571604 0.30030153 0.07105405]]
[1] [[0.18473749 0.2070313  0.23507028 0.30135845 0.07180247]]
[3] [[0.18496937 0.20733564 0.23369634 0.29950391 0.07449474]]
[3] [[0.18476897 0.20622926 0.23253534 0.29452025 0.08194617]]
[3] [[0.1835808  0.20556846 0.23063832 0.29355743 0.08665498]]
[3] [[0.18343316 0.20548382 0.23068456 0.29380418 0.08659428]]
[1] [[0.18397242 0.20731663 0.2357979  0.30624962 0.06666343]]
[3] [[0.18427784 0.20575546 0.23098833 0.2923868  0.08659156]]
[1] [[0.18456388 0.2072429  0.23610792 0.30447218 0.06761312]]
[1] [[0.18591658 0.2075413  0.23667708 0.30180122 0.06806383]]
```