FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Behavioral authentication system
Student:	Bc. Jan Pešek
Supervisor:	prof. Ing. Pavel Tvrdík, CSc.
Study program:	Informatics
Branch / specialization:	Computer Systems and Networks
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2023/2024

Instructions

The thesis aims to design and implement a system for behavioral authentication of users of mobile e-banking applications. The system will provide online scoring of user authenticity based on behavioral analysis of user data continuously collected from the application and smartphone sensors by the server.

Proceed in the following steps:
1) Conduct research on what data to collect from a smartphone application to enable behavioral authentication of mobile e-banking application users.
2) Analyze smartphone device sensors available via Android SDK that could be used for the behavioral authentication scores of users.
3) Develop an Android application simulating a mobile e-banking application that collects and sends user data to a server for statistical evaluation.
4) Develop a server application for collecting and storing user data.
5) Enhance the server with a module that evaluates data and estimates the authentication scores of users.
6) Test the quality of authentication score estimation.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 12 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Behavioral authentication system

Bc. Jan Pešek

Department of Computer Systems
Supervisor: prof. Ing. Pavel Tvrdík, CSc.

May 4, 2023

Acknowledgements

I would like to thank prof. Ing. Pavel Tvrdík, CSc. for the guidance during the writing of this diploma thesis. I am grateful to Petr Paščenko, Head of Data Science at Profinit EU, s.r.o. for his expert advice in consulting the data part of this work. Finally, I would like to thank all volunteers supporting my project during dataset collection.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 4, 2023 …………………

Citation of this thesis

Pešek, Jan. *Behavioral authentication system.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Cílem této práce je vytvořit Proof of Concept autentizačního systému pro ověření identity uživatele na základě jeho chování v mobilní aplikaci. Systém využívá data ze senzorů a dotykové obrazovky chytrého telefonu k ověření chování. Výsledkem je plně funkční behaviorální autentizační systém, jehož funkčnost je demonstrována v simulované aplikaci mobilního bankovnictví.

Klíčová slova behaviorální autentizace, senzory smartphonu, dotykový displej smartphonu, simulace aplikace mobilního bankovnictví, webové služby

Abstract

This thesis aims to develop a Proof of Concept authentication system that verifies a user's identity by analyzing his/her behavior in a smartphone application. The system exploits data from a smartphone's sensors and touchscreen for behavioral authentication. The result is a fully functional behavioral authentication system with its functionality demonstrated in a simulated mobile banking application.

Keywords behavioral authentication, smartphone's sensors and touchscreen, mobile banking simulation, web services

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

AUC Area under the Receiver Operating Characteristic Curve.

BAS Behavioral Authentication System.

BBD Behavioral Biometrics Database.

DTO Data Transfer Object.

DTW Dynamic Time Warping.

EER Equal Error Rate.

FSD Feature Store Database.

HTTP Hypertext Transfer Protocol.

JPA Java Persistence API.

JSON JavaScript Object Notation.

LOF Local Outlier Factor.

LoOP Local Outlier Probabilities.

REST API Representational State Transfer API.

RF Random Forest.

SVM Support Vector Machine.

UUID Universally Unique Identifier.

Chapter 1

Introduction

Modern smartphones have surpassed their predecessors in terms of competency. The size of the devices has grown, small displays and keypads have been replaced with touch-screens, connectivity has improved dramatically, and various motion sensors have been added. New ways of using these devices emerged, and mobile application developers compete for developing a better user experience. Even conservative banks no longer offer their products just in their branches and have made their services available remotely. The new intuitive smartphone applications attract many users who already take their smartphones as a part of everyday life. They use it to manage sensitive content, such as accessing their bank account, which brings additional security concerns.

After installing a new application that holds sensitive data, there is typically a registration process during which a user declares his identity. Before each subsequent access to the application, the user must prove that he is who he claims to be, or in other words, go through an authentication process. A combination of several authentication factors could be included in that process, such as something the user owns, something the user knows, and something the user is. In the case of a mobile banking application, the user owns a registered device, knows the PIN or password, or can use some personal identification biometric technology such as a fingerprint reader or facial recognition.

It is difficult to get hold of someone else's fingerprint, but a login procedure usually assumes an imperfection of biometric authentication and can be bypassed just by knowing the password, which is prone to vulnerabilities. From naive attacks like guessing a weak password or shoulder surfing, to more complex ones. Several studies report side-channel attacks. A study [1] reconstructs heat traces left by a finger after login action, another study [2] explores oily residues to guess a login pattern, and [3] measures the efficiency of guessing a PIN by analyzing motion sensors output. All these attacks assume physical access to a device, but there are also remote attacks, such as Accessibility Service Attack to interact with a smartphone remotely through accessibility service [4].

Listed examples of various attacks have in common that an attacker could pass a single point of authentication and use the accessed session undisturbed. In the context of the number of legitimate accesses to an application, any attack is a rare event, so requesting a re-authentication regularly during a session would ruin the user experience.

Behavioral authentication falls into the category of biometric authentication, thus something the user is. It assumes that each user has a characteristic behavior when using a smartphone, which is hard to mimic. For example, how a user holds the phone, clicks, scrolls, or the micro-movements captured by motion sensors. Biometrics in the context of behavioral authentication is thus represented by data produced by a smartphone's sensor or touchscreen. Similarly to other biometric authentication, such as fingerprint or facial recognition, a training phase is required to learn patterns that define a context for further comparison. But in contrast, behavioral authentication does not need specialized hardware components or user attention. The authentication process operates continuously in the background, even after the primary single-point authentication process is finished.

While an entered password is either correct or incorrect, behavioral authentication is not decisive to such a degree. Ultimately, an imperfection shows up in all biometric authentication methods. Commodity hardware installed in smartphones is imperfect and can produce data at an unstable sampling rate or be influenced by ambient noise. Even if hardware components were perfect, the human factor would still affect the accuracy. Human behavior changes over time, and the skill with which a device is operated is evolving. Moreover, the user's position, the activity he is currently doing, or even his mood can modify behavioral patterns. Due to these difficulties, lower accuracy of our proposed system can be expected than the accuracy of other proven authentication methods in commercial use. Behavioral authentication does not aspire to become the primary factor of authentication, instead, it creates an additional layer of security.

This diploma thesis describes the design and implementation of a Proof of Concept system for behavioral authentication. The proposed system is implemented as a web service that continuously receives data produced by motion sensors and a touchscreen of a smartphone during a user's activity within an application session. The system can store the received data for future system improvements and can extract relevant behavioral characteristics from the received data, which are then matched to past behavioral characteristics of the same user using statistical evaluation. The output of the statistical evaluation is an authentication score, interpretable as the probability of the session being performed by an attacker. The estimated authentication score is made available continuously to the smartphone application owner. The system does not take any further action based on the estimated authentication score.

This approach allows the smartphone application owner to choose an arbitrary threshold for classifying legal users and attackers. A session will be accepted if the probability is below this threshold; otherwise, it will be rejected. The threshold setting varies depending on the use case and is determined by the subjective cost of an error. Setting the threshold to a high value indicates that the application owner is prepared to face the consequences of accepting an attacker into a session. Presumably, it is more important that legal users are rarely bothered by false rejection or that the cost of individual verification is high, e.g. making a phone call to the client to ensure the legal user made the bank transaction. On the other hand, a low threshold will be set in a sensitive application that cannot tolerate an attacker, even at the cost of more frequent false rejections.

This diploma thesis is not supported by any bank, so I do not have access to a source code of any genuine mobile banking application or dataset. Therefore, the behavioral authentication system is developed and launched only in a simulated environment. The mobile banking application is simulated using a custom Android application, through which the data are collected, and the functionality of the final authentication system is demonstrated.

1.1 Diploma Thesis Structure

Chapter 2 is a survey of related work. Behavioral authentication of smartphone users is a relatively young topic and not much publicly studied. Several papers focus on users' behavior during a specific activity. I will examine what those activities are, what data are collected, what features are extracted to exploit the data for behavioral authentication, and what algorithms are used.

Chapter 3 describes components of the proposed behavioral authentication system and defines the system's functionality.

In Chapter 4, the data that can be read from sensors and touchscreens of Android devices are described, along with how to access those data programmatically.

Because there was not any available dataset relevant to the use case of this work, nor a definitive approach to designing a behavioral authentication system, it was necessary to collect a behavioral biometric dataset and find a way to exploit the collected data for behavioral authentication within this diploma thesis.

This diploma thesis is, therefore, further divided into three stages. In the first stage, I developed a client-side Android application and a server-side web service. The Android application, further described in Chapter 5, is designed to simulate a user interface of a typical mobile banking application and is implemented so that it collects data from selected sensors and touchscreen in the background and transports them to the web service. The web service receives data from the Android application, evaluates the data using an authentication scoring module, and stores the data with an authentication score in a database. However, the first stage aimed only to collect data, so the evaluation step was bypassed at the first stage. The web service implementation is described in Chapter 6. The developed Android application is currently publicly available on Google Play [5] and was shared externally to collect as much data as possible.

The collected data are analyzed in the second stage of this work. Chapter 7 describes data analysis of the data collected in the first stage. The data were used to investigate suitable feature extraction and modeling techniques for behavioral authentication.

And finally, in the last stage, the behavioral authentication system is implemented by enhancing the web application with the authentication scoring module. The scoring module was implemented based on selected techniques investigated in the second stage. The implementation is described in Chapter 8.

For demonstration purposes, the Android application was modified to not only send the behavioral biometrics but also to request the estimated authentication score from the behavioral authentication system and display the result in the user interface.

Chapter 2

Related Work

Behavioral authentication is not a smartphone device's privilege. Several studies attempt to exploit user behavior for authentication when using a computer. It has been reported that how a user types on a keyboard and moves a mouse while filling a form in a browser provides discriminatory features between users [6]. Some studies try to perform behavioral authentication through WiFi signals interfered by human motion around the room. The primary motivation is to protect the entire content of a room instead of a specific device without the help of extra hardware that the incoming person would have to carry [7]. However, this diploma thesis is focused on smartphone device applications, especially for activities done in a typical mobile banking app.

The internet portals *IEEE Xplore*, *ResearchGate* and *Google Scholar* were used as a starting point for the research, all of these allow to search by keywords, and so the following were used: *behavioral authentication, continuous authentication, behavioral biometry.* The most cited research papers, surveys, and their references related to smartphones were further considered.

While achieving stunning accuracy in discriminating users based on their behavior, many studies used a limited dataset size. Many users behave differently; finding features that discriminate between two participants is a much easier task than building an accurate authentication system for a hundred of users. Not only the number of users but also the number of repeated interactions by a single user is important to determine the impact of behavioral change over time. Therefore, model performance is not always a guarantee of a useful study.

2.1 Studied Biometrics

Related work can be divided according to activities performed by participants during data collection. Keystroke-based authentication focuses on typing on a virtual keyboard; touchscreen-based authentication explores gestures; motion-based authentication systems rely on data produced during walking, phone rotating or picking it from a table; and finally, multimodal authentication takes into account data from motion sensors and touchscreen combined. This diploma thesis focuses on an authentication system for mobile banking applications that no one expects to be active in the background. Instead, it typically serves a relatively quick action and then is terminated. Therefore studies related to gait are not considered. Scrolling or typing activities are, on the other hand, relevant to this work.

Behavioral biometrics consists of data produced by a touchscreen or sensors. An Android touchscreen produces a **touch event** when a user puts a finger on a touchscreen, moves with the finger while in contact with the touchscreen, or lifts the finger from the touchscreen. Each touch event includes properties such as x- and y-coordinates, timestamp, the pressure on the screen, and the area covered by the finger. The details about touch events are described in Section 4.3. A sequence of touch events that begins with touching the screen and ends with lifting the finger is referred to as a **stroke**. Data from sensors that are included in behavioral biometrics are sampled continuously at a specified rate forming a time series of **sensor events**. A sensor event contains a timestamp and three-dimensional data vector that contains a sensor's sample.

Research of behavioral authentication for phones was initiated by *Frank et al.* [8] in 2012. They investigate if it is possible to authenticate users while they perform basic navigation steps such as scrolling or swiping using data produced by a touchscreen only. To exploit the strokes for behavioral authentication, the authors propose several features replicated by other studies and also considered in this work. The study claims users use distinctive screen areas for their strokes, therefore using coordinates of the two end-points as a feature. The median velocity of the five last points is able to distinguish users that stop the finger before lifting it from those that lift their finger while it still has a finite velocity. Mean resultant length quantifies how directed the stroke is on a scale between 1 for a straight line and 0 for uniformly random angles of line segments. The trajectory length, the direct distance between its endpoints, and the ratio between these two define a measure of angular dispersion. Because all strokes deviate a bit from the straight line, they use the largest absolute perpendicular distance between the end-to-end connection and the trajectory to distinguish if the largest deviation is on the left or right sides of the end-to-end connection. This might be an indicator of whether the user is left-handed or right-handed. Some users scroll steadily and slowly, while others quickly scroll to a new position. This can be detected by stroke duration and time between two consecutive scrolls. The authors indicate the area covered by a finger during a stroke is one of the most informative features.

A more recent paper [9] analyses strokes with accelerometer and gyroscope sensors' time series while drawing a triangle on a touchscreen. Touch features are extracted in the same way as in the case of [8]. From the sensors' time series, they extract sensor events that are produced during a stroke. The extracted subseries are then transformed into scalar features by applying simple statistical functions such as minimum, maximum, mean, or standard deviation on each of the three dimensions. Features extracted from a touchscreen, accelerometer, and gyroscope sensor data are merged into a single feature vector representing a single stroke.

Other studies do not focus on long-lasting touchscreen interactions in the form of gestures. Instead, they try exploiting behavioral data produced by tapping on a key. In [10], the authors investigate behavioral biometrics when typing a four- or eight-digit PIN. They collect touch events and sensor events from a gyroscope and accelerometer. They use a similar extraction procedure as [9], i.e., aggregating sensor data subseries using simple statistics during a stroke, but instead of using all three dimensions, they use only the magnitude computed as $m = \sqrt{x^2 + y^2 + z^2}$, where x, y, z denotes the three sensor events' dimensions. An explanation of these data is provided later in Section 7.2.4. Authors form a feature vector using sensors' magnitudes, pressure, and area covered by a finger at the beginning and end of a stroke. They also use a stroke duration and time between two consecutive strokes.

An innovative approach is chosen by *Sitová et al.* [11]. They define new types of features to capture subtle micro-movements of a user while tapping on a virtual keyboard a free text. They try to quantify a resistance of a hand grasp to a force exerted by a stroke and how quickly perturbations caused by pressure from a stroke disappear after the stroke is complete. To do that, they extract samples from the accelerometer, gyroscope, and magnetometer sensor during a stroke with a 100 ms neighborhood window and compare the statistical characteristic of the three-dimensional data time series before starting a stroke, during the stroke, and after the stroke ends. However, it was reported that these features do not perform better than the ones used in [10], although their combination is beneficial. I will not use these types of features in this diploma thesis, and so they will not be further described. However, the approach that includes the time neighborhood of a stroke to the feature extraction is innovative and potentially beneficial.

Behavioral authentication is especially advantageous for mobile banking applications, and as a result, there are studies that specifically examine this area. *Incel et al.* [12] were able to collect data using a genuine banking application. However, the set of used features is not different from the ones mentioned above. Another paper [13] studies behavioral authentication for a mobile banking application, and in addition, they collect gravity, linear acceleration, and rotation vector sensor data. However, their approach to extracting features is not innovative either.

The research papers mentioned above have in common that the data collection was always performed in a controlled environment. Participants of [11] were told to sit at a table. In [10], the participants had to hold the device with the left hand and write with the index finger of the right hand. Data were collected using devices of the same model, using a custom application often consisting of a single activity. This is not the case of [14], where privacy is not an issue, and the data collector runs long-term in the background regardless of the application. However, this can only be achieved with a rooted device, therefore it is irrelevant to this work.

2.2 Modeling Techniques

Related works significantly differ in algorithms used to evaluate extracted data. [8, 10, 9, 12] consider each stroke as an individual sample passed to a model. The model, therefore, outputs multiple predictions for each session, which are then aggregated by averaging or majority voting into a single prediction. *Sitová et al.* [11] try a different approach by averaging test feature vectors sampled during a time window before passing it to a model. Both methods are based on the same concept and are meant to reduce the variance in the final decision. The difference is whether the aggregation happens at the input or output of a statistical evaluation. However, the choice of the time window size directly impacts the system's security if a real-time continuous authentication system is considered. For example, [11] suggests using a time window of 140 seconds, which implies an attacker remains undetected for more than two minutes.

[8, 9, 12, 13] adopts binary classification algorithms. Solid results are achieved by Support Vector Machine (SVM). During training, this model finds a separation hyperplane between classes, i.e., legal user and attacker's samples, which serves as a prediction decision boundary. Thus, samples from both classes in equal portions are needed for training. In addition, [8] sees an advantage in generalizing the observed data because individual samples are forgotten once the separation hyperplane is found. Other studies conclude that the most acceptable algorithm is the Random Forest (RF).

Authors of [11, 10] propose template methods. These methods find a centroid by averaging available feature vectors during a training phase. The found centroid is referred to as a template, which is later compared with arriving test vectors. Different metrics are used for the comparison, [10] uses Z-Score while [11] reports the best results with Manhattan distance. The computed distance is then converted to a binary response using a predefined threshold. Their reason for not using binary classification is an assumption of deploying their authentication system directly on a user's phone and transmitting sensitive data of other users for the learning phase of binary classificator is inconvenient and introduces a security risk.

2.3 Summary

All studies agree that collecting touch events together with sensor events is beneficial, especially sensor events from an accelerometer and gyroscope. When analyzing strokes representing single taps, the size of a touch area appears to be an important characteristic for behavioral authentication. The extraction of scroll features is more complex, as the sequence of touch events potentially contains more details about a user's behavior. Sensor data processing is always initiated with time synchronization with strokes. From the continual sensor time series, those sections sampled during or close to a touch action are extracted. These extracted subseries are then aggregated into scalar values quantifying the force applied for the touch or the instability of the grasp using simple statistical functions. Hence they compare the absolute values generated by sensors, assuming that sensors are stable and reliable across different manufacturers.

Although several studies were mentioned, none of them published the source code used for the analysis. Additionally, with the exception of [11], which only evaluates free text typing, no study published its dataset. I discovered a potentially valuable dataset called *BrainRun* [15], named after a smartphone game where players solve logic puzzles by clicking and sliding on the screen. The dataset includes behavioral metrics from 2218 players who used different devices in an unsupervised environment. Unfortunately, the dataset quality was found to be inadequate during data exploration. The sensor data does not have timestamps, and an unknown transformation has been applied to the data. Consequently, it is necessary to collect our own dataset in order to develop the behavioral authentication system.

Author and year	Size	Activity	Model	EER
Frank et. al. (2012) [8]	41	Scrolling	SVM	4.00 %
Park et. al. (2016) [9]	94	Drawing triangle	RF	0.70 %
Sitová et. al. (2015) [11]	100	Text typing	Template method	8.53 %
Zheng et. al. (2014) [10]	80	PIN input	Template method	3.65 %
Incel et. al. (2021) [12]	45	Mobile Bank	SVM	3.50 %
Estrela et. al (2021) [13]	51	Mobile Bank	RF	6.85 %

Table 2.1: Summary of relevant related work

Equal Error Rate (EER) indicates an error rate of a binary classification algorithm with the decision boundary threshold set to a value so the False Acceptance Rate and False Rejection Rate are equal. Although the mean estimated EER across mentioned studies is below 5 %, their estimations probably contain systematic bias. In real-life situations, users do not behave consistently as they do under supervision. They may hold their phone in different ways, be affected by external factors, or use devices of varying quality, which can impact the quality of the data passed to the authentication system.

Chapter 3

Main Design Decisions

This diploma thesis develops a behavioral authentication system. Or in other words, a system that verifies a user's identity based on behavior while interacting with a smartphone application. User behavior is represented by data from sensors and touchscreens that are built into a smartphone and react to user activities. These data are referred to as **behavioral biometrics**. The behavioral authentication system receives behavioral biometrics, matches it to a set of past behavioral biometrics of the same user, and evaluates the user's authenticity.

While some of the related work from the previous chapter assume the deployment of their proposed authentication system directly on a user's device, our authentication system is centralized and is implemented as a web application. The reason is that our intention is to develop an informative authentication system, in the sense that it does not decide whether to accept or reject a user but only informs the relevant authority about the authenticity of a user. The user's authenticity is quantified by our system using an **authentication score**, which can be interpreted as a probability that an attacker controls the device.

The Behavioral Authentication System (BAS) proposed in this diploma thesis provides a user's authentication score upon request by the relevant authority. It is assumed that the **relevant authority** is the owner of the smartphone application that BAS protects and therefore has the power to decide whether to accept or reject a user based on the authentication score.

The BAS consists of two components and a relational database, as visualized in Figure 3.1. The first component is called **BehavioralProcessor**, which receives behavioral biometrics from a smartphone application and forwards it to the second component called **ScoringModule**, which computes the authentication score. The BehavioralProcessor then stores the behavioral biometrics with the computed authentication score in the relational database, which is referred to as Behavioral Biometrics Database (BBD).

The BAS offers an additional layer of security to protect against fraud, especially in sensitive applications like mobile banking. Unfortunately, genuine behavioral biometrics data from a mobile banking application is unavailable for this diploma thesis, so the client side is only simulated.

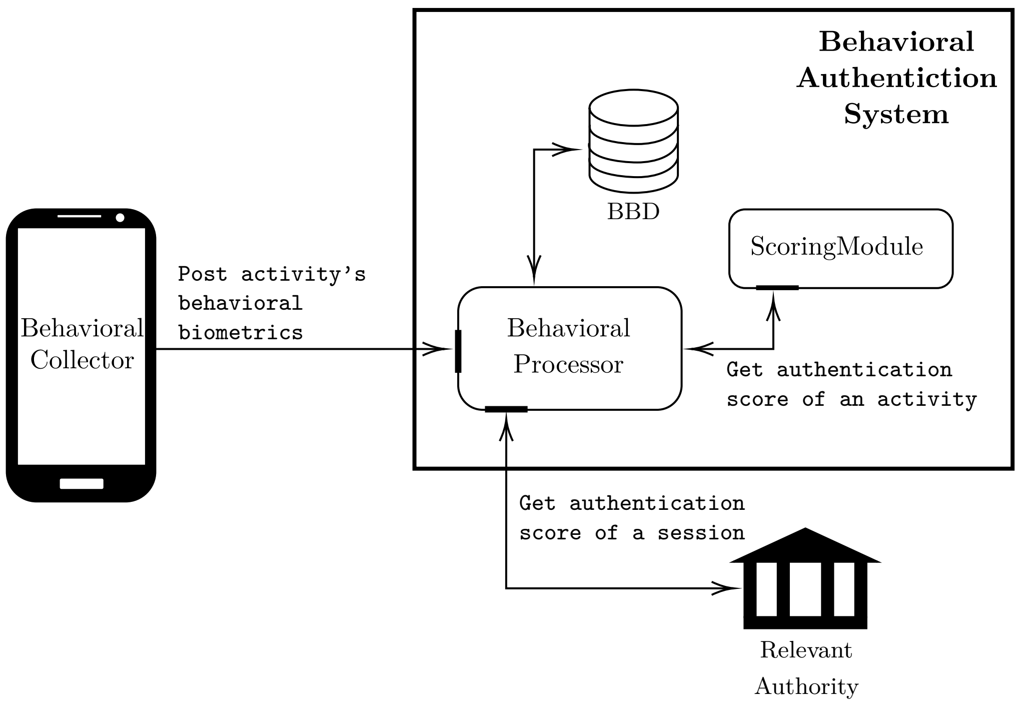

Behavioral Authentiction System
BBD
ScoringModule
Behavioral Collector
Post activity's behavioral biometrics
Behavioral Processor
Get authentication score of an activity
Get authentication score of a session
Relevant Authority

Figure 3.1: A Simplified Architecture Overview

The client-side simulation is done through an Android application named **BehavioralCollector**. Its user interface is designed to simulate actions done in a mobile banking application, such as entering a PIN, scrolling a list, typing an account number, or clicking on buttons. It consists of four independent **activities** that a user completes in a given order while behavioral biometrics are collected in the background. Once an activity is finished, the data are sent to the BehavioralProcessor.

A **session** refers to a group of activities that are sent from the same run of the BehavioralCollector application. A session begins when the user starts a new application run and ends when the user exits the application run.

The BAS evaluates behavioral biometrics from the BehavioralCollector per completed activity. However, the relevant authority is interested in the overall authentication score for an entire session interpretable as the probability that the session is performed by an attacker. To provide the authentication score of a session, the BehavioralProcessor combines the authentication scores of all activities from the session.

While the BehavioralCollector and BehavioralProcessor are straightforward to implement, the ScoringModule performs a statistical data evaluation. Thus, before the implementation of the ScoringModule, it is necessary to design a data pipeline that extracts features from behavioral biometrics and uses applicable modeling techniques to match the features of the newly arrived activity's behavioral biometrics with the features of old activities' behavioral biometrics.

Extracted features should have the potential to discriminate between users. In other words, features should remain stable over a long period of time for a single user while showing significant variation from other users. In order to identify these features, it is necessary to collect a dataset of behavioral biometrics.

We use the BehavioralCollector and BehavioralProcessor applications to create the behavioral biometric dataset. I published BehavioralCollector on Google Play, distributed it to volunteers, and asked them to use the application on their smartphones repeatedly without supervision. BehavioralProcessor collects and stores the incoming data in a database while the ScoringModule is inactive.

I have analyzed the collected data, studied suitable feature extraction techniques, and investigated possible techniques for behavioral authentication. Selected techniques were implemented within the ScoringModule. This diploma thesis proceeded in the following steps:

1. A way to read data from a touchscreen and sensors of an Android device was explored.

2. The BehavioralCollector was developed and made publicly available on Google Play.

3. The BehavioralProcessor was developed to enable the collection of data from BehavioralCollector applications controlled by volunteers.

4. The collected dataset was analyzed to find applicable feature extraction and modeling techniques.

5. The ScoringModule was implemented based on the conclusions of the previous step.

6. The BehavioralCollector was modified to display a session's authentication score in the user interface for demonstration purposes.

Chapter 4

Android technology

This chapter presents what sensors an Android device can have according to the Android documentation [16] and shows how to read the data it produces using Android Sensor API. A dedicated section will cover retrieving touch event data from a touchscreen as it has a different reporting mechanism than sensors. However, this chapter does not explore the data in depth. Data description is covered later in Section 7.2.

4.1 Sensors Overview

Android devices can have built-in sensors capable of monitoring three-dimensional device motion, orientation, or various environmental conditions near a device. **Environment sensors** monitor relative ambient humidity, illuminance, ambient pressure, and ambient temperature [17]. The output values from environment sensors are not directly influenced by a user's behavior when using a smartphone, which makes them irrelevant for behavioral authentication.

The next group is called **position sensors**. These are used to determine the position of a device. The **proximity sensor** returns a value to estimate how close is a nearby object to the front of the device. However, it is usual that this sensor returns only a binary value indicating if the device is close to a user's face or not, which is widely used to disable the touchscreen during a phone call. For its insensitivity, it is not a useful behavioral biometrics. The **magnetometer sensor** provides geomagnetic field strength values for the three coordinate axes. The remaining sensors from this group are a fusion of an accelerometer, gyroscope, or magnetometer sensor. They are referred to as software sensors because their values do not come directly from hardware components.

A **geomagnetic rotation vector sensor** reports a device's position relative to the magnetic north pole by fusing an accelerometer with a magnetometer. It allows a device to be used as a compass. **Game rotation vector sensor**, in contrast, reports a device's orientation in an application's frame of reference. By documentation [18] specifics, it must not use a magnetometer and only fuses an accelerometer and gyroscope. Ideally, a phone rotated and returned to the same real-world orientation should report the same game rotation vector values, which is helpful in games or virtual reality development [19].

The last group contains motion sensors. These sensors reflect direct user input such as tilt or shake, making them the most attractive group for behavioral biometrics analysis. The two most discussed sensors in related work are accelerometers and gyroscopes. These are also the only ones defined as hardware sensors and are also essential for extending user experience. **Accelerometer** reports an acceleration force applied to a device in all three axes, and **gyroscope** outputs angular velocity in all three axes. Together, they can detect a device's movement and enable a dynamic gaming experience, responding to lifting a phone, estimating the number of steps a user takes, or improving navigation systems. Anyway, for our diploma thesis, it is important that they can capture micro-movements in a user's hands.

On top of that, Android Open Source Project provides three software-based motion sensors: a gravity sensor, a linear acceleration sensor, and the already mentioned rotation vector sensor. All three rely on a gyroscope and accelerometer, and so if one of the two is not built into a device, none of the three derivatives is available for use. The **gravity sensor** indicates the direction of gravity. When a device is at rest, the output of a gravity sensor should be identical to that of an accelerometer. The **linear acceleration sensor** provides acceleration cleansed of gravity force [20].

Data on the application level, even from the base hardware sensors, do not represent the raw output of physical sensors. Correction such as bias or temperature compensation is applied to provide more stable results [18].

4.2 Sensors Programmatically

Many independent manufacturers produce Android-powered devices. To ensure some level of consistency, Android Open Source Project maintains Compatibility Definition Document [21] with requirements and recommendations. Regarding sensors, it only defines the expected behavior when using Android Sensor API and does not directly target the physical sensors.

Programmatically a sensor is represented by a `Sensor` class accessible from a system service `SensorManager` via a sensor type name. The type names are defined in the Android Compatibility Definition Document and determine how the referenced sensor behaves and what data it provides. A sensor listener must be registered at the system service to read data from a sensor. The listener should implement `onSensorChanged()` callback function, which despite the name continuously at a specified rate, receives `SensorEvent` objects of the chosen sensor representing a sensor's sample and contains:

Timestamp at which the event was measured in nanoseconds since the device's boot, including inactive time.

Data three-dimensional vector, which interpretation depends on the sensor type.

Accuracy value in range $[0, 3]$. The higher, the better. Zero accuracy means an unreliable sensor.

During the listener registration, the required sampling frequency must be specified. Android uses a batching optimization, instead of sharing sensor events individually from the hardware component, they are buffered in a queue and processed in a batch [22].

The Android Sensor API is linked to the underlying hardware through the Framework layer [23]. When a first application registers a listener to a sensor, the Framework sends a request to activate the corresponding sensor. When additional applications register to the same sensor, the Framework considers the requirements of each application and sends the updated requested parameters to the hardware abstraction layer. The new sampling frequency will be the maximum of the requested sampling frequencies, and so some applications might receive events at a higher frequency than requested. When the last application registered to a sensor unregisters from it, the framework sends a request to deactivate the sensor so power is not consumed unnecessarily.

4.3 Touch Events Overview

From the Android architectural perspective, a touchscreen is an input device, and so the data it produces go through a different pipeline than a sensor's samples. At the lowest layer, the physical touchscreen produces signals, which are further reported to the system by bus interruptions. The device-specific signals are then translated by a device driver in the Linux kernel into a standard Linux input protocol event format. This process includes calibration to provide the desired accuracy and responsiveness. Touchscreens measure touch properties using a device-specific scale, thus not directly meaningful to applications without giving the physical characteristics as a context. The Android system uses calibration parameters encoded in touchscreen configuration files to transform and normalize the values. Finally, the low-level touch event representation is then passed to the stateful Android InputReader component, which decodes the input events, updates the internal state about the details of each active touch, and then sends them to the InputDispatcher, which forwards them to the appropriate application window [24].

The touch events are reported to an Android application as a `MotionEvent` object, which includes:

x- and y-coordinates specify the touch position in display pixels.

Phase describes a phase of a touch event. Phase Down starts the touch action by putting a finger on the screen, and phase Up ends the touch action when lifting the finger from the screen. Touch events created by moving a finger on the screen are reported as phase Move.

Timestamp of the event in milliseconds since boot, not counting time spent in deep sleep.

Pressure describes the approximate physical pressure applied to the touch device as a normalized value between $[0, 1]$. The value is non-zero when a finger touches the device and zero otherwise. Applications can use pressure information to implement pressure-sensitive drawing and other effects.

Touch Major reports the approximate length of the longer dimension of the touch contact in pixels.

Touch Minor reports the approximate length of the shorter dimension of the touch contact in pixels or is equal to the Touch Major when only the approximate diameter of the contact area can be measured.

Size field describes the normalized contact area size of the touch relative to the largest possible touch that the touchscreen can sense. The value is usually derived from Touch Major and Minor.

Orientation describes the inclination of the touch as an angular measurement.

Pointer ID field is a non-negative integer identifying a touch independently when multiple touches are active during a multi-touch action. When multiple fingers touch the device, each finger should be assigned a distinct ID used as long as the finger remains in contact. IDs are reused when their associated touch action finishes.

The only required properties by the specification [24] are x- and y-coordinates and the phase. In contrast to the Sensor API, there is no rate specifying the sampling delay of the touch events. In addition, while the sensor event's timestamp is interpreted as the number of nanoseconds since the last boot, including the time the device was in a deep sleep, the touch event's timestamp shows milliseconds since boot, not counting time spent in deep sleep, which makes them incomparable.

4.4 Touch Events Programmatically

From a user's perspective, an Android application consists of several windows between which a user can navigate. Each of the windows consists of components a user can interact with. On the source code level, those components are represented by `View` classes.

Touch events are delivered to a `View` as a stream through a registered listener callback. However, consistency is not guaranteed. Some events may be dropped or modified before being delivered, and so it is advised to be prepared to handle anomalous situations, such as receiving a new touch event with phase Up, without prior Down [25].

Developing an Android application for behavioral biometrics data collection would mean registering a listener callback for every single `View` with which a user can interact. In terms of source code modularity, it would be beneficial to draw a full-screen overlay that reads the touch event stream before being delivered to the target `View`. Or in other words, perform a Clickjacking on the owned application. A paper [4] written in 2017 studies security concerns of an Android permission, which provides the capability to draw transparent overlays able to catch a user touch event. On the technical side, however, the overlay's behavior is controlled by a series of flags, using which a developer has to decide between letting the touch event go through the overlay without reading it, or consuming the touch event by the overlay without forwarding, thus making the user interface non-interactive. Although the paper proposes a way to bypass Android's security mechanisms, we want to rely on conventional methods of using Android Software Development Kit. Therefore, a listener callback will be registered for each interactive component.

Registering a listener to each interactive component makes it impossible to read touch events from a standard software keyboard because it is, in fact, an external application. Mobile banking applications do not use system keyboards for security reasons but implement a custom one. Therefore, we will adopt the same approach when developing the BehavioralCollector. Registering a listener callback per an interactive component comes with the extra benefit of filtering out unintentional touches outside the workspace and assigning a label of a pressed key to a touch event at the device level.

Chapter 5

Client-side Application

This chapter describes the design and implementation of the Android application called BehavioralCollector, representing the client side of the BAS. Its user interface is designed to simulate a typical mobile banking application. It is implemented so it continuously collects behavioral biometrics in the background and sends it regularly to the BehavioralProcessor web service, described in Chapter 6. The BehavioralCollector will be used firstly for data collection and then to demonstrate the complete authentication system.

The BehavioralCollector clearly cannot substitute genuine mobile banking applications regardless of the level of authenticity the user interface gives. Based on widely used mobile banking applications, we identified typical activities that need to be done in order to send a banking transaction. The BehavioralCollector simulates these typical activities with its user interface.

However, it should be noted that available mobile banking applications offer flexibility in the sense of navigating through the user interface. This degree of freedom is not possible in BehavioralCollector, and thus collected behavioral biometrics are captured during predetermined activities of users in a strictly defined order. The chosen user interface, therefore, defines which user activities will be further explored.

The BehavioralCollector was made publicly available on Google Play [5], allowing me to distribute it among volunteers efficiently and trustworthy. It is assumed that volunteers installed the BehavioralCollector on their own devices that they are familiar with. We did not supervise or instruct the volunteers on how to hold a smartphone or in which body position to stay. It can be assumed that they used the BehavioralCollector as it was natural for them.

5.1 Collected Data

Chapter 2 summarises related work to determine what data should be collected from smartphone devices to enable behavioral authentication. The survey concluded that touchscreen, accelerometer, and gyroscope data can lead to an acceptable behavioral authentication system. However, today's smartphones have other sensors that might be worth exploring. Since this work is a Proof of Concept on a limited scale, the significantly larger data volume that would have to be transmitted and persisted will not cause serious difficulties. Obtaining more data is preferable to repeating the data collection.

The magnetometer sensor was included in several studies of behavioral authentication. Although its values are influenced by environmental bias like iron distortions, it is still a sensitive hardware component that could capture micro-movements close to and during a touch. The game rotation vector is preferred to the geomagnetic rotation vector influenced by a magnetic north pole. It reduces the bias caused by a user's orientation and thus is more focused on the motion produced during phone interactions.

Data collection includes touch events and sensor events from a magnetometer, and all the motion sensors, namely an accelerometer, gyroscope, gravity sensor, linear acceleration sensor, and game rotation vector sensor.

5.2 Designing the User Interface

The BehavioralCollector simulates the typical activities done in a mobile banking application. Generally, any application that protects sensitive data starts with a reliable authentication procedure, such as typing a PIN or, less frequently, a password. After logging in, a menu of supported operations is displayed, from which a user can choose by scrolling and clicking. These steps are necessary to take prior to any further action and, therefore, would also be done by a potential attacker. Assuming an attacker intends to make a banking transaction to an account not stored in the victim's records, the attacker would type the account identification number. Hence, the BehavioralCollector collects behavioral biometrics of entering a PIN, clicking on buttons, scrolling through a list, and typing a ten-digit number.

A login activity is straightforward to simulate. The only design decision is wheater to define a global PIN, a set of rotating PINs, or let participants select a personal PIN. Defining a single global PIN for all participants limits the variability in the data and makes data analysis easier. We would be assured the users' discrimination is based on their behavior and not by typing different digit sequences. On the other hand, the resulting analysis could be biased by the PIN selection. Requesting a PIN *5-5-5-5* would probably result in significantly different behavioral biometrics than forcing users to click on the key in keyboard corners, i.e., *1-3-7-9*. Defining a set of PINs was chosen by [10]. From my point of view, changing a PIN does not happen frequently, and I want the PIN to be entered from a user's memory. Therefore, the **login activity** lets participants select a personal PIN on the first run of BehavioralCollector, which has to be repeated on each subsequent run. This approach captures the most realistic behavioral biometrics.

Specific actions done to navigate through a mobile banking application can vary considerably. Typically, however, this consists of scrolling a menu list and clicking a button to choose the operation. I divided these two activities. The first focuses on capturing behavioral biometrics during clicking on the screen, and the other captures characteristics during scrolling gestures. The **clicking activity** consists of five buttons that a user must click on. One is in the middle, and the rest are in each corner of the touchscreen. The **scrolling activity** includes a vertical list of numbers, which participants must scroll through to find and tap on two negative values. The position of the two negative values is generated randomly so that a negative value is always within a group of ten non-negative values. The intention is to force a participant to truly search the list by applying a scroll gesture several times, without just flicking to the end of the list, and to prevent the values from being generated close to each other.

Finally, the **typing activity** simulates typing of an account number. It is designed so that a participant has to copy-type a randomly generated ten-digit number. Figure 5.1 shows the user interface design of each activity as screenshots captured using a smartphone emulator in Android Studio [26].

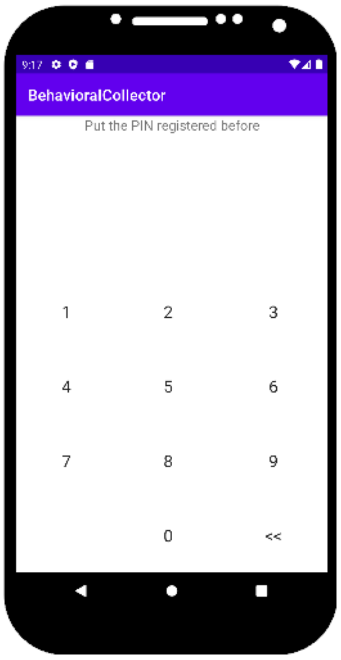
5.3 Architectural Design

This section does not aim to describe how to develop an Android application. This diploma thesis addresses a broader perspective, and describing implementation of an Android application in detail would be unnecessarily overwhelming. However, it is worth presenting the key architectural components of the BehavioralCollector. Figure 5.3 shows a simplified class diagram of the structure of the BehavioralCollector, which will be further described.

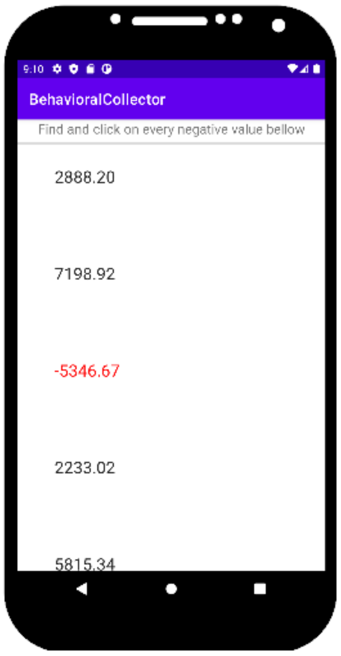
An Android application is built on several independent windows that a user interacts with. From the Android code perspective, the window is represented by an `AppCompatActivity` class. As the user navigates through an application, the class instances change their state in the lifecycle [27], and so the class provides callbacks to let the instance know when the state changes, so a reaction to creating, stopping, or resuming a window can be implemented.

Although the four simulated activities forming the BehavioralCollector are independent from the user's perspective, they share the same way of registering the sensor listeners. When starting an activity, it is necessary to register an event listener for each sensor, and in turn, when leaving the activity, it is expected to unregister the listeners to avoid useless resource consumption. It is, therefore, convenient to create a parent class `SensorActivity` that extends the `AppCompatActivity` class and overrides the `onResume()` and `onPause()` callback methods to handle sensor listener management. Since the data collection does not take place in a controlled environment, various devices can contribute. To unify the sample rates, categories of 50, 100, 200, and 400 Hz were created. Based on a sensor's capability, a listener is registered to the highest possible sample rate according to the categories.

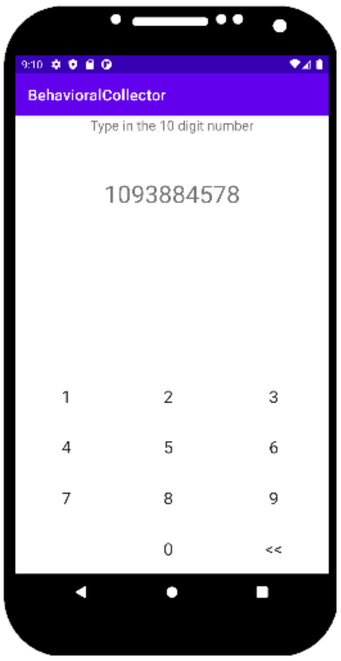
The activities also share the backend procedure of collecting and sending behavioral biometrics. Therefore, I implemented the `EventCollector` class as a singleton so that only one instance of this class exists at runtime, shared for each activity. As a participant can focus on only one activity at a time, there is no reason to be concerned with causing a race condition when accessing the data structures inside the singleton.



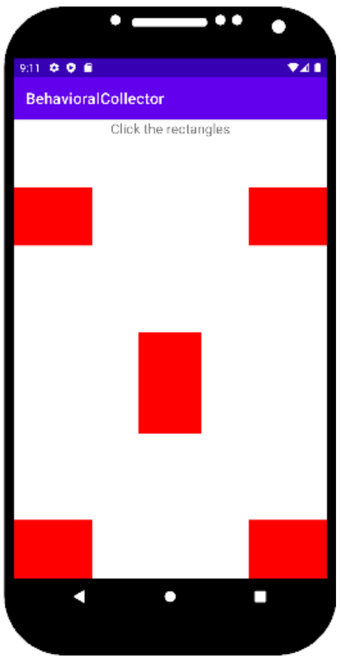
(a) Login Activity



(b) Scrolling Activity



(c) Typing Activity



(d) Clicking Activity

Figure 5.1: Design of the four simulated activities of the BehavioralCollector

The `EventCollector` is initialized at the start of the BehavioralCollector. Each BehavioralCollector run is called a session and is identified with a randomly generated UUID. To keep a record of the device, generating a random identifier and storing it in the phone's memory might be a too fragile option. The data of the BehavioralCollector could potentially be deleted by a user or a phone cleaning application, resulting in the loss of the identifier. Therefore a more reliable identifier is used. `ANDROID_ID` is a hexadecimal string randomly generated when a user first sets up a device and should remain unchanged for the lifetime of the user's device. It is, however, unique only to a combination of a user's account and device [28]. If a device is controlled with multiple Google accounts, each has a different `ANDROID_ID`. Although some unique hardware identification numbers could be used, I do not want to be too intrusive. The assumption of participants not changing their Google accounts during the data collection is not overly restrictive.

The `EventCollector` implements method `addEvent()` overloaded for both event types, `SensorEvent` and `MotionEvent`. Sensor listener is implemented, so it just forwards the received `SensorEvent` to `EventCollector`. Touch event listeners pass the `MotionEvent` together with a key label representing the name of a button targeted by a user. The label describes the digit pressed on the software keyboard during login and typing activity or the position of a button in the clicking activity, such as middle, top left, top right, bottom left, and bottom right. `EventCollector` transforms a received event and appends it to a buffer shared for both event types.

The transformation steps are taken to handle the different timestamps representation both events use. While the `SensorEvent`'s timestamp represents nanoseconds since the device's boot, including inactive time, the timestamp of a `MotionEvent` is milliseconds since boot, excluding the time spent in deep sleep. Modifying event objects with the current time is not the right way due to a significant variable delay between producing the event by a sensor or a touchscreen and modifying it in the `EventCollector`, which is additionally amplified by the batching optimization. Instead, at the start of an activity, the current times of both representation formats are retrieved and then used as a reference value that is subtracted from the corresponding event type. Therefore, the new timestamp interpretation for both event types is approximately nanoseconds since an activity started.

After a user finishes an activity, the BehavioralCollector sends the behavioral biometrics it collected to the BehavioralProcessor. Android system, by default, does not allow performing long-lasting operations, such as network access, in the main thread as that would block the user interface and potentially result in an application not responding dialogue [29]. As the BehavioralProcessor performs only basic HTTP requests without a user noticing, there is no need to use some advanced frameworks for handling network operations. Instead, we use a lightweight `HttpURLConnection` [30] dispatched in a dedicated thread from a thread pool. This functionality is wrapped in `HttpClientExecutor` class. The data are sent in a HTTP Post method payload serialized into JSON format and compressed using gzip to increase the network throughput during the data-intensive communication. The data sending is triggered by finishing an activity using the `EventCollector`'s `flush()` method, which also clears the event buffer and initiates a new activity. The resulting data structure represented as a JSON has the following form:

```
{
  "activityId": <uuid>,
  "activityType": {Login,Scrolling,Typing,Clicking}Activity,
  "timestamp": <ms on activity start>,
  "session": {
    "sessionId": <uuid>,
    "timestamp": <ms on session start>,
    "channel": <smartphone model name>,
    "deviceType": "SMARTPHONE",
    "deviceId": <ANDROID_ID>
  }
  "sampleRates": {
    "accelerometer": <accelerometer sampling rate>,
    "gyroscope": <gyroscope sampling rate>,
    ...
  },
  "events": [ModifiedMotionEvent, ModifiedSensorEvent, ...],
  "__debug": <PIN or 10-digit challenge>
}
```

Figure 5.2: Data structure sent to the server-side represented as a JSON

It is necessary to collect data from a single user over multiple days to consider potential intersession behavioral changes. To avoid a situation where a participant is motivated enough to install the BehavioralCollector but contributes with only a single session, our application included a reminder notification that pops up after 24 hours of inactivity during the data collection stage.

5.4 Publishing the BehavioralCollector

To make the BehavioralCollector accessible to volunteers, the best approach is to publish it on Google Play. After the Android application is developed and functional in the Android Studio emulators, there is no need to make any significant changes to the source code before publishing it.

First, a developer account must be created by filling out multiple forms with the application's description, purpose, and requested permissions. Once submitted, the approval process takes a few weeks before the application can be uploaded and accessed by volunteers through a standard link on Google Play [5].

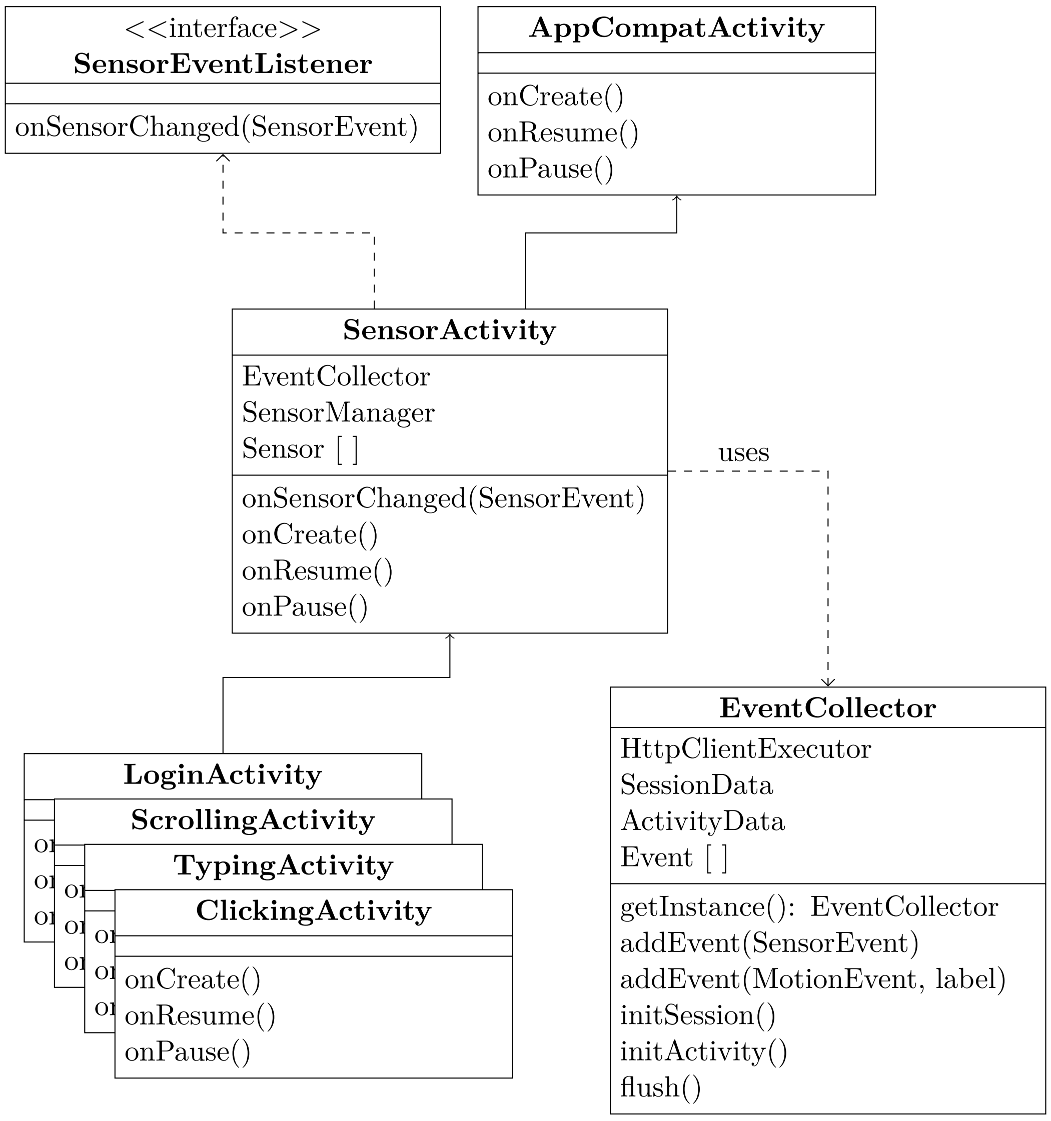


Figure 5.3: Class diagram of the core of the BehavioralCollector application

Chapter 6

Server-side Application

This chapter describes the development of the server-side web service called BehavioralProcessor. It is the first component of the BAS. The BehavioralProcessor receives behavioral biometrics captured during a user's interaction with one of the four simulated activities in the BehavioralCollector. The received data are then passed to the ScoringModule, which evaluates the behavioral biometrics and assigns it an authentication score. The BehavioralProcessor then stores behavioral biometrics and the authentication score in the BBD. The BehavioralProcessor also provides a REST API endpoint for the relevant authority to query a session's authentication score. The BehavioralProcessor should meet the following requirements:

1. The BehavioralProcessor processes the received data asynchronously to avoid long-lasting connections with Android devices.
2. The received data are forwarded to the ScoringModule, which assigns them an authentication score.
3. The received data and the assigned authentication score are persisted in the BBD.
4. The BehavioralProcessor provides a session's authentication score to the relevant authority on demand.

At this stage of this diploma thesis, we want to collect behavioral biometrics to provide a dataset for the second stage of the thesis without evaluating the received data. The ScoringModule is therefore bypassed for the first stage.

It is advisable to use a web application framework when developing a web server application. Using a framework comes with some resource overhead, but it allows the developer to focus solely on the core functionality, which makes the process easier. Furthermore, the resulting source code is easier to maintain since it adheres to the expected structure and design patterns. We have chosen *Spring Framework*, a widely used framework on the industry level built on Java [31]. It offers data access and integration modules, simplifying communication with other technologies used in this work. For instance

Redis, a fast and in-memory open-source key-value data store [32], and *PostgreSQL*, an open-source relational database system [33].

The BehavioralProcessor is composed of three services. The **queuing service** stores received data in an input queue, the **processing service** consumes and processes the queued data, and finally, the **reporting service** responds to relevant authority queries. This chapter only describes key components implemented to meet the requirements stated above and omits implementation details or commonly used design patterns.

6.1 Queuing Service

Behavioral biometrics collected by the BehavioralCollector are sent to the Behavioral-Processor as a compressed JSON included in HTTP Post payload. To make the data ingestion available, the BehavioralProcessor defines a REST API endpoint. In Spring terminology, a class defining endpoints is referred to as **controller**. Although the controller converts JSON payload into a Java object representation, called Data Transfer Object (DTO), internally, it cannot handle compressed data by default. Therefore, an uncompression step is taken before accepting the data in the controller. After the received data conversion, the content of the DTO reflects the JSON structure sent by the BehavioralCollector application. An object is considered invalid if it does not contain all the necessary metadata, such as device, session, and activity identification, and valid data for further analysis. Invalid DTOs are discarded.

It is not best practice for a HTTP request to maintain connections for extended time periods, and the processing of received data can potentially be time-consuming. Additionally, a BehavioralCollector client expects just an acknowledgment of the behavioral biometrics being received correctly. Therefore, it is preferable to use asynchronous processing.

The asynchronicity is achieved by an input queue. Implementing the queue in the BehavioralProcessor's memory is not a scalable solution as it offers only a very limited size and would put a load on the whole web service. Instead, we use Redis as an external queuing technology. Although recent versions of Redis add various messaging functionality [34], for our needs, its native list structure is sufficient to work as an input queue. Redis implements the list structure as a linked list of string values assigned to a specific key. New elements can be pushed from both the list's head and tail at a constant time as well as reading and extracting.

Since Redis stores the data as sting values, the DTO must be serialized again. To make the temporary data storing less resource demanding, we use *Protocol-Buffers* [35] message format for serialization, supposedly [36] much less resource-intensive and six times faster than JSON serialization. However, this long-term effectiveness requires much more effort to put it to work. While JSON serialization is done by calling a predefined generic function, the Protocol-Buffers need a definition of proprietary data structures, which must be compiled into Java-compatible classes and included in the Java project. Thanks to a Java plugin [37], the compilation and inclusion steps are part of the project's building procedure. This means that only `.proto` files that reflect the DTO need to be defined to specify the class attribute names and their primitive datatypes. This significantly more complex procedure is explained by an intention to be space-efficient and platform-neutral. Identical `.proto` files can also be compiled into other programming languages than Java. This makes it possible to read serialized data from different projects.

After the DTO is mapped to the Protocol-Buffer message object, it is ready to be passed to the input queue. Spring offers a convenient Redis connector abstraction called *RedisTemplate*, which is, in fact, a Redis connection wrapper. The underlying connector package still has to be defined and configured manually. However, the RedisTemplate significantly improves the source code readability, as it handles internally the object conversion, connection pooling, and other issues associated with maintaining a connection. So the service's logic implementation just pushes the object to the Redis list at the specified key using the RedisTemplate. If Redis reaches its memory limit, it will start to reply with an error to write commands but will continue to accept read-only commands [38]. Hence if the input queue reaches its maximum size, subsequently received data will be dropped, and a BehavioralCollector client will receive an exception about the data not being queued. Figure 6.1 visualize the described process of receiving behavioral biometrics from a BehavioralCollector client.

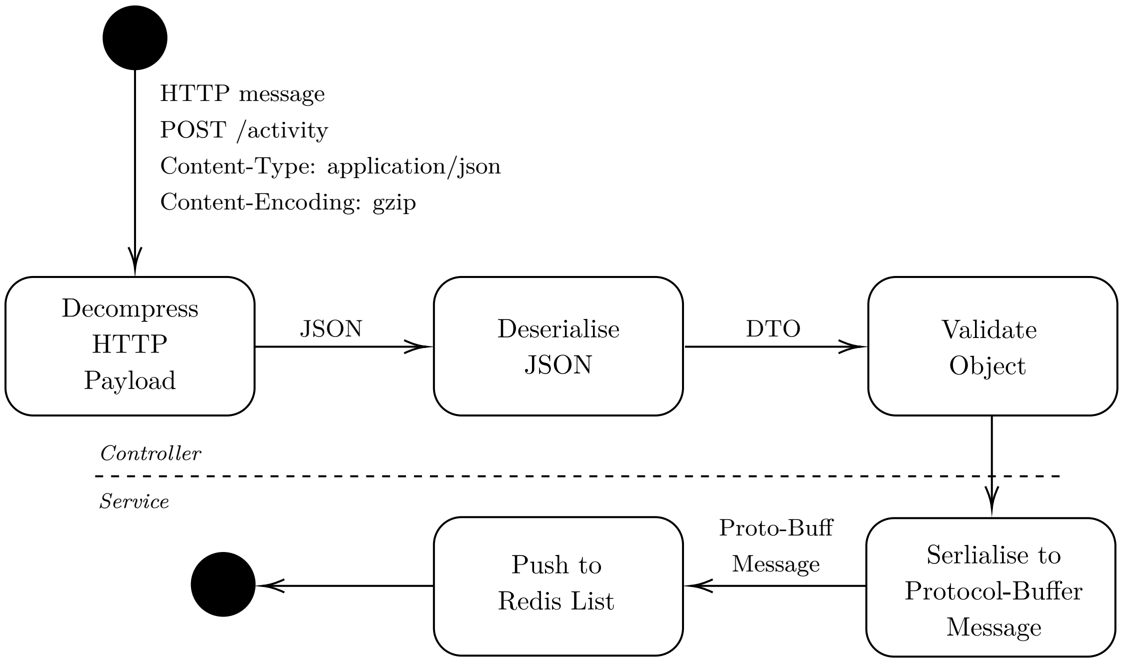


Figure 6.1: Activity diagram of receiving data from a BehavioralCollector client

6.2 Processing Service

The data processing service represents a three-step data pipeline. First, if the input queue is non-empty, it extracts an element from the head of the queue. Validation of the input has already been done when the data were received. Therefore it can proceed directly to a second step, built of a ScoringModule that computes an authentication score. And finally, after the ScoringModule completes the procedure, the data and authentication score are stored in the BBD. Although the ScoringModule has not been designed yet, its placeholder is created in the data pipeline implemented in this section.

A three-step data pipeline, consisting of reading from a source, processing the data, and storing them in another storage, is such a common design pattern that Spring has a framework for this kind of processing, called *Spring Batch* [39]. Spring Batch lets a developer focus on the processing logic without taking care of the background infrastructure. The essential functionality such as logging and tracing, transaction management, processing statistics, or resource management is included [39]. The framework keeps track of the data pipeline processing status whenever a new execution starts. It is, therefore, effortless to limit the number of concurrently running tasks and prevent duplicate processing. In addition, it is straightforward to scale the entire process. With only slight modifications, delegating some of the work to other computing machines or letting data processing run in parallel on multiple threads is possible.

To make the data pipeline operational, it is necessary to implement three interfaces `ItemReader`, `ItemProcessor`, and `ItemWriter` provided by the framework. The steps that build the data pipeline are visualized as a simplified activity diagram in Figure 6.2. The reader step tries to retrieve data from the input queue using the pop operation on the Redis list. If there are data to process, they are passed to the processing step.

The processing step just forwards the data to the ScoringModule. The data represents behavioral biometrics of a completed activity of the BehavioralCollector application, and so the ScoringModule responds with the evaluated authentication score of the activity. Both the data and the response is passed to the writer.

Spring Framework provides a data integration module for easy and efficient communication with a database system through Java Persistence API (JPA) support, which allows managing database operations programmatically. JPA can be seen as a compatibility layer that handles transitions between Java objects and relational databases. A Java object that mediates the form of data for the JPA is called an **entity**. The individual attributes in an entity should therefore be of primitive data types or have an associated rule to convert the non-primitive data type. If an entity contains a reference or a collection of references to other entities, they will also be in relation to each other at the database level. Create, Read, Update, and Delete operations on a specific entity are performed through a **repository**, which is a database connection abstraction provided by Spring Framework. So, to complete the writing step of batch processing, we define an entity for each data class. Our data model is activity-centric, meaning behavioral biometrics and details of the corresponding session are associated with a specific BehavoralCollector application's activity data object, as in the data structure described in Figure 5.2. To persist the activity-related data, an implementation of only a single repository that handles an activity entity is required.

Details of currently active sessions are cached in Redis during the last step of the data pipeline. That is an optimization to save relational database queries and reduce the response time of relevant authority's queries when the authentication score of a session is requested. The cache item is stored as a Redis Hash structure, which is an associative array. It does not hold behavioral biometrics but only metadata and the response of ScoringModule for each activity of that session. The item is evicted ten minutes after the last modification.

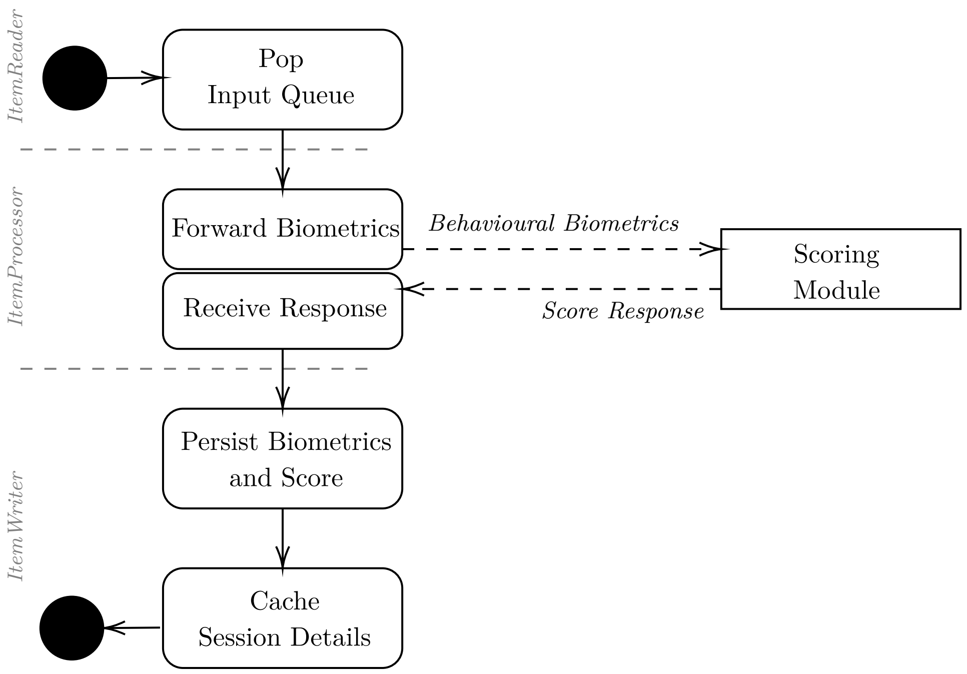


Figure 6.2: Activity diagram of behavioral biometrics processing service

At this stage of the diploma thesis, we want to collect behavioral biometrics to provide a dataset for the second stage of the thesis. The ScoringModule is therefore bypassed, hence an authentication score is not available.

In the first stage, when behavioral biometrics are collected without evaluation, the response time to new data in the input queue may not be crucial. However, once the ScoringModule is active, it is necessary to evaluate behavioral biometrics promptly. This is to ensure an authentication score is available when a user or attacker makes an attempt to do a critical operation, like a bank transaction. The reader step, therefore, uses a blocking pop operation. This means that the pop operation will be blocked until new data is available in the input queue. After behavioral biometrics are received, they go through the data pipeline, and a new blocking pop operation is initiated. If there are several connections that are blocked, the one that waits more time is served [40].

6.3 Reporting Service

The BAS does not react to the estimated authentication score but only provides the authentication score to the relevant authority. Therefore, the decision about rejecting or accepting a user is delegated to the relevant authority. A session's authentication score is provided via REST API request. As the response includes more attributes than the authentication score, it is referred to as **authentication report**.

The relevant authority requests an authentication report of a session based on the unique identification of a session. It works on the assumption that the relevant authority is also the smartphone application owner, so the session identification could be shared between the BAS, which receives it in the metadata of behavioral biometrics, and the relevant authority, that is able to read it from the smartphone application.

The authentication report consists of the identification of the queried session, the authentication score of the session, and set flags. The flags set is created by the ScoringModule and provides additional information about the session's authentication score. The steps taken from receiving a request to create an authentication report are visualized as an activity diagram in Figure 6.3.

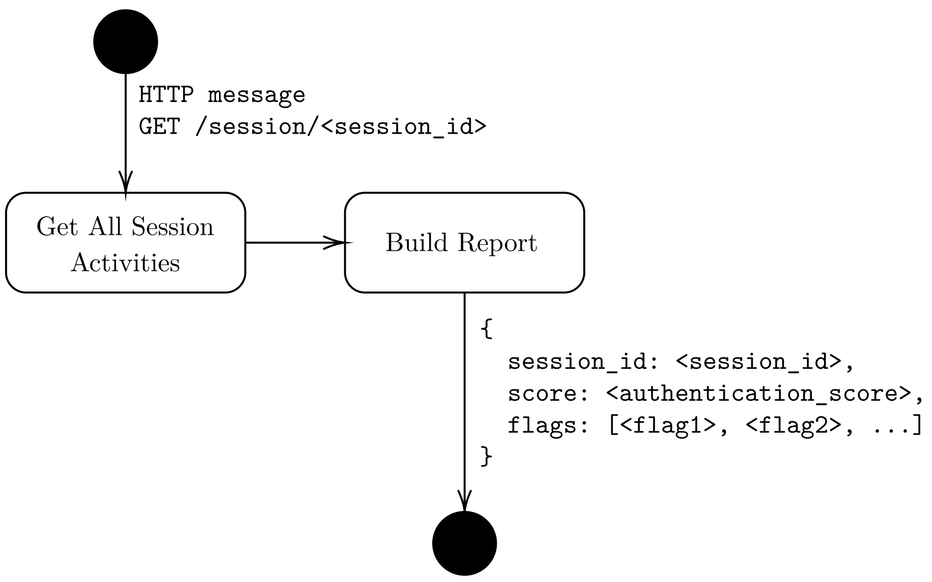

HTTP message
GET /session/<session_id>
Get All Session Activities
Build Report
{
session_id: <session_id>,
score: <authentication_score>,
flags: [<flag1>, <flag2>, ...]
}

Figure 6.3: Activity diagram of providing an authentication report

First, all already evaluated activities that belong to the queried session are retrieved. If the queried session is currently active, the activities are cached in Redis. Otherwise, it is necessary to retrieve them from the BBD.

Second, based on the retrieved activities of the session, the authentication report is built. Authentication scores of all available activities are combined using the best performance technique described in Chapter 7. The authentication report is sent to the relevant authority in JSON format.

6.4 User Assignment

A device producing behavioral biometrics is identified using the `ANDROID_ID`. The Android documentation [28] states that the `ANDROID_ID` is unique for a combination of a Google Account and device. It is generated on the device's first boot and wiped on a factory reset. That implies the `ANDROID_ID` for a specific user changes when any of the following events occurs:

1. User uses a different device.
2. User changes a Google Account.
3. User does a factory reset of the used device.

Presumably, when a user changes a Google Account or wipes the device, it does not affect the behavior while using the device. For these cases, it might be worth considering different identification of a user than `ANDROID_ID`.

During the data collection stage or later in a real-world scenario, it could be beneficial to be able to assign a user to a specific session and keep a record of which user is using which device. Therefore, another REST API endpoint is defined to allow assigning a user identification to a session. However, we did not have an opportunity to use this functionality during the data collection stage, as we were not aware that any participant in the data collection contributed with more than one `ANDROID_ID`.

6.5 Deployment

Finally, this application needs to be deployed on a publicly accessible server. Among all the computing resource providers, *Heroku* [41] fits our limited-scale Proof of Concept best. It offers an easy-to-deploy Platform as a Service development environment, which means a developer can get an instance of PostgreSQL or Redis in a matter of few clicks, avoiding the exertion of setting up and maintaining the server and the database instances. A Spring web application can be deployed just by pushing a Java Web Archive file through the Heroku command-line interface.

The database is generated automatically during the BehavioralProcessor start, with respect to entity definitions. The database schema is visualized in Appendix in Figure A.1.

Chapter 7

Data Analysis

In the first stage of this diploma thesis, a system enabling data collection was built. It consists of a client-side Android application called BehavioralCollector and a server-side application called BehavioralProcessor. The BehavioralCollector simulates a mobile banking application user interface through four different types of activities. When a user works on any of the activities, data from six sensors and a touchscreen is collected, forming **behavioral biometrics** of a user during an activity. The BehavioralCollector was published on the Google Play store [5], which allowed us to distribute it to volunteers efficiently, and enabled dataset creation.

The primary objective of the data analysis process is to propose a behavioral authentication scoring module. This module takes behavioral biometrics collected during an activity of the BehavioralCollector application and matches it to a set of behavioral biometrics produced by the same device during the same activity type in previous runs of the BehavioralCollector. A set of behavioral biometrics used for the matching is called a **historical context**. The output of this matching is an **authentication score**, interpretable as a probability that the activity is performed by an attacker.

In a single run of the BehavioralCollector, a user can complete multiple activities, forming a **session**. It is not convenient to evaluate a session based on multiple authentication scores. Therefore the second objective is to find a way to combine individual authentication scores per activity into a single authentication score, interpretable as a probability that the session is performed by an attacker.

Firstly, in the **Modeling Technique** section, a modeling technique to be used for the authentication scoring module is described, and different ways of combining authentication scores of individual activities into a single authentication score of a session are proposed.

The data analysis process starts with a **Data Description** section, in which the quantity and quality of the data contained in the dataset are described. The **Data Preprocessing** section describes the preprocessing used to prepare data for the feature extraction phase. In the **Feature Extraction** section, feature extraction techniques are described. Our approach is to explore each activity type of the BehavioralCollector application individually.

The feature extraction step outputs a large number of features for each activity type. However, we want to keep only those features that are informative for the behavioral authentication use case. A feature is informative for behavioral authentication if its values are stable for the same user yet discriminatory between users. The **Feature Selection** section describes a feature selection process based on the selected model's performance.

The **Combination Technique Selection** section describes the process of selecting the best-performing combination technique. Finally, the last section **Summary** summarizes this chapter.

7.1 Modeling Technique

Our dataset was constructed in an unsupervised environment and thus contains data from different smartphone models. Participants in the data collection could be in any body position, be in motion or be stationary, and grasp the device in any way. The first assumption of the data analysis process is that each smartphone device included in the dataset was controlled by exactly one user, and no two devices were controlled by the same user. Thus, we assume that the number of unique devices in the dataset is the same as the number of participants in the data collection. Therefore, a phrase *user's behavioral biometrics* has the meaning of *behavioral biometrics produced by a device*, and both represent the same part of the dataset.

The dataset only contains behavioral biometrics of legal owners of the devices in the dataset, therefore, an attack can only be simulated. The attack simulation works as follows. The dataset contains behavioral biometrics of n users $\mathcal{U} = \{u_1, ..., u_n\}$, $n \geq 2$. To simulate an attack on the user u_i, where $1 \leq i \leq n$, behavioral biometrics of users from set $\mathcal{U} \setminus \{u_i\}$ is used. In that case, the users from set $\mathcal{U} \setminus \{u_i\}$ are called **attackers**. This type of attack represents a zero-effort attack. The **zero-effort attack** is an attack where an attacker does not try to mimic a victim's behavior.

Our approach is activity-centric, meaning behavioral biometrics belongs to a specific activity completed in the BehavioralCollector application in a specific session by a specific user. Before a statistical evaluation, the raw data forming behavioral biometrics of a single activity are transformed into a feature vector. The **feature vector**, or **sample**, therefore, represents characteristics of a user's behavior during an activity of the BehavioralCollector application.

The behavioral scoring module aims to detect behavior that does not conform to expected behavior based on historical context. The statistical evaluation of the behavior is done by matching a new unseen feature vector with a set of feature vectors of past behavior characteristics made by the same user during the same activity type. The set of feature vectors is referred to as **feature matrix**. The result of this matching is a behavioral authentication score interpretable as a probability that the activity is being performed by an attacker.

7.1.1 Design Decision

Section 2.3 summarises algorithms used for behavioral authentication in related work. Several research papers rely on binary classification or template methods. However, none of those approaches is robust or scalable enough.

A user behaves differently in different situations. His behavioral biometrics are influenced by his body position or the phone grasp. It is expected the samples will create clusters according to this behavior change. A robust algorithm should therefore take into account the existence of these different clusters when evaluating the authentication score. Additionally, models that require a computationally intensive training phase problematize the gradual adaptation to behavior change. Training the model on the fly would be time ineffective, so the alternative is to persist the model and retrain it regularly or when a behavioral drift is detected. Thus, managing these models over a long time introduces additional implementation complexities.

To address this issue, we consider only anomaly detectors that operate by analyzing a historical context feature matrix with a new unseen feature vector added and provide an outlier score without requiring extensive training.

From the *scikit-learn* documentation [42], the most acceptable algorithm is Local Outlier Factor (LOF). This algorithm does not consider being an outlier as a binary outcome, and instead, it assigns a degree of an outlier-ness, called an outlier factor. The word local is used in the sense that only a limited number of the nearest neighborhoods of each sample are taken into account. This is particularly useful for datasets with complex structures, such as multiple clusters of various densities [43]. The resulting LOF's score can be understood as a ratio of a sample's local density to its k neighbours' local densities, with densities estimated by traditional distance metrics. If the ratio is higher than 1, the sample's local density is lower than the neighbors', which indicates an outlier. However, in real-world datasets, interpreting this value is not always so straightforward, which gave rise to Local Outlier Probabilities (LoOP) algorithm.

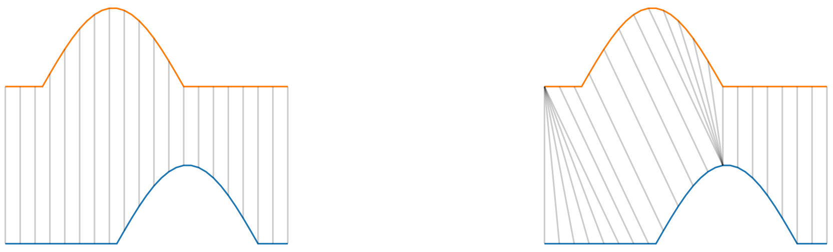
Authors of LoOP mention in the official paper [44] that interpretation of estimated LOF score differs from dataset to dataset. While an outlier score x in one dataset means an outlier, in another dataset, the score x is not extraordinary and thus does not imply an outlier. Therefore, the LOF scores cannot be compared between clusters. Authors propose a normalized scoring to become independent from the specific data distribution by mapping into the $[0, 1]$ range, interpretable as the probability of a given sample being an outlier. This is especially beneficial for this work, as the goal is to produce a probability that an attacker performs an activity. Using LoOP thus eliminates the post-processing part for converting a model's response into probability.

Output is not the only difference between the two algorithms. While LOF performs calculations in Euclidean space, LoOP makes use of probability space. Similarly, only k neighbors are included in the evaluation process. The probabilistic model assumes neighbors are approximately normally distributed around the queried sample. The local density is estimated using statistical extent parametrized by λ, which increases the outlier-ness score of samples that deviate more than a given λ-times the standard deviation. Therefore, the LoOP algorithm will be used in the scoring module.

7.1.2 Algorithm Adaptation

Authors of related work transform captured time series of sensors into scalar features by applying basic aggregation functions such as minimum, maximum, mean, or standard deviation. In this work, I exploit time series data more sophisticatedly using the Dynamic Time Warping (DTW) algorithm, which quantifies the dissimilarity of two time series.

While Euclidean distance between time series works in a predefined point-to-point manner, DTW is more complex. It computes distances between every possible pair of points in the two time series, out of which the shortest warping path is determined. This technique, therefore, takes into account a time shift between series as shown in Figure 7.1.



(a) Euclidean point-to-point distances (b) DTW distances

Figure 7.1: Comparision between time series dissimilarity measures

Our proposed feature set consists of various data types, such as time series, scalar or vector of scalars. Each of these benefits from different dissimilarity measures. Python's implementation of LoOP [45], used in this work, provides two ways of passing an input. Either as a feature matrix, which will be transferred to a distance matrix internally, or by passing precomputed distance matrix, giving a caller flexibility in using various distance metrics. There is, therefore, an intermediate step needed between the feature matrix and LoOP input.

Given feature matrix $\mathbf{F}^{n \times m}$ of n samples and m features, and vectors $\mathbf{d}$ and $\mathbf{w}$, where $|\mathbf{d}| = |\mathbf{w}| = m$, $d_i : \mathbf{F}_{\bullet i} \to \mathbf{M}^{n \times n}$ is a function mapping a column i of feature matrix to a pairwise distance matrix $\mathbf{M}^{n \times n}$, and $w_i \in \mathbb{R}^+$ is a weight of feature i, **average distance pairwise matrix A** is defined as follows:

$$\mathbf{A} = \frac{1}{m} \sum_{i=1}^{m} w_i \cdot d_i(\mathbf{F}_{\bullet i})$$

However, the LoOP implementation expects the average distance pairwise matrix in the form of two separate matrices $\mathbf{D}^{n \times k}$ and $\mathbf{N}^{n \times k}$, where $\mathbf{D_{i\bullet}}$ is a row vector of distances from sample i to k-nearest neighbors, and $\mathbf{N_{i\bullet}}$ their corresponding indices.

The output of LoOP is a vector $\mathbf{o}$, where o_i is the outlier-ness score interpretable as the probability that the sample i is an outlier.

Distance metrics

Different features may benefit from different distance metrics applied. As previously mentioned, the DTW algorithm compares features represented by time series. Various distance metrics can be applied to scalar features and affect the ability to discriminate between users. We test four different distance metrics, namely Euclidean, Manhattan, Chebyshev, and Cosine distance.

Feature normalization

It is advisable to standardize the feature values when using multiple features. Without proper normalization, each feature ranges different scales, so higher-scaled features would have a higher weight when averaging the distance matrices. There are two main approaches to choose between – min-max scaling and standard scaling.

The effect of min-max scaling is limiting the range of values into a predefined scale, typically $[0, 1]$ using the formula:

$$\frac{\mathbf{x} - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})},$$

with a column feature vector $\mathbf{x}$. This approach is, however, highly influenced by the minimum and maximum values in the vector, i.e., when an extraordinarily high value is in the vector, the other values are scaled to values close to zero.

Standard scaling subtracts the mean and scales to unit standard deviation by applying the formula:

$$\frac{\mathbf{x} - mean(\mathbf{x})}{std(\mathbf{x})}$$

The resulting values are interpretable as a number of standard deviations from the zero mean. This method will be further used.

To reduce the effect of outliers and make the feature space denser and closer to the normal distribution, which is beneficial for the statistical model, the logarithm function is applied before the standardization.

7.1.3 Score Combination

Users of the BehavioralCollector application can freely complete any number of activities in a single session. The algorithm presented in the previous sections evaluates each activity independently. However, producing a single authentication score for a session would be more convenient for the behavioral authentication system. The following text proposes multiple strategies, among which are decided later based on the performance of each strategy.

Each activity type has its model. The **first strategy** uses an authentication score estimated for an activity type with the best-performing model. If there is more than one activity of that type in a session, the first is chosen. So once a user completes the relevant activity, an authentication system can publish the overall session authentication score. However, studies claim this is not a wise procedure, as the discarded answers may contain some helpful information, and even a simple arithmetic mean of all the outputs is beneficial for the resulting accuracy [46]. Therefore, the **second strategy** to test is the arithmetic mean of all available activity scores.

Each activity is designed to capture different behavioral biometrics. The individual scores can thus be viewed as incrementally observing new evidence of the authenticity of the user's behavior. The **third strategy** is therefore a Bayes' rule [47], used in the following way:

- $\mathcal{P}(attacker)$ denotes the prior probability of an activity being performed by an attacker,
- $\mathcal{P}(\mathcal{A} \mid attacker)$ denotes the likelihood of observing activity's behavioral biometrics $\mathcal{A}$ given the activity is performed by an attacker (i.e. the model's output),
- $\mathcal{P}(attacker \mid \mathcal{A})$ is the posterior probability of an activity $\mathcal{A}$ being performed by an attacker,
- $\mathcal{P}(attacker \mid \mathcal{S})$ is the posterior probability of a session $\mathcal{S}$ being controlled by an attacker

$$\frac{\mathcal{P}(attacker)}{\mathcal{P}(\neg attacker)} \times \frac{\mathcal{P}(\mathcal{A} \mid attacker)}{\mathcal{P}(\mathcal{A} \mid \neg attacker)} = \frac{\mathcal{P}(attacker \mid \mathcal{A})}{\mathcal{P}(\neg attacker \mid \mathcal{A})},$$

because odds for probability p is expressed as $odds(p) := \frac{p}{1-p}$, the equation could be simplified to:

$$odds\,(\mathcal{P}(attacker)) \times odds(\mathcal{P}(\mathcal{A} \mid attacker)) = odds(\mathcal{P}(attacker \mid \mathcal{A})),$$

and for the incremental updates, where n is the number of activities in a session:

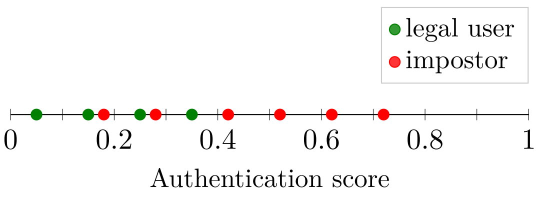
$$odds(\mathcal{P}(attacker \mid \mathcal{S})) = odds(\mathcal{P}(attacker)) \times \prod_{i=1}^{n} odds(\mathcal{P}(\mathcal{A}_i \mid attacker)),$$

and to convert odds o back to probability: $probab(o) := \frac{o}{1+o}$.

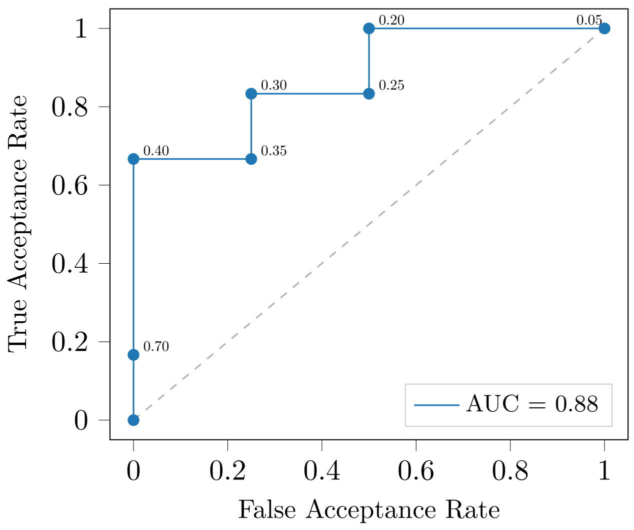
7.1.4 Model Performance Metric

Authors of related work measure their authentication algorithm performance using False Rejection Rate, i.e., the proportion of authentication decisions in which legal users are incorrectly rejected, and False Acceptance Rate, i.e., the proportion of authentication decisions in which attackers are incorrectly accepted. However, these performance metrics require binary authentication decisions – accepted or rejected. Although this could be solved by setting a threshold for our modeling technique, the threshold's value depends on the cost of error, as explained in the introductory chapter.

Our use case requires a threshold-independent performance metric. One approach might be to evaluate the performance based on all possible thresholds. Figure 7.2a illustrates possible model outputs for ten activities, from which four are produced by a legal user and the rest by attackers. Each point in the plot in Figure 7.2b then shows the False Acceptance Rate and True Acceptance Rate for a threshold, and these points form a curve called the receiver operating characteristic curve.



(a) Possible outputs of the LoOP



(b) Receiver Operating Characteristic Curve of the output from 7.2a

Figure 7.2: Output of a model and corresponding Receiver Operating Characteristic Curve visualization

The Area under the Receiver Operating Characteristic Curve (AUC) is a widely used metric to measure a model's performance irrespective of chosen classification threshold [48]. The AUC score is a value between $[0, 1]$. Score 0 implies a model's predictions are always wrong, whereas score 1 implies an infallible model. AUC close to 0.5, visualised by the dashed diagonal in Figure 7.2b, represents purely random decisions.

7.2 Data Description

Chapter 4 introduced the types of sensors that an Android device can include and listed data that can be read from a single touch event. This section focuses on data in our dataset collected from various devices using the BehavioralCollector application. First, the dataset is described. Then we examine the quantity and quality of the collected data. No specialized software is used for data analysis. The whole process is done manually in a Jupyter Notebook environment with a Python kernel with the help of commonly used packages like Numpy [49] and Pandas [50]. The dataset is stored in PostgreSQL relational database deployed in the Heroku environment and is accessed programmatically using SQLAlchemy toolkit [51]. The database schema is illustrated in the appendix in Figure A.1

The purpose of the BehavioralCollector application is to simulate typical actions done in a mobile banking application and consists of four activities.

1. Login Activity simulates a login process in which a user has to type a four-digit personal PIN registered on the first run of the BehavioralCollector application. So the data collection participants repeatedly wrote remembered digits.

2. Scrolling Activity captures behavioral biometrics during scrolling gestures. A user is led to scroll through a vertical list and click on two negative values in there. This simulates an options menu or transaction history scrolling.

3. Typing Activity simulates typing an account number by copy-typing a randomly generated ten-digit number.

4. Clicking Activity simulates navigating through an app by clicking on five rectangles in the middle and in the corners of a touchscreen.

I published the BehavioralCollector application on Google Play in October 2022 [5]. The generated link to install the application was first shared on several social networks: a Discord Channel for the Faculty of Information Technology CTU students, a Discord Channel for Instituto Superior Técnico in Lisbon students, and a Facebook Page for students accommodated at the Strahov dormitory. On top of that, a request to participate in the data collection was made during Profinit's data science team meeting. Finally, the application was also shared on *nomadtask.com*, a website where people share tasks to be done for a fee [52].

Regardless of the target group, simple instructions followed the request for support – *complete each activity three times in a session and do so for the next at least three days.* The reasoning behind repeating activities in one run is to collect more data during a single session. The three-day lower limit was set as a compromise between potential users' motivation to participate and the need to collect data across multiple sessions. Of course, the more days a user participates, the more likely it is to reveal patterns in collected behavioral biometrics.

The data analysis began in January 2023. Between the initial publishing and the beginning of the analysis, no major update of the BehavioralCollector affecting the collected data structure had not been released.

7.2.1 Metadata

When a new run of the BehavioralCollector starts, a new session is initiated and lasts until the application is terminated. A session is identified using randomly generated UUID on a device. A session is paired with a device through an Android ID [28] that is created by the Android system on the first device's boot and remains the same until the device is wiped or a user's Google account is changed. These identifications are stored in the *sessions* table.

A user is free to repeat the activities multiple times in a single session. There is no mechanism to choose a specific activity to complete, instead, a defined sequence, reflected in the enumeration list above, is followed. Of course, a user can exit the BehavioralCollector before completing each activity in the series or go back using the Android's built-in return button. A completed activity is represented as a row in *activities* table and contains randomly generated UUID as identification and an activity type, which is one of *LoginActivity*, *ScrollingActivity*, *TypingActivity* or *ClickingActivity*. An activity also has associated debug information used only for *LoginActivity* and *TypingActivity* type. In the first case, it contains the registered PIN; in the second case, it contains the randomly generated ten-digit number.

Sensors of different devices vary in sampling rate capability. As various devices participated in the data collection, the dataset also includes the selected sampling frequency of each sensor for each activity in the *sample_rates* table.

7.2.2 Data Quantity

According to the collected data, 49 distinct devices completed at least one activity. Data collection took place remotely, hence in an uncontrolled environment. Apparently, when an application is released on Google Play, the application is subjected to a test. That resulted in several devices in the dataset that ran only the first activity and set the same PIN every time just before a production release was approved. Moreover, these devices' sensor values are always constant, so the tests were probably taken in a virtual environment. Even some volunteers stopped contributing before reaching the three sessions limit. After the three-session constraint is applied, 27 devices are filtered out.

Of the 22 devices remaining, 12 of them completed three sessions. 7 devices contributed to the dataset with four to seven sessions, another two completed fifteen sessions, and the last did 48 sessions. Inspecting the number of activities, there are 288 *Login Activities*, 277 *Scrolling Activities*, 272 *Typing Activities*, and 243 *Clicking Activities*. The decreasing trend is caused by the three devices with the most sessions. Apparently, these users tended to contribute regularly but did not always have the opportunity to go through the whole process.

7.2.3 Touch Data

A row in a *touch_events* table represents a single touch event. A touch event is created every time a user puts a finger on a touchscreen, lifts the finger from the touchscreen, or moves with the finger on the touchscreen, denoted by a *phase* attribute of a touch event. While sliding a finger on a touchscreen, touch events are sampled at an unspecified rate. A *timestamp* attribute holds the number of nanoseconds relative to the beginning of an activity. Then, to identify the intention of a touch event, *x-coordinate* and *y-coordinate* is collected, together with a label of a key pressed. The key label is called *debug* in the dataset. For each touch event, *height* and *width* of the touchscreen in pixels is collected to enable touch coordinates normalization, which is necessary for future comparability across different devices.

Touchscreens installed in modern smartphones can handle multi-touch actions. Therefore Android assigns a *pointer id* to every new pointer touching the screen and remains the same for each subsequent touch event belonging to the pointer's stroke. In our case, the pointer is always a user's finger.

On top of these touch event properties, Android's touchscreen provides more details about a touch event, such as *force* applied to the touch screen, contact *area* size and diameters, such as *touch minor* and *touch major* whose approximate a finger's size. Unfortunately, all these particular values represent an approximation value normalised to device-specific range [25], hence are not inter-device comparable. And so, it is impossible to decide if the values reflect characteristics of a user or characteristics of a device. Therefore, none of these properties are considered in the data analysis.

To clarify the format of touchscreen data collected, see Table 7.1. The table contains five touch events, one per row. The example shows a participant typing numbers *4* and *5* on a numeric keyboard. The participant started the touchscreen interaction by putting a finger on the key *4* about three seconds after the activity started, which produced the first row of the table. The finger was not fixed during the stroke and moved a short distance, which produced the second row. Before the finger was lifted, the user put another finger on key *5*, causing a multi-touch action, which is recorded as a new *pointer id*. In the end, both fingers are lifted; first, the finger touching key *4*, then the finger touching key *5*.

timestamp [ns]	phase	debug	x [px]	y [px]	h [px]	w [px]	pointer id
$3020 \cdot 10^6$	BEGIN	4	141.8	1196.3	1920.0	1080.0	0
$3070 \cdot 10^6$	MOVED	4	145.5	1186.3	1920.0	1080.0	0
$3127 \cdot 10^6$	BEGIN	5	537.5	1240.3	1920.0	1080.0	1
$3151 \cdot 10^6$	ENDED	4	145.5	1186.3	1920.0	1080.0	0
$3227 \cdot 10^6$	ENDED	5	537.5	1240.3	1920.0	1080.0	1

Table 7.1: Example of collected touch events

7.2.4 Sensor Data

The dataset contains two groups of sensors. The motion sensor group has an accelerometer, gyroscope, gravity sensor, linear acceleration sensor, and game rotation sensor. From the position sensor group, there is only a magnetometer. There are six tables in the dataset to distinguish between the sensors on the data level, one for each sensor type. A row represents a single sample produced by a sensor. Irrespective of sensor type, each row includes *timestamp* representing nanoseconds since the activity starts, an *accuracy* of a sensor on scale 0 to 3, where 0 means unreliable sensor data and 3 is the best accuracy possible, and three-dimensional data x, y, z, which interpretation depends on a sensor's type.

An **accelerometer** measures the acceleration forces applied to a device in all three axes, as visualized on the right in Figure 7.3. An accelerometer includes gravitational acceleration, so an accelerometer's magnitude, computed as $m = \sqrt{x^2 + y^2 + z^2}$, should be approximately $9.81\, m/s^2$ while a device is stationary. We conduct an experiment to test the expected behavior and hence test an accelerometer's calibration.

For the following experiment, we sample data from each Xiaomi Mi A1 smartphone's sensor. First, we lay the phone on a table with the display facing up for five seconds; then, the phone is picked up by its upper edge to stay perpendicular to the table for the next five seconds. And finally, the phone is put down with the display facing down, also by holding the upper edge of the phone to keep in contact with the table. Sampled accelerometer's data are visualized as time series on the left in Figure 7.3, time series are denoised using a median filter.

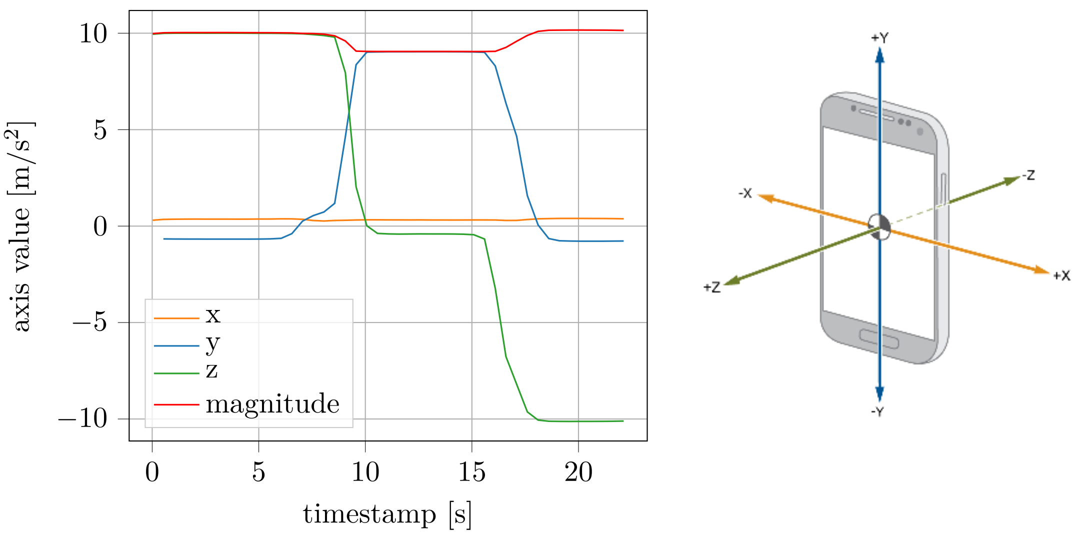


Figure 7.3: Denoised accelerometer data series (left) and axes visualization [53] (right)

In the first five seconds, the x and y axes should be zero, and the z-axis should hold the whole magnitude close to $9.81\, m/s^2$. We can see only a tiny difference from that expectation. During the perpendicular position, after ten seconds from the start, the z and x axes should be close to zero. Unexpectedly, the magnitude fell below 9.2 m/s^2 in the perpendicular position. At about 18 seconds, the phone was put back on the table with the display facing down. The values of the y and z axes were set as expected, but the magnitude rose to a value higher than in the first position to almost 10.2 m/s^2. These findings led us to check the calibration of accelerometers across multiple devices. Significant differences exist in measured magnitudes; therefore, two devices could produce different values despite staying in the same position.

A **gyroscope** measures a device's rotational speed in rad/s around the three physical axes. All three axes are close to zero while a device is stationary, regardless of position, as shown in Figure 7.4, which captures the same experiment as for the accelerometer above. Because the phone was picked up and later put down over the phone's bottom edge, the x-axis is the most active axis during the position change.

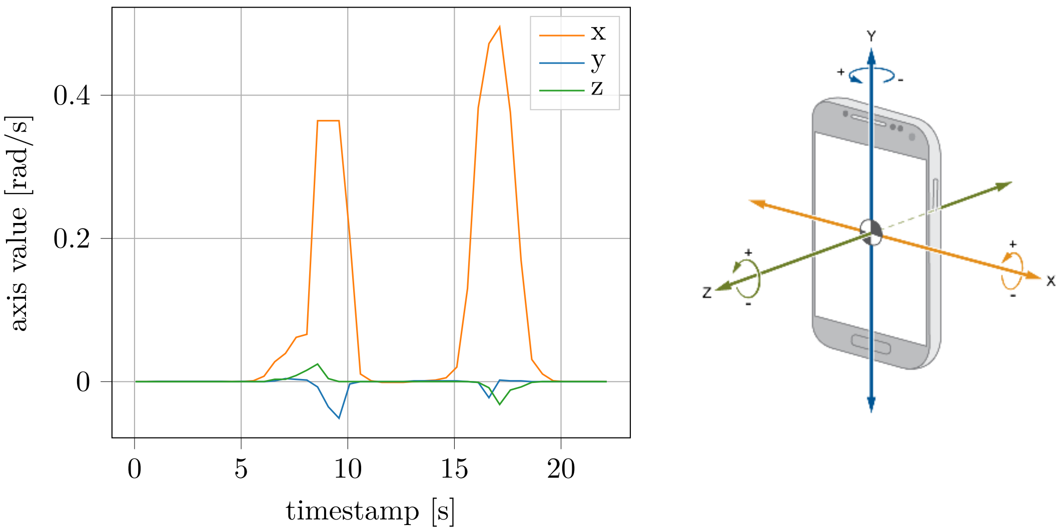


Figure 7.4: Denoised gyroscope data series (left) and axes visualization [54] (right)

A **magnetometer** sensor measures the geomagnetic field for all three axes in μT. Contrary to the previous motion sensors, values produced by a magnetometer are not influenced primarily by movements during interaction with an application. Instead, it is influenced by the orientation according to the north magnetic pole and possible ambient noise.

Linear acceleration sensor produces the acceleration force applied to a device on the three axes, excluding the gravity force. Compared to an accelerometer, this sensor filters out the additive component of gravity force, meaning the linear acceleration vector has a nonzero magnitude only when the phone is accelerating. **Gravity sensor** produces values also in m/s^2. When a gravity sensor's axes are added to linear acceleration sensor's axes values, the result should equal the original accelerometer's values.

The **game rotation vector** sensor is a fusion of an accelerometer and gyroscope sensors. It helps to orient in space relative to the start of an application. A rotated phone returning to the same orientation should report the same values [55]. Its three axes provide dimensionless units.

Only two devices out of the 22 in the dataset miss some of the requested sensors. One device only has an accelerometer sensor; the other misses gyroscope, linear acceleration, and rotation vector sensors. Various smartphone models in the dataset result in significant differences among the sensor sample rates.

7.3 Data Preprocessing

The previous section highlighted several inconsistencies in the dataset that prevent comparisons between different devices. This section describes how the data are preprocessed before features are extracted.

For the data preprocessing, it is worth abstracting the individual events. From now on, a group of subsequent touch events produced by the same finger, starting with putting the finger on a touchscreen and ending with lifting the finger from the touchscreen, are referred to as **stroke**. Individual sensor events, sampled during an activity run by the same sensor, are referred to as **sensor's time series**.

7.3.1 Strokes

Various devices contributed to the dataset. In fact, no two devices are of the same model, which causes an inability to analyze most touch event properties. From the remaining attributes, timestamp and phase have the same format and interpretation across devices. However, to exploit x- and y-coordinates, they should be normalized first.

To do that, the x-coordinate is divided by the screen width and the y-coordinate by the screen height. The interpretation of the normalized coordinates is a relative position to the display size. However, further analysis of the Typing Activity showed that more than this normalization is needed to eliminate the effect of various screen sizes in the dataset for the y-coordinate. The problem is that Android apparently does not strictly define if the top status bar or the bottom software navigation bar should influence the coordinate system. This is particularly evident when comparing a classical touchscreen with a modern frameless hole-punched display. The former keeps the coordinates inside the stated width and height, but the latter display type could produce values outside the scope. This effect does not apply to the x-axis as screen sides have no add-ons.

To eliminate this effect completely, there needs to be more data in the dataset to find the appropriate normalization method for the y-coordinate. The y-coordinate's absolute value is, therefore, omitted from further processing.

7.3.2 Time Series of Sensors

Authors of related work share an idea of how to preprocess the time series produced by each sensor during an activity. They synchronize strokes with sensors' time series using timestamps and extract those sections sampled during some specific user interaction. The rationale for this kind of processing is that while a user is idle, sensors only produce data affected by ambient noise. In contrast, if the series is limited to only those parts when a user touches a screen, a device receives impulses from the user's deliberate behavior.

We demonstrate the impact of a stroke on a senor's time series in Figure 7.5. It shows a three-dimensional time series of an accelerometer during a Clicking Activity completed by a random participant. The grey area marks the time sections when the participant touched the screen with a finger, here to click on rectangles in the middle or in the corners of the screen one by one. The time series shows a significant variation close to and during the period when a touch occurs. Selecting only the time series sections (or **subseries**) accompanied by touch could be indeed beneficial to discriminate users. The effect is particularly evident on the z-axis. Accelerometer's z-axis is perpendicular to the phone's display, so its acceleration changes more significantly than other axes when a force is applied to a touchscreen. The shown effect is also observed for different users.

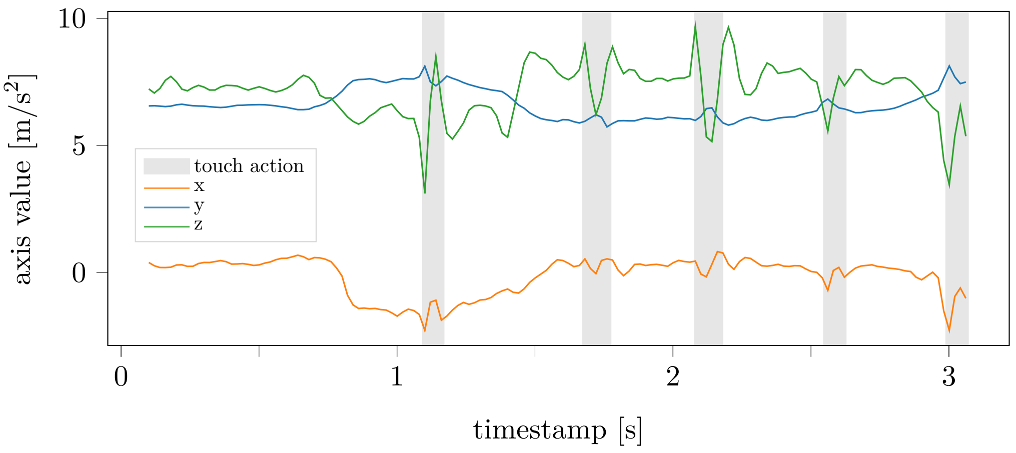


Figure 7.5: The impact of strokes to accelerometer values during Clicking Activity

Subseries normalization

It was previously mentioned that different devices might have different settings for sensor sample rates, and it was suspected that the motion sensors might be calibrated differently across devices. Thus, a method to normalize the subseries must be applied.

First, let us focus on the first problem. An extracted part of a time series during a stroke may be sparse due to an underpowered sensor or momentary limitations on the operating system side. To ensure sample rate consistency, time series are interpolated to obtain a new smoothed spline with a sample every 2 ms.

To address the potential differences in calibration, a time series is normalized by subtracting its mean and dividing by its standard deviation, i.e.

$$\frac{\mathbf{x} - mean(\mathbf{x})}{std(\mathbf{x})},$$

where $\mathbf{x}$ denotes a sensor's axis. The time series thus loses its original units. Instead, a sample's value now indicates the number of standard deviations from the zero mean.

Therefore, dissimilarity measures using the DTW algorithm will not be affected by sampling rate differences nor by absolute values of samples. The dissimilarity will only be measured based on the produced curve. Figure 7.6 shows the methodology described above.

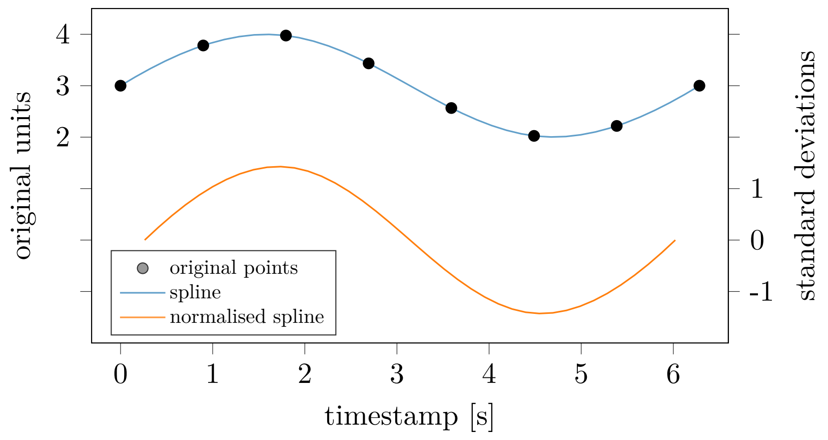


Figure 7.6: Time subsequence preprocessing technique.

Subseries length

After a closer look at Figure 7.5, it can be seen that some variation in the time series also happens just before a stroke begins and just after a stroke finishes. The device gets into motion before a user touches a touchscreen. This effect is most noticeable on smartphones with large displays. If such a device is held with a right hand, the effort to reach the top left corner with a thumb is accompanied by movement of the whole device. This effect is exploited by research paper [11] mentioned in Chapter 2. The research team built features based on micro-movements 100 ms before and after a stroke occurs, measuring the grip's instability.

Figure 7.7 illustrates the effect when a random user touches the top left rectangle during the Clicking Activity. We extracted and normalized the accelerometer's z-axis subseries of all the user's available interactions with the top left rectangle, shown as the background lines. The black line is the mean of those subseries. The red line is the rolling mean of those subseries with window size 10. Significant variation starts when the rolling mean's window moves away from the mean, which is 20 ms before the touch happens, in this case.

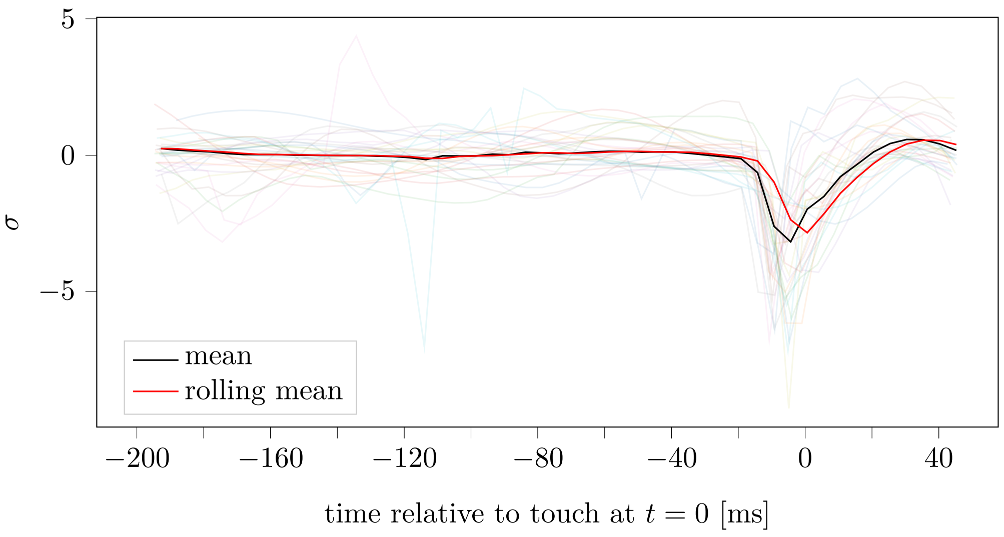


Figure 7.7: Visualization of variation in time series before a touch

The conclusion of the specific test visualized in Figure 7.7 would be that the best time epsilon neighborhood before a stroke targeted to the top left rectangle of the user is 20 ms. This test design is applied to all users, each outputting a preferred time epsilon. The final value defining the epsilon neighborhood before a stroke targeted to the top left rectangle is the average value of these individual outputs. The same approach is used for each interactive element of the BehavioralCollector application, for both time epsilon before and time epsilon after a stroke.

However, some users use their phones smoothly. Before ending a stroke, they already focus on the following action and do not necessarily put a device into the initial position. Therefore we consider three **epsilon neighborhood strategies** for extracting a sensor's subseries in the feature extraction process:

1. Both time epsilons are set to zero before and after a stroke. This effectively means only a subseries during a stroke is considered.

2. Time epsilon before a stroke is set to the precomputed value, and the time epsilon after a stroke is set to zero.

3. Both time epsilons before and after a stroke are set to precomputed values.

7.4 Feature Extraction

Each activity in the BehavioralCollector application was designed to simulate different actions done in a typical mobile banking application. The user interface of each activity type is depicted in Figure 5.1. Each activity type is treated individually during the feature extraction.

Our modeling technique, described in Section 7.1.2, firstly computes distance matrices for each feature separately, which could lead to obscure the discriminatory potential. Consider the following example in Figure 7.8. Combining *Feature 1* and *Feature 2* into two-dimensional space enables us to clearly distinguish the legal user's clusters from attackers' clusters. It is not, however, possible to separate them in a single dimension when projecting them on one of the axes. To overcome this situation, a combination of features, i.e., using a vector of scalars as a feature, is also considered.

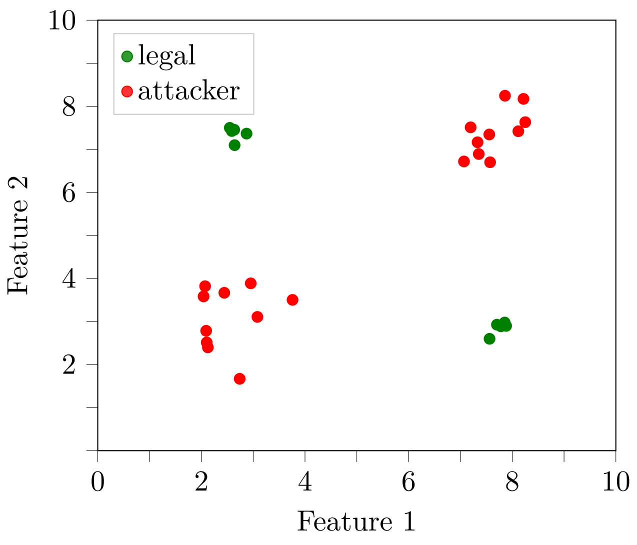


Figure 7.8: Effect of combining features

7.4.1 Login Activity

To complete the Login Activity, a user has to enter a four-digit PIN registered at the first start of the BehavioralCollector application. Thus, users enter the same digit sequence from memory on each run. To do that, a user uses our custom 11 keys numeric keyboard. The eleventh key is a backspace, which can remove the last written digit. Also, when a user types a wrong PIN, the input is automatically cleared, and the user starts again. Therefore, more than four digits could be written when a user completes the activity. We remove strokes representing deleted input, so only the four strokes that compose the correct PIN are considered.

From the data representing a stroke, only timestamps and x-coordinates are left, as the rest of a touch event's properties are not comparable between devices. Therefore, for the Login Activity features extraction, we denote the i-th stroke targeted to the i-th digit of the PIN as:

$$\mathbf{s_i} = [t_i^s, t_i^e, x_i],\ 1 \leq i \leq 4,$$

with a timestamp t_i^s of the stroke's start, timestamp t_i^e of the stroke's end, and x_i, the x-coordinate where the stroke started, hence where a user targeted the touch. We then define **duration** of a stroke $\mathbf{s_i}$ as:

$$d_i = t_i^e - t_i^s,\ 1 \leq i \leq 4,$$

and time between two consecutive strokes $\mathbf{s_j}$ and $\mathbf{s_{j+1}}$, also called **flight time**, as:

$$f_j = t_{j+1}^s - t_j^e,\ 1 \leq j \leq 3.$$

For **touch features**, we extract time features capturing how quickly a user is typing a PIN: $[d_1, d_2, d_3, d_4, f_1, f_2, f_3, \frac{1}{4}\sum_i^4 d_i, \frac{1}{3}\sum_j^3 f_j]$, where the last two features are interpretable as **mean touch duration**, and **mean flight time** respectively. Position touch features $[x_1, x_2, x_3, x_4]$ are added to test if users tend to choose a distinctive touch location in the x-axis.

Similarly **sensor features** also reflect the order of the strokes. Sensor features are subseries of an axis or magnitude of a sensor during or close to a stroke. Three different lengths of subseries are extracted according to the epsilon neighborhood strategy.

Feature combination is applied only to touch features, and all possible combinations of sizes 2 to 4 are considered.

7.4.2 Scrolling Activity

The Scrolling Activity is designed to capture a user scrolling through a vertical list. The activity is completed after a user finds and clicks on two negative values in the list. In this case, a user interacts with a touchscreen more extensively. Feature extraction from scrolls is the focus of the study [8], which proposes multiple scroll features also considered in this activity type.

For a stroke representing a scroll, not only the starting and ending touch events are taken into account, but also the touch events created by sliding on the touchscreen. For the Scrolling Activity, a scroll is defined as a sequence of n touch events e:

$$(e_n)_{n\in\mathbb{N}},\ \text{where } e_i = [x_i, y_i, t_i],$$

with x-coordinate x_i, y-coordinate y_i and timestamp t_i. First, we describe relevant features extractable from a scroll according to [8].

The device reacts independently from a stroke location, and the study claims that users tend to use distinctive screen areas. Because of the incomparability of the y-coordinates, only the x-coordinates $[x_1, x_n]$ of a scroll's endpoints are considered. The duration of a scroll d is another feature and is computed as:

$$d = t_n - t_1$$

We then compute pairwise velocity v_j between e_j and e_{j+1} as:

$$v_j = \frac{\sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2}}{t_{j+1} - t_j},\ 1 \leq j < n$$

and pairwise acceleration a_j between e_j and e_{j+1} as:

$$a_j = \frac{v_{j+1} - v_j}{t_{j+1} - t_j}, \ 1 \leq j < (n-1)$$

The median velocity and acceleration of the last five touch events of a scroll represent two features that can distinguish a user who stops a finger before lifting from a user who flicks. On the other hand, the list in the scrolling activity is relatively short, and the user has to concentrate on the values to find the negative ones, which could reduce the potential of these features.

The length of a scroll's trajectory l_{traj}, defined as:

$$l_{traj} = \sum_{j=1}^{n-1} \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2},$$

and distance of end-to-end line l_{ee}, defined by the first and last touch event, as:

$$l_{ee} = \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2},$$

are both considered to be scroll features. The ratio $\frac{l_{ee}}{l_{traj}}$ measures how much the scroll deviates from the straight line, and we also test its discriminatory potential.

Although a scroll is just a sequence of touch events from the data perspective, it could also be viewed as a group of vectors oriented in the scroll path direction, as depicted in Figure 7.9a, which brings additional processing options. Directions with unit magnitude between two consecutive touch events e_j and e_{j+1} are defined as:

$$z_j = exp(i\theta_j), \text{ with } \theta_j = atan2(y_{j+1} - y_j, \ x_{j+1} - x_j).$$

The mean resultant length given by:

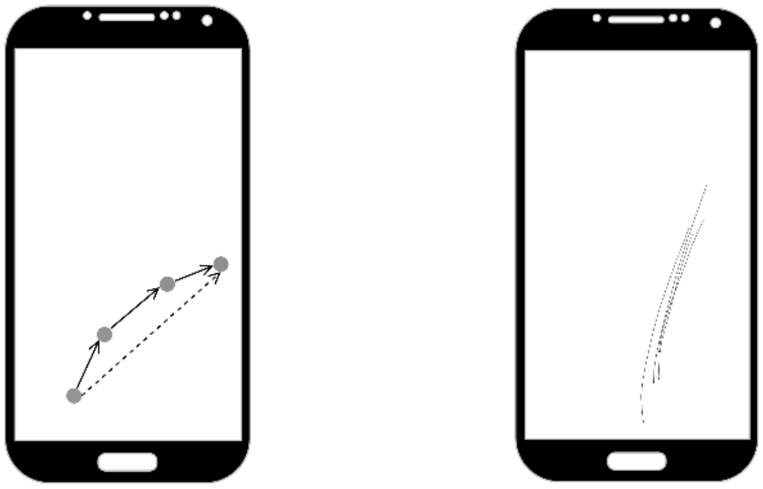
$$R = \frac{1}{n-1} \sum_{j=1}^{n-1} z_j,$$

quantify how directed the trajectory is with a scale between 1 for a straight line and 0 for uniformly random angles of line segments. End-to-end line direction is found as:

$$\overline{\theta} = atan2(y_n - y_1, \ x_n - x_1).$$

Both the R and $\overline{\theta}$ are considered to be scroll features.

To distinguish if the largest deviation is on the left side or the right side of the end-to-end line, which might indicate whether the user is left-handed or right-handed, the largest absolute perpendicular distance between the end-to-end connection and the trajectory is found. Perpendicular distances are found as a dot product of a vector forming the end-to-end line and perpendicular projections of vectors forming the trajectory of a scroll. The feature is represented by the largest absolute distance.



(a) Scroll as a set of vectors (b) Group of collected scrolls

Figure 7.9: Vector representation of scrolls (a) and collected group of scroll paths (b)

Finally, the 2D scrolling path normalized using standard scaling is extracted to be compared using the DTW algorithm.

At the beginning of the Scrolling Activity, the list is at its top, meaning a user has to make at least one scroll down by placing a finger on a touchscreen and moving it upwards. A user is not motivated by the activity design to make a scroll in the opposite direction. The data analysis also implies that users tend to choose a scrolling style that they continue to repeat for every other scroll in the activity, as visualized in Figure 7.9b. It could be beneficial to aggregate n longest scroll-down actions into an average scroll, representing the chosen style, which also could reduce the noise produced by relying on the stability of a single scroll. An average scroll is built as an element-wise average of a set of scrolls. To average time series, they are first interpolated to the same length, and then a point-by-point average is found. We decided to extract the features described above from the longest scroll down, an average scroll of two longest scrolls down, and an average scroll of three longest scrolls down.

A user has to click on two negative values in the list. Therefore, we also add durations and x-coordinates of strokes $\mathbf{s_1} = [t_1^s, t_1^e, x_1]$, $\mathbf{s_2} = [t_2^s, t_2^e, x_2]$ representing the two clicks to the feature set.

Sensor features are extracted during or close to a scroll action for each of three dimensions and magnitude and each sensor using the three subseries length strategy. Only the touch features are considered in the feature combination.

7.4.3 Typing Activity

To complete the typing activity, a user has to copy-type a randomly generated ten-digit number using a numeric keyboard with a backspace. If the number a user wrote does not match the generated one, a popup message shows up to notify about a mistake in the string, in which case the user has to find and correct the mistake using the backspace key.

Unlike the Login Activity, there is no repeating pattern in a user's action, and in this case, the order is irrelevant. Individual digits may be repeated in the generated number, so more complex preprocessing must be applied before the feature extraction step. For each key label $k \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, «\}$, we define a set of strokes $\mathcal{S}_k$, each containing n strokes $\mathbf{s} = [t^s, t^e, x],\ n \in \mathbb{N}^0$ representning clicks on the corresponding key.

Strokes with the same key label are grouped, enabling two aggregation types. Either only the first performed stroke from a set $\mathcal{S}_k$ is considered, or all the strokes are aggregated to an average stroke representing the set. For example, if a user types key *5* on the numerical keyboard twice, the average stroke duration of the key *5* is the average of durations of the two strokes. When averaging subseries, they are first interpolated to the same length, and then a point-by-point average is found.

For the feature extraction, the same operations are applied as for the strokes in the Login Activity type. For the **touch features**, x-coordinate and stroke duration are considered. A flight time for each key depends on the generated sequence of digits, so only the mean flight time is extracted. **Sensor features** are again represented by subseries during a stroke.

With 11 keys on the keyboard and taking into account the two strokes aggregation approaches, 1631 features will be considered in the selection process. However, due to the randomness of the Typing Activity, a significant part of the values is missing in the feature matrix, which compensates for the width of this matrix at the resource intensity. Because of the activity's randomness, no features are combined.

7.4.4 Clicking Activity

To complete the Clicking Activity, a user has to click on all five rectangles. The rectangles are located in the top left and top right screen's corners, middle of the screen, bottom left and bottom right screen's corners. The chosen clicking order does not affect the activity in any way, so it is up to a user's preference. This decision could hold some information about a user's behavior, but the activity aims to simulate general navigation through a mobile banking application, considering the order would be overly specific.

For the Clicking Activity, there are exactly five strokes, each representing a position of the targeted rectangle $\mathbf{s_i} = [t_i^s, t_i^e, x_i],\ i \in \{tl, tr, m, bl, br\}$. For the **touch features**, strokes durations and x-coordinates are considered. The flight time of a stroke is influenced by the clicking order, so only the mean flight time is included. Similarly, the **sensor features** are subseries associated with a rectangle position. On top of that, all combinations of the touch features of sizes 2 to 5 are considered.

7.5 Feature Selection

The feature extraction process output four large feature sets. However, only some features have sufficient discriminatory potential. The higher the discriminatory potential, the higher ability to discriminate between users. Due to the wide variety in the behavior of users and the number of feature columns without clear hints of which to exclude, an automated procedure that assigns a discriminatory potential score to each feature is designed. This procedure is based on the model's performance, hence the score is represented by AUC. A feature with a high discriminatory potential score close to 1.0 is perfectly stable for a single user across sessions and significantly differs from simulated attackers' sessions. A discriminatory potential score close to 0.5 does not include any information that would help to discriminate between users.

7.5.1 Procedure Design

The reasons for selecting a particular model have already been defended, and therefore it is also involved in the feature selection process. This process assigns a numerical score based on the model's performance to each feature, indicating its quality in terms of its ability to discriminate between different users. After that, specific feature selection for each activity type is made manually based on achieved scores.

A model's performance will be iteratively measured for each feature and each user individually. Different users behave differently and can therefore achieve significantly different discriminatory potential scores on each feature. When testing a model's performance for a user, all other users in the dataset simulate attackers.

Traditionally, a model's performance measure starts by splitting the dataset into two parts. The first part provides a historical context, and the second is used to estimate the model's performance. Unfortunately, our dataset is not large enough for this methodology. A strict static dataset split is too data wasteful in our case and might not give us relevant performance estimation as the testing phase would include a small number of samples. Cross-validation solves this problem by splitting the dataset into k-folds. Then, iteratively for each fold, $k-1$ of the folds are used as historical context, and the remaining is used for the performance evaluation. The result of the k-fold cross-validation is a vector of k performance measures. In our case, a vector of three AUC scores. Its mean is the discriminatory potential score of a feature for a specific user. For the feature selection performance testing, the LoOP's number of neighborhoods is set to three.

Our dataset size does not allow the creation of a validation set, so selecting a specific feature set for each user separately could lead to overfitting. For that reason, the final discriminatory potential score for a feature is a mean of feature scores across all users.

The related work studying performance metrics for behavioral authentication algorithms [56] points out that a mean score overlooks the security implications of different distributions. In our case, it means there is a difference between a mean resulting from uniformly distributed AUC values $[0.83, 0.83, 0.83]$ across three users and the same mean resulting from values $[0.99, 0.99, 0.50]$. While the former represents a random error, the latter shows a systematic error for the third device tested. The study suggests taking into account the Gini coefficient [57] together with the overall mean value. The Gini coefficient is a value in $[0, 1]$. The higher the value, the higher the inequity between individual scores.

7.5.2 Result

The previous section described a computational-intensive feature-scoring procedure that outputs the mean discriminatory potential score across all users and the corresponding Gini coefficient for each feature. Closer examination revealed no significant differences in the Gini coefficients, therefore, for selecting features for each activity type, only the mean AUC score is considered.

Selecting a feature set is done manually for each activity type. First, the features are sorted in descending order according to their discriminatory potential score, and the best-scored feature is used to build a baseline model. Then, we add another feature, and if that results in a better model's performance, the feature is added to the selected feature set; otherwise, it is discarded. This is incrementally done with features with discriminatory potential scores greater than 0.65. This iterative process excludes features already included in some combination.

For clarity, the selected features for each activity are commented on without stating the resulting score of each feature.

Login Activity

Although participants could freely choose their personal PIN, half of them chose sequence *1-2-3-4*. Only participants that used this PIN are included in the feature scoring process. Including also participants who used different PINs could lead to overestimated discriminatory potential scores, as that could reflect different typing sequences instead of just a user's behavioral characteristics. However, all users will be included in the final evaluation.

The best-performing feature for Login Activity turned out to be a combination of first flight time, second tap duration, third tap duration, and last flight time measured using Cosine distance. The users repeatedly type the same sequence from memory, and therefore, the speed of tapping on the keyboard plays a crucial role when analyzing users' behavior. Better performance was achieved using Cosine distance, which means the ratio of the times is more relevant than absolute values. Sensor features include the gyroscope, gravity, and accelerometer series during the individual strokes with the computed time epsilon neighborhood.

Scrolling Activity

For the Scrolling Activity, the two-dimensional DTW comparison of the normalized scroll path of the mean of the two longest scrolls performed the best. Then other features related to the mean scroll's trajectory style are used – the ratio between trajectory and end-to-end distance length, the length in the y-axis, scroll start position based on the x-coordinate, mean resultant length, and direction. From the sensor features, an accelerometer's z-axis series turned out to be relevant during the click on the first negative value, and a gyroscope's y-axis series during the mean scroll.

Typing Activity

Typing Activity features achieved a significantly lower score compared to other activities. The activity has no repeated action, and the limited dataset size does not give much opportunity for closer pattern matching. Nevertheless, mean flight time and mean tap duration performed the best. Accelerometer and gyroscope time series features are used during tapping keys *6, 8* and *9*. Because the keyboard is situated in the lower part of the screen, reaching most keys is comfortable. However, when holding the phone in the right hand and typing with the thumb, numbers *6, 8, 9* are the most difficult to reach. Moving the keyboards on top of a touchscreen could be beneficial to capture more specific behavioral biometrics.

Clicking Activity

The following features are selected for the Clicking Activity. Gyroscope's y-axis series for the middle, top left, and bottom right rectangle click, an accelerometer's z-axis series during the top left rectangle click, and a combination of the top left stroke duration, top right stroke duration and mean duration stroke measured using Manhattan distance.

The best-performing feature turned out to be a gyroscope's y-axis series for the top left rectangle click with precomputed time epsilon neighborhood. The reasoning is that the top left rectangle is the most difficult to reach, which leads a user to perform a characteristic motion captured by a gyroscope's y-axis.

Sanity Check

A model's performance should improve with the increasing number of sessions used as a historical context. As a sanity check, each model's performance is evaluated using the selected features, but now the available historical context is increased incrementally. If the model performance does not improve with the growing historical context, the captured features could reflect the characteristics of a device instead of a user's behavioral characteristics. This quick test shows an improvement tendency in each model's performance, so each activity's selected feature set passed the sanity check.

7.6 Combination Technique Selection

The selected modeling technique evaluates each activity separately. Therefore, when a user performs multiple activities in a BehavioralCollector's session, it results in a vector of authentication scores for individual activities. This section compares the performance of each score combination strategy defined in Section 7.1.3.

We start with the performance estimation of each activity type's model. This is done using Leave One Out Cross Validation, which can be seen as a special case of k-fold cross-validation with the k set to the number of samples in the dataset. As a result of this procedure, each sample in a feature matrix has the authentication score assigned by the LoOP based on all the legal user's available context, except the sample being currently tested. The result is then used to evaluate each activity type's model performance and test each of the proposed authentication score combination strategies.

The section describing different authentication scores combination strategies suggests three tests – selecting the best-performing model's output as the overall session authentication score, applying arithmetic mean, and using Bayes' updating. Figure 7.10 visualizes the tests' results.

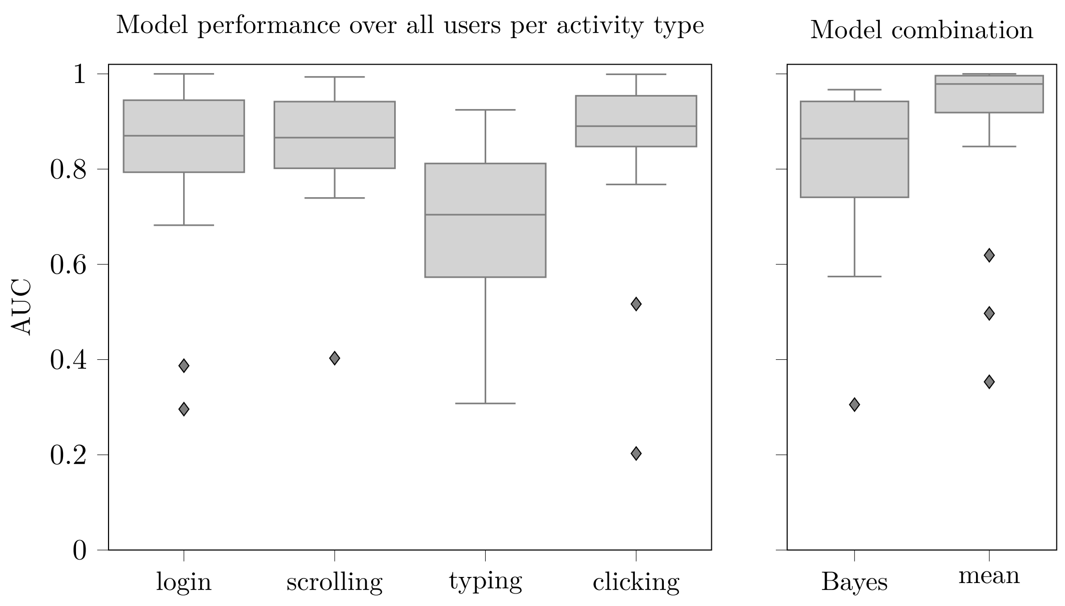


Figure 7.10: Individual and combined models performance estimation

The model for Clicking Activity has the highest median performance and is therefore considered the best-performing model. Nevertheless, it does not exceed the performance of the arithmetic mean approach, thus selecting the session's overall score based on the best-performing model is not a wise decision. Unexpectedly, the arithmetic mean approach outperforms the Bayes' updating. The Bayes' rule expects the pieces of evidence to be conditionally independent of one another [47]. Every activity type's model partly relies on features extracted from motion sensors, which likely hold similar information across activities. Thus the Bayes' updating overestimates the combined score. On the other hand, the main advantage of the arithmetic mean is to reduce noise in the final decision, which is apparently an important effect of the score combination.

7.7 Model Calibration

Two attributes parametrize the LoOP algorithm. The number of neighbors parameter sets how many local neighbors are included in the local density estimation, which influences the density estimation accuracy, hence the model's accuracy.

Extent parameter λ also modifies the density estimation. It increases the outlier-ness score of samples that deviate more than a given λ-times the standard deviation.

7.7.1 Number of neighbors

There are only three participants that contributed with more than ten sessions. Therefore, the test of the appropriate number of neighbors parameter is based on just these three users. The values $3, 5, 7$, and 10 were tested and re-evaluated using the Leave One Out method. The highest average AUC was achieved with *n_neighbors* set to 5.

7.7.2 Extent parameter

A model that scores sessions without any evidence should produce an indecisive score of 0.5. On the other hand, a decision that considers new information should move further from this value and closer to the truth, i.e., 0.0 or 1.0. The more decisive the model is, the more precise results are required. For appropriate calibration, the average score of a legitimate user's and attackers' sessions should be close to 0.5. For example, if the average authentication score of a legitimate user's sessions is 0.1, while for attackers' sessions is 0.4, it is not a well-calibrated decision. Instead, it is desirable to shift the authentication scores to 0.35 : 0.65. This can be achieved by scaling the model response with some value or calibrating the model.

The dataset needs more data to find a suitable value for the scaling approach. Therefore, only the extent value in the LoOP implementation will be adjusted. We repeated the performance estimation test from the beginning of this section using all possible extent parameter settings. Table 7.2 shows the average authentication scores assigned to legal and attack sessions.

Extent	Legal sessions scores and std	Attack sessions scores and std
3	0.17 (0.04)	0.40 (0.09)
2	0.25 (0.05)	0.55 (0.12)
1	0.41 (0.09)	0.76 (0.14)

Table 7.2: Mean authentication score and standard deviation of legal and attack sessions

By default, the extent is set to 3, meaning a sample is penalized if it deviates more than three times the standard deviation from the neighbors. This setting gives an uncalibrated authentication score. Regardless of whether an attacker or a legal user performs them, most sessions are scored below 0.5. Setting the extent parameter to 1 shifts the scores and increases the standard deviation. However, some legal sessions are scored close to 0.5, which is not the right calibration. Therefore we set the extent parameter to 2, as that represents the most acceptable setting.

7.8 Summary

This chapter described the process used to find methods to exploit behavioral biometrics from our private dataset for behavioral authentication. From the dataset, we used behavioral biometrics of 22 volunteers that repeatedly completed activities in BehavioralCollector and contributed with at least three sessions.

The dataset includes only legal users using their own devices. Thus, to evaluate a discriminatory potential of a feature and a statistical model's performance, we had to create a concept of a simulated attacker based on the data of other legal users.

Data collection was conducted in an uncontrolled environment on different types of smartphones, which complicated the analysis. Most properties of a touch event are values on a specific smartphone model's scale. Thus, these properties were excluded, as they would reflect a smartphone's characteristics instead of a user's behavioral characteristics. Sensor data are affected by different device calibration or sensor sampling rates. Therefore, the produced time series were interpolated to the same sampling rate and normalized so that absolute values do not directly affect the evaluation.

The data processing started with extracting feature sets from the dataset, one set for each type of simulated BehavioralCollector activity. The features extracted from touchscreen data directly reflect the features used in related work, representing characteristics of touchscreen interactions, such as key holding time, style of a scroll, etc. For the feature extraction from sensor data, we used a novel approach. We extracted time series of sensors data that were produced during a stroke. These series are compared to each other using a DTW algorithm that measures dissimilarity between two time series. That is a more robust method of exploiting micro-movements of the device captured by sensors during a user interaction than simple statistical functions used in related work.

However, not all extracted features have discriminative potential and thus informative value for behavioral authentication. We, therefore, created a procedure that assigns each feature a score quantifying its discriminative potential, based on which we selected specific features that are further used in the ScoringModule.

The behavioral authentication system aims to detect behavior that does not conform to the expected behavior of a user based on historical context. For statistical evaluation of behavioral biometrics, we, therefore, used an outlier detection method called Local Outlier Probabilities (LoOP), which assigns an outlier-ness score to the data. The assigned score can be, in our use case, interpreted as a probability that an activity is performed by an attacker.

The ScoringModule was built to receive and evaluate a single activity from BehavioralCollector. The relevant authority that decides whether to accept or reject a user needs to know the authentication score of an entire session. Thus, in this chapter, we have found a way to aggregate activity scores into a session authentication score. The most efficient method was found to be the arithmetic mean.

The session's authentication score calculated in this way assumes that the session is just as probably to be controlled by an attacker as it is to be controlled by a legitimate user. In a real-world scenario, however, the chance of an attack is significantly smaller. Therefore, we further modify the session's authentication score by including a qualified prior probability using the Bayes' rule. To demonstrate the authentication system, we keep the prior probability to 0.5.

To demonstrate the capability of the proposed methodology for behavioral authentication, we scored each session with an authentication score using the Leave One Out method. The scored sessions were then divided into two groups based on whether a legal user or a simulated attacker had performed them. The authentication scores achieved in both groups are visualized in Figure 7.11.

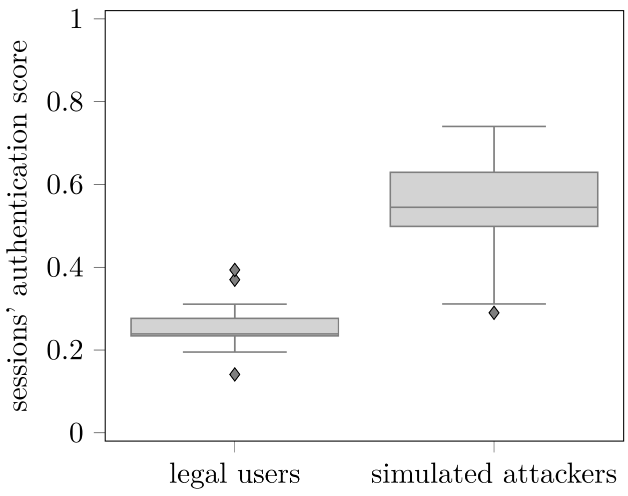


Figure 7.11: Sessions' authentication scores

Only a small overlap exists between the distribution of authentication scores assigned to sessions performed by legal users and attackers. This means it is possible to distinguish with a high probability between sessions performed by a legal user and sessions performed by an attacker using the authentication score.

It can be seen from the boxplot in Figure 7.11 that the full range of possible authentication scores $[0, 1]$ is not used. Although users differ in their behavior, they are not extremely different based on our proposed features. To have an accurate behavioral authentication system, users would have to make an identical motion, always at the same time. Their devices would have to produce sensor samples at a constant sampling rate and be equally accurate. However, these assumptions are not possible in the real world. Users behave differently over time, and their devices contain inaccurate commodity hardware. All of this contributes to noise in behavioral biometrics, which further results in less decisiveness of the statistical evaluation.

Table 7.3 shows estimated Equal Error Rate (EER) metrics for a clear comparison with other studies. However, it should be noted that in determining the EER metric, the threshold is set so that the False Acceptance Rate and False Rejection Rate are equal. However, in a real-world application, the threshold setting depends on the project in which the authentication system is used, based on the subjective cost of the two error types.

	median EER (std)
Login Activity	0.18 (0.17)
Scrolling Activity	0.19 (0.11)
Typing Activity	0.33 (0.16)
Clicking Activity	0.16 (0.17)
Session Score	**0.07** (0.15)

Table 7.3: Estimated EER for each activity type's model and best-performance combination technique for evaluation of a session's authentication score

Although the feature selection process for *Login Activity* included only participants with the PIN set to *1-2-3-4*. The performance evaluation of the whole dataset supports the right choice of features irrespective of the PIN used.

The Typing Activity's model performs significantly worse than others. As mentioned before, the activity does not force users to repeat an action. Instead, randomly generated digits are copied, so with the limited number of sessions in the dataset, our methods did not find strong features to discriminate between users.

This chapter described and tested different approaches to exploit behavioral biometrics collected using the BehavioralCollector application. The best-performing approach achieved an Equal Error Rate of 7 %, which is close to the performance of related work. While this might seem like a high error rate for a primary authentication system, a more pragmatic view is that it can detect different types of vulnerabilities that conventionally used authentication methods are not able to detect.

Despite the limited dataset, which was collected by a small group of volunteers for a short period of time, we were able to find discriminatory characteristics of users' behavior in the data that, together with the chosen modeling technique, could distinguish an attacker from a legitimate user based on the behavior in the BehavioralCollector application.

Chapter 8

Scoring Module

This chapter describes the last part of the work, which is an implementation of the ScoringModule and integration into the BAS by enabling communication with the BehavioralProcessor.

The data analysis process was done in a separate Python environment using Jupyter Notebooks. The first section, therefore, discusses the design decision of which technology to use for implementing the ScoringModule. Then the ScoringModule is implemented based on procedures proposed in Chapter 7 and integrated into the BAS. The ScoringModule has the following functionality:

1. The ScoringModule receives behavioral biometrics of a completed activity from the BehavioralProcessor.

2. It transforms the received behavioral biometrics into a feature vector according to the activity type based on the proposal of Chapter 7.

3. It loads the historical context of the corresponding device and transforms it into a feature matrix.

4. It matches the feature vector with the feature matrix using the modeling technique proposed in Chapter 7.

5. It returns the authentication score of the received behavioral biometrics back to the BehavioralProcessor.

8.1 Design Decision

While the BehavioralProcessor is implemented in Java, the data analysis process uses a Python environment. It is, therefore, necessary to consider how to implement the ScoringModule to enable communication with the BehavioralProcessor component. We have taken into account five different implementation methods.

The first method is to unify programming languages. Java and its packages, however, are not designed for intensive data operations. Although there are implementations of some of the used algorithms in Java, these are mostly untested standalone projects. Spring Framework supports database or security management extensions and is generally more robust than Python frameworks, such as Flask [58]. Thus converting the BehavioralProcessor to Python would be a step back in terms of production-ready software.

The second method is to use the implementation of Python in Java, called Jython [59]. Unfortunately, the Jython project does not support the packages used in the data analysis, such as Numpy or SciPy [60]. These are implemented in CPython only.

The third method is to write the authentication module as a Python script that the BehavioralProcessor application would execute as a subprocess. The communication between components would be through standard input and output. However, a basic wall-clock measuring shows that the initialization of the script takes more time than the data processing.

The fourth method uses Java Embedded Python [61], a project that embeds CPython in Java Native Interface. It provides a reusable Python interpreter directly in the Java runtime, significantly reducing the time overhead. A complication is that the interpreter is bound to a specific thread and cannot be shared. This implies that a thread pool would have to be maintained in the BehavioralProcessor to handle the interpreter instances. Otherwise, starting a new interpreter from each thread introduces a similar time overhead as executing a script.

The last considered method is building the ScoringModule as a web service communicating with the BehavioralProcessor via REST API. While this adds another web service that complicates deployment and maintenance, it also separates the individual roles of the authentication system and allows better scalability. Since the other proposed methods cause significant time overhead or unconventional implementation, we develop the ScoringModule as a separate web service.

8.2 Implementation

The ScoringModule is implemented in Flask [58], a web framework written in Python. To meet the requirements stated at the beginning of this section, the ScoringModule proceeds in steps visualized as an activity diagram in Figure 8.1, which will be further described.

The BehavioralProcessor processes behavioral biometrics from an input queue, in which the data are stored as a protocol-buffer message. This efficient serialization is also used when forwarding the data to the ScoringModule. Data were already validated at the BehavioralProcessor level. So, once the HTTP payload is deserialized into an internal structure holding behavioral biometrics of a single activity of the BehavioralCollector application, the evaluation pipeline starts. The first step is feature extraction.

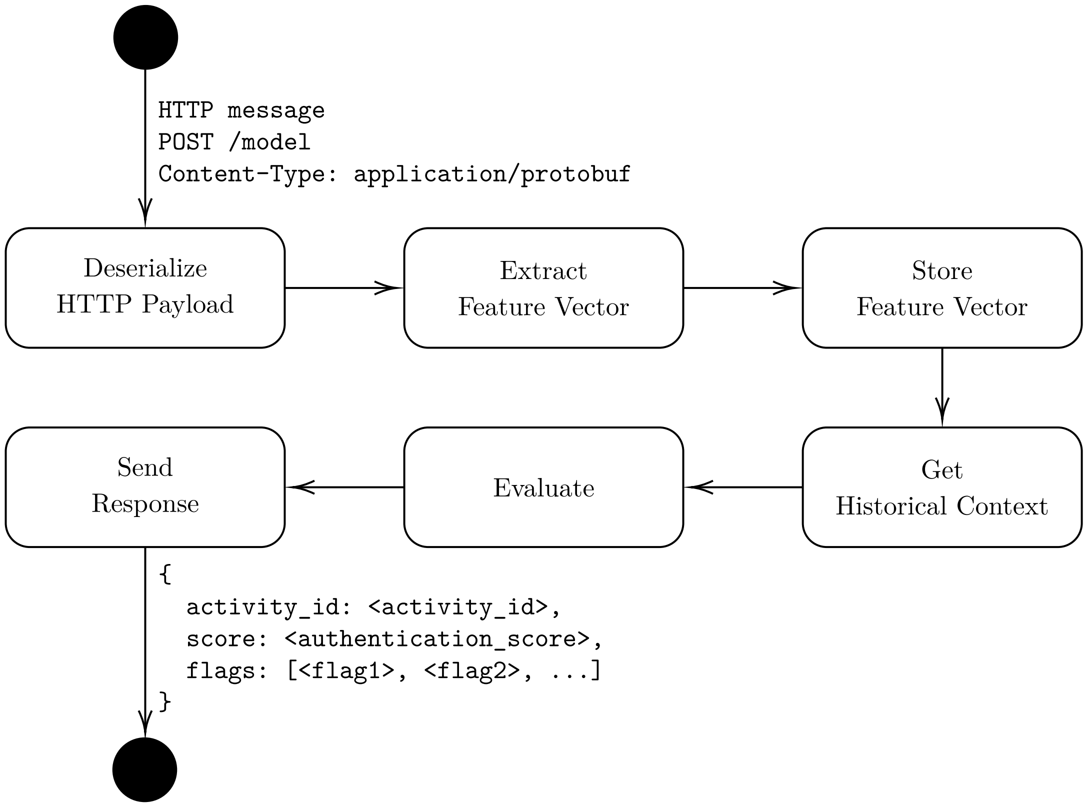


Figure 8.1: Activity diagram of the ScoringModule

8.2.1 Feature Extraction

The subset of behavioral biometrics used and the features included in the resulting feature vector depends on the activity type currently evaluated. This architectural design adheres to the strategy pattern [62], meaning there is an abstract class `Extractor` defining an `extract_vector()`, taking behavioral biometrics as the only argument. Each of its four subclasses implements the method according to the feature selection result of Chapter 7 process for the specific activity type. The output of the method is a feature vector.

8.2.2 Store Feature Vector

Feature extraction is a deterministic process. To optimize the ScoringModule's response time by avoiding repeated extraction, feature vectors are stored in a database called Feature Store Database (FSD). The pre-computed feature vectors can be later reused.

Flask's SQLAlchemy extension [63] is used for the relational database integration. Similarly to Spring's JPA, SQLAlchemy also creates tables according to entity class definitions. Schema of FSD is visualized in Appendix in Figure A.2. Each activity type has a dedicated table, where each row represents a feature vector. There is also a *devices_features* table that holds a device identification and references to its feature vectors for convenient Create, Read, Update, and Delete operations on a specific device's data. After this step, the previously extracted feature vector is stored in the FSD.

8.2.3 Get Historical Context

The currently processed behavioral biometrics of an activity from the BehavioralCollector needs a historical context to be matched during the evaluation step. The historical context includes all the past behavioral biometrics captured during the same activity type in sessions older than the current one produced by the same device.

The FSD contains already processed feature vectors, so the transformed historical context, referred to as a feature matrix, is obtained from that database. However, there is no synchronization mechanism between the FSD and the BBD managed by the BehavioralProcessor. If the BehavioralProcessor is unavailable, no data are received, hence no behavioral biometrics is passed to the ScoringModule. On the other hand, if the BehavioralProcessor is available, but the ScoringModule is unavailable, behavioral biometrics from smartphone devices are ingested and stored in the BBD but not passed to the ScoringModule, hence not transformed and stored in the FSD.

To overcome the inconsistency issue, the BBD is considered a source of truth. After a feature matrix is obtained from the FSD, a consistency check is done by querying the BBD. If an inconsistency is detected, missing historical context is transformed into feature vectors and stored in the FSD. At this step's end is a feature vector and feature matrix, both necessary for the evaluation step.

8.2.4 Evaluation

If the historical context's size is less than three, no evaluation is done. Otherwise, the feature vector with the feature matrix is evaluated using the modeling technique described in the previous chapter. The Local Outlier Probabilities (LoOP) algorithm is configured with the *extent* parameter set to 2, and the *n_neighbors* varies according to available historical context size C, so that

$$n\_neighbors = min(C,\ 5),$$

which follows the conclusion from Chapter 7. Therefore, the output of this step is an output of the LoOP algorithm, interpretable as the probability that the evaluated activity was performed by an attacker with respect to the historical context.

8.2.5 Response

Finally, a response for the BehavioralProcessor can be built. Its structure is visualized in Figure 8.1 as a JSON. It consists of the evaluated activity's identification that was included in the metadata of behavioral biometrics, the computed authentication score, and a set of flags that further describes the response of the ScoringModule. Table 8.1 shows possible flags and their interpretation.

Flag	Description
`EMPTY_CONTEXT`	There is no historical context for the device and activity type.
`SMALL_CONTEXT_SIZE`	Size of historical context is less than 10.
`REPEATED_PIN_INPUT`	User's PIN was not correct for first time during Login Activity.

Table 8.1: Possible response flags with description

8.3 Deployment

I used the Heroku development environment to deploy the ScoringModule. Heroku does not allow deploying two web applications on a single server, so the ScoringModule was deployed separately from the BehavioralProcessor. The PostgreSQL database instance is, however, shared between these two web services.

Before deploying a Flask application to production, the application should be wrapped into a web server software [64], we use Waitress [65] as it supports Windows, where the application is developed, and also Linux, where the application is deployed.

Deploying a Flask application using Heroku is not as convenient as deploying a Spring application. The deployment is done by building a docker image holding the ScoringModule that is pushed to the Heroku environment.

To integrate the ScoringModule into the behavioral authentication system, the BehavioralProcessor's processing service is modified so it forwards data to the ScoringModule via REST API as a protocol-buffer message in HTTP post payload. The Spring Framework provides a convenient way to perform REST API calls, called *RestTemplate*, which also handles the deserialization of a response and provides a mechanism for retrying a call if an error during the connection occurs.

8.4 Functionality Demonstration

To demonstrate the authentication system just by using a smartphone, the BehavioralCollector application is modified so it requests the authentication report once the sequence of activities is completed. The obtained authentication report is then displayed in the user interface.

We introduce one final functionality called an **impostor mode** for a convenient behavioral authentication system demonstration. The impostor mode is a mode that can be optionally turned on in the BehavioralCollector application using a switch button on the application start. When the impostor mode is active, the produced behavioral biometrics metadata contains a flag indicating the impostor mode is enabled. This causes behavioral biometrics not to be included later in the historical context. The authentication system processes behavioral biometrics in the same way as it would otherwise.

The purpose of the impostor mode is to allow a group of users to test the functionality of the behavioral authentication system. It is assumed that the owner of the smartphone device regularly uses the BehavioralCollector application and therefore has a broad historical context associated with his device. The device with the application running and active impostor mode can then be lent to someone else to simulate an attacker. The session of the simulated attacker will be evaluated but not included in the context of the legitimate user. This version of the BehavioralCollector is still publicly available on Google Play [5].

Similarly, as it was not possible to organize data collection in a controlled environment, it is now not possible to organize a test of the behavioral authentication system in which a group of volunteers simulates attackers on a specific physical smartphone device. However, my smartphone device was lent to another person to conduct a quick experiment. While the median authentication score of my sessions is 0.24, the session of the other person has an authentication score of 0.56. Although it is not a perfect test, the observed result matches expectations.

Conclusion

In this diploma thesis, a system for behavioral authentication was developed. This system verifies the identity of users based on their behavior in a smartphone application. Our system is only informative. It does not directly decide to reject or accept a user. Instead, it only informs the decision authority about a user's authentication score, interpretable as a probability that an attacker is performing a smartphone application session.

Our system is implemented as a web service that continuously receives and evaluates behavioral biometrics from a smartphone application. Behavioral biometrics consists of sensor and touchscreen data that are directly affected by a user's interaction with a smartphone application. The authentication process then evaluates a user's authenticity by trying to match the arriving behavioral biometrics to the previously captured behavioral biometrics of the user.

This diploma thesis focuses on mobile banking use, so we developed a client-side Android application called BehavioralCollector, which has four activities. The first activity simulates typing a PIN, the second simulates scrolling through a menu, the third simulates typing in an account number, and the last simulates navigating through an application by clicking on different positions on the touchscreen. The BehavioralCollector collects data from a touchscreen and six sensors in the background, including the accelerometer, gyroscope, linear acceleration sensor, gravity sensor, rotation vector sensor, and magnetometer. we initially shared this application among volunteers to collect a dataset of behavioral biometrics, and it is now available on Google Play [5] to demonstrate the functionality of the entire behavioral authentication system.

The data was analyzed to identify behavioral characteristics stable to each individual but different among multiple users. For data analysis, we collected a dataset comprising behavioral biometrics of 22 users. Although the dataset was small, we were able to identify a distinct set of features for each activity type that can between users. During simulated PIN typing, the speed at which the PIN is entered is a prominent characteristic. In simulated scrolling, an important feature is the shape of the trajectory of the scroll. When simulating typing an account number or navigating through an application, the micro-movements of the device caused by a user's touch on the touchscreen are the most distinguishing characteristics. Even though we only used data from our mobile banking simulator, it is reasonable to assume that the suggested features will also be effective in real-world applications.

Our system employs a sophisticated algorithm known as Local Outlier Probabilities (LoOP) to evaluate captured behavioral biometrics. This outlier detection algorithm assigns an outlier-ness score to the evaluated data, interpretable as a probability that a given sample is an outlier. In our use case, the outlier-ness score is interpreted as a probability that a given activity in a smartphone application is performed by an attacker. The LoOP algorithm has the advantage of considering the complex data structure resulting from unstable user behavior.

The LoOP algorithm evaluates each activity in a session separately. We determine a session's authentication score, which indicates the probability of an attacker performing the BehavioralCollector session, by calculating the average score of each activity in the session. This score combination approach performed better than other methods we tested.

Based on the data analysis findings, a separate module was created to evaluate behavioral biometrics. Our authentication system has two components, which are illustrated in Figure B.1 in Appendix. The BehavioralProcessor gathers behavioral biometrics from client-side applications, and ScoringModule evaluates these biometrics to assign an authentication score. The system stores both the authentication score and the behavioral biometrics in a relational database.

Our proposed behavioral authentication system achieved an Equal Error Rate of 7 % on the collected dataset. This outcome is close to other studies in this area. The behavioral authentication system is not meant to replace primary authentication methods like PIN or password. Instead, it serves as an additional security measure to detect potential attacks that may arise due to the vulnerabilities of using only one authentication mechanism. Behavioral authentication will not be as accurate as the widely used authentication methods. This is because widely used smartphones have only commodity hardware built-in, which introduces noise into behavioral biometrics. Additionally, even if users have flawless hardware, the unpredictable nature of human behavior can still present difficulties.

Related works in this area typically analyze data from a controlled environment and do not often consider real-world applications of their proposed system. The dataset used in this diploma thesis was constructed by volunteers using various devices without supervision, which resulted in inconsistencies in the dataset, making data analysis more challenging. Furthermore, this diploma thesis developed a functional behavioral authentication system that can improve the security of a smartphone application and protect confidential data.

Future Work

Each user in the dataset used only a single device, so we were unable to analyze any changes in behavioral biometrics that may occur when switching between different smartphone devices. Our system evaluates the authentication score by matching the set of behavioral biometrics produced by a specific device, regardless of who is using it. One way to improve the system is to gather data from users who use different devices and study how it affects their behavioral biometrics.

Related work has shown that users' behavior varies depending on their body position. Furthermore, it has been noted that behavior differs depending on whether a user is sitting, lying down, or walking. This relates to situations where the user is not holding a device but instead interacting with a device that is placed on a table. Although by employing LoOP, our system is prepared for complex data structure evaluation, we were unable to analyze these particular cases as our dataset was too small.

Bibliography

[1] Abdelrahman, Y.; Khamis, M.; et al. Stay Cool! Understanding Thermal Attacks on Mobile-based User Authentication. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, Association for Computing Machinery, ISBN 978-1-4503-4655-9, pp. 3751–3763, doi:10.1145/3025453.3025461. Available from: `https://dl.acm.org/doi/10.1145/3025453.3025461`

[2] Aviv, A.; Gibson, K.; et al. Smudge Attacks on Smartphone Touch Screens.

[3] Cai, L.; Chen, H. On the Practicality of Motion Based Keystroke Inference Attack. In *Trust and Trustworthy Computing*, volume 7344, edited by S. Katzenbeisser; E. Weippl; L. J. Camp; M. Volkamer; M. Reiter; X. Zhang, Springer Berlin Heidelberg, ISBN 978-3-642-30920-5 978-3-642-30921-2, pp. 273–290, doi:10.1007/978-3-642-30921-2_16. Available from: `http://link.springer.com/10.1007/978-3-642-30921-2_16`

[4] Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop. Available from: `https://cloak-and-dagger.org/`

[5] Pešek, J. Behavioralcollector - Android Apps on Google Play. Available from: `https://play.google.com/store/apps/details?id=com.wultra.behavioralcollector`

[6] Kuchyňová, K. Ověřování identity uživatele založené na behaviorálních charakteristikách.

[7] Shi, C.; Liu, J.; et al. WiFi-Enabled User Authentication through Deep Learning in Daily Activities. volume 2, no. 2: pp. 13:1–13:25, ISSN 2691-1914, doi:10.1145/3448738. Available from: `https://dl.acm.org/doi/10.1145/3448738`

[8] Frank, M.; Biedert, R.; et al. Touchalytics: On the Applicability of Touchscreen Input as a Behavioral Biometric for Continuous Authentication. volume 8, no. 1: pp. 136–148, ISSN 1556-6021, doi:10.1109/TIFS.2012.2225048.

[9] Park, J.; Kim, T.; et al. Touch Gesture Data Based Authentication Method for Smartphone Users. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, ACM, ISBN 978-1-4503-4455-5, pp. 136–141, doi:10.1145/2987386.2987410. Available from: `https://dl.acm.org/doi/10.1145/2987386.2987410`

[10] Zheng, N.; Bai, K.; et al. You Are How You Touch: User Verification on Smartphones via Tapping Behaviors. In *2014 IEEE 22nd International Conference on Network Protocols*, ISSN 1092-1648, pp. 221–232, doi:10.1109/ICNP.2014.43.

[11] Sitová, Z.; Šeděnka, J.; et al. HMOG: New Behavioral Biometric Features for Continuous Authentication of Smartphone Users. volume 11, no. 5: pp. 877–892, ISSN 1556-6021, doi:10.1109/TIFS.2015.2506542.

[12] DAKOTA: Sensor and Touch Screen-Based Continuous Authentication on a Mobile Banking Application — IEEE Journals & Magazine — IEEE Xplore. Available from: `https://ieeexplore.ieee.org/document/9367144`

[13] Estrela, P.; Albuquerque, R.; et al. A Framework for Continuous Authentication Based on Touch Dynamics Biometrics for Mobile Banking Applications. volume 21: p. 4212, doi:10.3390/s21124212.

[14] Shen, C.; Li, Y.; et al. Performance Analysis of Multi-Motion Sensor Behavior for Active Smartphone Authentication. volume 13, no. 1: pp. 48–62, ISSN 1556-6013, 1556-6021, doi:10.1109/TIFS.2017.2737969. Available from: `http://ieeexplore.ieee.org/document/8006292/`

[15] Papamichail, M. D.; Chatzidimitriou, K. C.; et al. BrainRun: A Behavioral Biometrics Dataset towards Continuous Implicit Authentication. volume 4, no. 2: p. 60, ISSN 2306-5729, doi:10.3390/data4020060. Available from: `https://www.mdpi.com/2306-5729/4/2/60`

[16] Sensors Overview — Android Developers. Available from: `https://developer.android.com/guide/topics/sensors/sensors_overview`

[17] Environment Sensors — Android Developers. Available from: `https://developer.android.com/guide/topics/sensors/sensors_environment`

[18] Sensor Types. Available from: `https://source.android.com/docs/core/interaction/sensors/sensor-types`

[19] Position Sensors. Available from: `https://developer.android.com/guide/topics/sensors/sensors_position`

[20] Motion Sensors — Android Developers. Available from: `https://developer.android.com/guide/topics/sensors/sensors_motion`

[21] Android CDD. Available from: https://source.android.com/static/docs/compatibility/android-cdd.pdf

[22] Batching. Available from: https://source.android.com/docs/core/interaction/sensors/batching

[23] Sensor Stack. Available from: https://source.android.com/docs/core/interaction/sensors/sensor-stack

[24] Touch Devices — Android Open Source Project. Available from: https://source.android.com/docs/core/interaction/input/touch-devices

[25] MotionEvent. Available from: https://developer.android.com/reference/android/view/MotionEvent

[26] Download Android Studio & App Tools. Available from: https://developer.android.com/studio

[27] The Activity Lifecycle. Available from: https://developer.android.com/guide/components/activities/activity-lifecycle

[28] Settings.Secure. Available from: https://developer.android.com/reference/android/provider/Settings.Secure

[29] Processes and Threads Overview. Available from: https://developer.android.com/guide/components/processes-and-threads

[30] HttpURLConnection — Android Developers. Available from: https://developer.android.com/reference/java/net/HttpURLConnection

[31] Why Spring. Available from: https://spring.io

[32] Redis. Available from: https://redis.io/

[33] Group, P. G. D. PostgreSQL. Available from: https://www.postgresql.org/

[34] Redis Message Broker — Redis Enterprise. Available from: https://redis.com/solutions/use-cases/messaging/

[35] Protocol Buffers. Available from: https://protobuf.dev/

[36] Beating JSON Performance with Protobuf. Available from: https://auth0.com/blog/beating-json-performance-with-protobuf/

[37] os72. Protoc-Jar. Available from: https://github.com/os72/protoc-jar

[38] Redis FAQ. Available from: https://redis.io/docs/getting-started/faq/

[39] Spring Batch - Reference Documentation. Available from: https://docs.spring.io/spring-batch/docs/current/reference/html/

[40] BLPOP. Available from: https://redis.io/commands/blpop/

[41] How Heroku Works — Heroku Dev Center. Available from: https://devcenter.heroku.com/articles/how-heroku-works

[42] 2.7. Novelty and Outlier Detection. Available from: https://scikit-learn/stable/modules/outlier_detection.html

[43] Breunig, M. M.; Kriegel, H.-P.; et al. LOF: Identifying Density-Based Local Outliers. volume 29, no. 2: pp. 93–104, ISSN 0163-5808, doi:10.1145/335191.335388. Available from: https://dl.acm.org/doi/10.1145/335191.335388

[44] Kriegel, H.-P.; Kröger, P.; et al. LoOP: Local Outlier Probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, Association for Computing Machinery, ISBN 978-1-60558-512-3, pp. 1649–1652, doi:10.1145/1645953.1646195. Available from: https://doi.org/10.1145/1645953.1646195

[45] PyNomaly/Test_loop.Py at Main · Vc1492a/PyNomaly. Available from: https://github.com/vc1492a/PyNomaly

[46] Bates, J. M.; Granger, C. W. J. The Combination of Forecasts. volume 20, no. 4: pp. 451–468, ISSN 1473-2858, doi:10.2307/3008764, 3008764. Available from: https://www.jstor.org/stable/3008764

[47] Bayes' Rule: Guide. Available from: https://arbital.com/p/bayes_rule_guide/

[48] Classification: ROC Curve and AUC — Machine Learning. Available from: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[49] NumPy. Available from: https://numpy.org/

[50] Pandas - Python Data Analysis Library. Available from: https://pandas.pydata.org/

[51] SQLAlchemy - The Database Toolkit for Python. Available from: https://www.sqlalchemy.org/

[52] Pešek, J. NomadTask Quest 17574. Available from: https://nomadtask.com/profile/maker/quest/17574

[53] Measure Linear Acceleration along X, Y, and Z Axes in m/S2 - Simulink. Available from: https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html

[54] Measure Rotational Speed around X, Y, and Z Axes in Rad/s - Simulink. Available from: https://www.mathworks.com/help/supportpkg/android/ref/gyroscope.html

[55] Sensors Overview. Available from: `https://developer.android.com/guide/topics/sensors/sensors_overview`

[56] Eberz, S.; Rasmussen, K. B.; et al. Evaluating Behavioral Biometrics for Continuous Authentication: Challenges and Metrics. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, Association for Computing Machinery, ISBN 978-1-4503-4944-4, pp. 386–399, doi:10.1145/3052973.3053032. Available from: `https://doi.org/10.1145/3052973.3053032`

[57] Gini Coefficient. Available from: `https://en.wikipedia.org/w/index.php?title=Gini_coefficient&oldid=1151310873`

[58] Welcome to Flask — Flask Documentation (2.2.x). Available from: `https://flask.palletsprojects.com/en/2.2.x/`

[59] Jython. Available from: `https://www.jython.org/`

[60] SciPy - Frequently Asked Questions. Available from: `https://scipy.org/faq/#does-scipy-work-with-jython-or-cnet`

[61] Jep/Src/Main/Java/Jep at Master · Ninia/Jep. Available from: `https://github.com/ninia/jep`

[62] Strategy. Available from: `https://refactoring.guru/design-patterns/strategy`

[63] Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (3.0.x). Available from: `https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/`

[64] Deploying to Production — Flask Documentation (2.2.x). Available from: `https://flask.palletsprojects.com/en/2.2.x/deploying/`

[65] Waitress — Flask Documentation (2.2.x). Available from: `https://flask.palletsprojects.com/en/2.2.x/deploying/waitress/`

Appendix **A**

Database Schemata

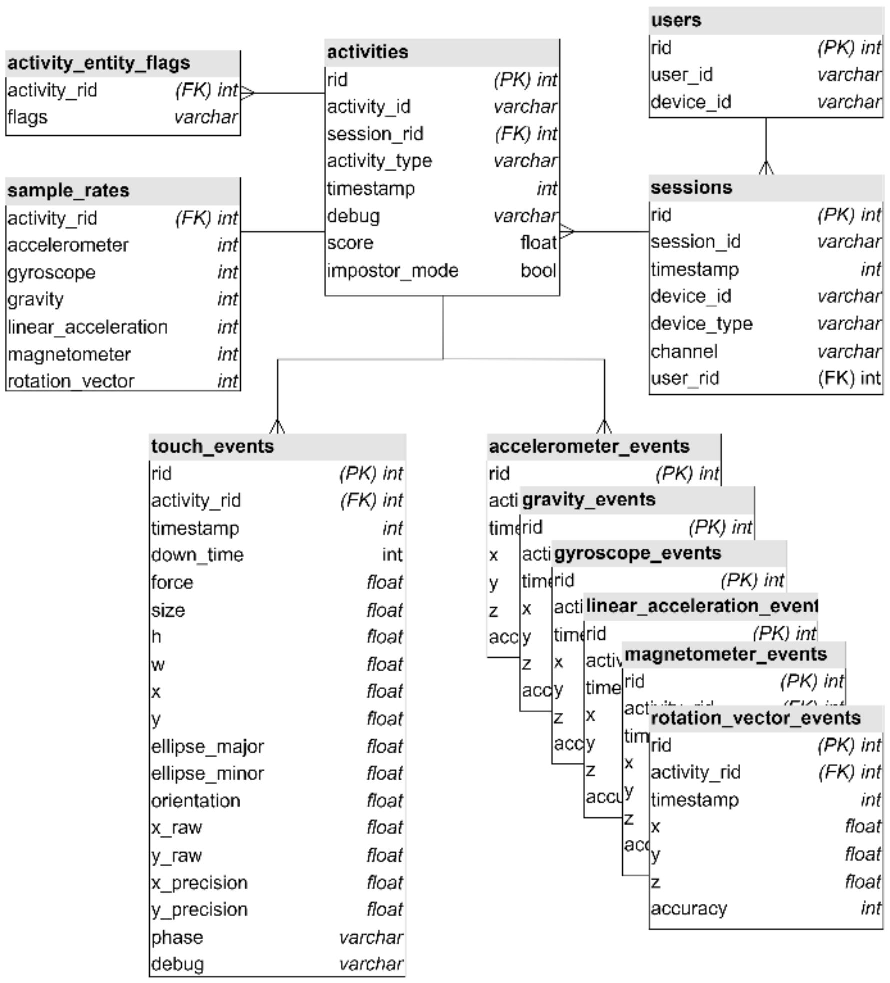


Figure A.1: Behavioral biometrics database schema

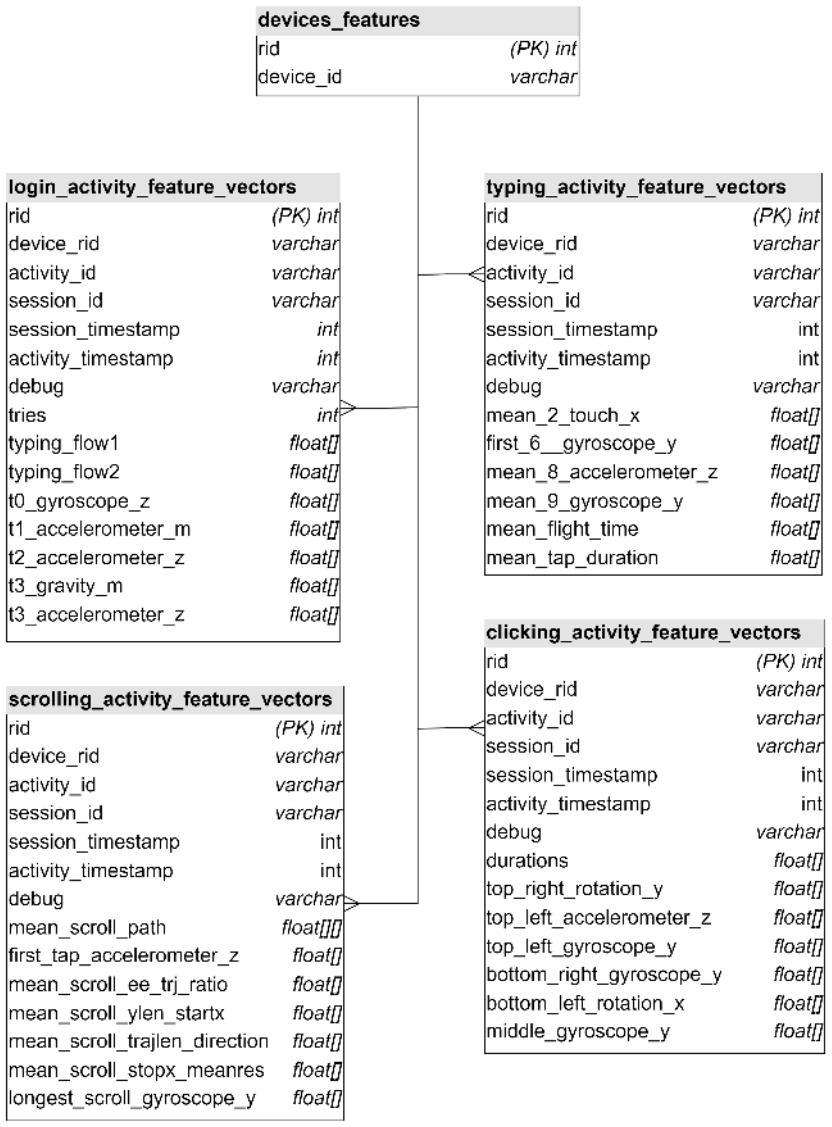
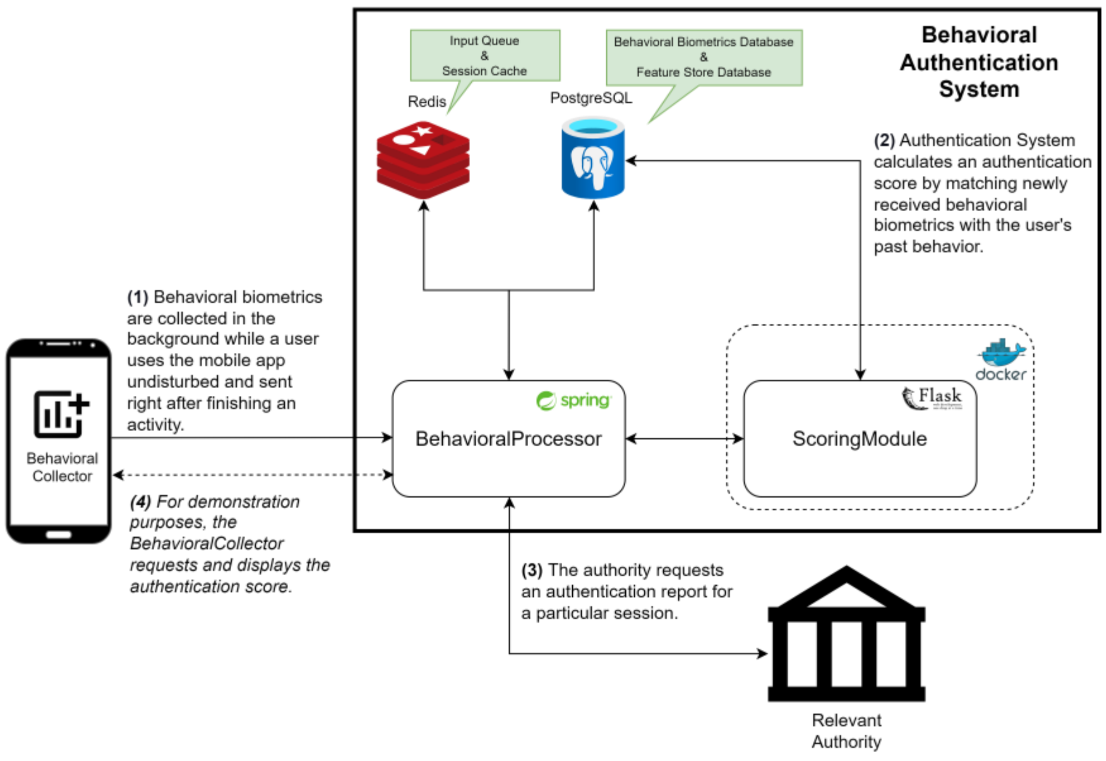

devices_features
rid (PK) int
device_id varchar
login_activity_feature_vectors
rid (PK) int
device_rid varchar
activity_id varchar
session_id varchar
session_timestamp int
activity_timestamp int
debug varchar
tries int
typing_flow1 float[]
typing_flow2 float[]
t0_gyroscope_z float[]
t1_accelerometer_m float[]
t2_accelerometer_z float[]
t3_gravity_m float[]
t3_accelerometer_z float[]
typing_activity_feature_vectors
rid (PK) int
device_rid varchar
activity_id varchar
session_id varchar
session_timestamp int
activity_timestamp int
debug varchar
mean_2_touch_x float[]
first_6__gyroscope_y float[]
mean_8_accelerometer_z float[]
mean_9_gyroscope_y float[]
mean_flight_time float[]
mean_tap_duration float[]
scrolling_activity_feature_vectors
rid (PK) int
device_rid varchar
activity_id varchar
session_id varchar
session_timestamp int
activity_timestamp int
debug varchar
mean_scroll_path float[][]
first_tap_accelerometer_z float[]
mean_scroll_ee_trj_ratio float[]
mean_scroll_ylen_startx float[]
mean_scroll_trajlen_direction float[]
mean_scroll_stopx_meanres float[]
longest_scroll_gyroscope_y float[]
clicking_activity_feature_vectors
rid (PK) int
device_rid varchar
activity_id varchar
session_id varchar
session_timestamp int
activity_timestamp int
debug varchar
durations float[]
top_right_rotation_y float[]
top_left_accelerometer_z float[]
top_left_gyroscope_y float[]
bottom_right_gyroscope_y float[]
bottom_left_rotation_x float[]
middle_gyroscope_y float[]

Figure A.2: Feature Store database schema

Appendix B

Detailed Architecture



Behavioral Authentication System
Input Queue & Session Cache
Redis
Behavioral Biometrics Database & Feature Store Database
PostgreSQL
(2) Authentication System calculates an authentication score by matching newly received behavioral biometrics with the user's past behavior.
(1) Behavioral biometrics are collected in the background while a user uses the mobile app undisturbed and sent right after finishing an activity.
Behavioral Collector
spring
BehavioralProcessor
docker
Flask
ScoringModule
(4) For demonstration purposes, the BehavioralCollector requests and displays the authentication score.
(3) The authority requests an authentication report for a particular session.
Relevant Authority

Figure B.1: Detailed architecture and communication diagram