JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Philipp Kapfer

Submitted at
**Institute of
Networks and Security**

Supervisor
René Mayrhofer

Adviser
Josef Scharinger

August 2016

PhonyKeyboard: Sensor-enhanced Keystroke Dynamics Authentication on Mobile Devices

Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Good things come to those who wait.

— Nayobe, 1984

ABSTRACT

In this Master Thesis, we propose a novel sensor-enhanced keystroke dynamics authentication system for mobile devices. In Chapter 1, we describe biometric systems and how soft biometrics can be used to authenticate users. A comparison of biometric features sets the frame for our approach. In Chapter 2, we describe how keystroke dynamics can enhance password authentication and what types of sensors in mobile devices complement them best. In Chapter 3 we summarize related approaches and describe the lessons we learned.

Part two documents our authentication system called *PhonyKeyboard*. In Chapter 4, we describe the theoretical concepts of our approach independent of implementation decisions. Chapter 5 documents the implementation and how we added biometrics to the *Google Keyboard*. We also introduce an example client. Furthermore, we integrated it into the CORMORANT framework. In Chapter 6 we evaluate the system with real-life data and conclude that sensor-enhanced authentication reaches an EER of 10.44 %, although not using sensors only increases it by 0.44 %. Chapter 7 summarizes the thesis and gives an outlook.

ZUSAMMENFASSUNG

In dieser Masterarbeit schlagen wir eine neuartige Tastaturanschlagdynamik-Authentifizierung vor, die durch Sensordaten verbessert wird. In Kapitel 1 beschreiben wir Biometriesysteme und wie schwache Biometrie Benutzer authentifizieren kann. Ein Vergleich biometrischer Merkmale bildet den Rahmen für unseren Ansatz. In Kapitel 2 beschreiben wir, wie Tastaturanschlagdynamik die Passwortauthentifizierung verbessert und welche Arten von Sensoren in Mobilgeräten sie am besten ergänzen. In Kapitel 3 fassen wir verwandte Ansätze zusammen und beschreiben, was wir daraus lernten.

Teil zwei dokumentiert unser Authentifizierungssystem namens *PhonyKeyboard*. In Kapitel 4 beschreiben wir die theoretischen Konzepte unseres Ansatzes unabhängig von Implementierungsentscheidungen. Kapitel 5 dokumentiert die Implementierung und wie wir Biometrie zur *Google-Tastatur* hinzugefügt haben. Wir stellen auch einen Beispielclient vor. Zudem integrierten wir die App in das CORMORANT-Framework. In Kapitel 6 evaluieren wir das System mit Echtdaten und schlussfolgern, dass durch Sensordaten verbesserte Authentifizierung eine EER von 10,44 % erreicht, wohingegen keine Sensoren zu benutzen sie nur um 0,44 % erhöht. Kapitel 7 fasst die Arbeit zusammen und gibt einen Ausblick.

CONTENTS

Part I

THE SCIENCE OF KEYSTROKE DYNAMICS

1 INTRODUCTION

Over the time, biometrics evolved from a special application for high security areas to a technology that is available for the masses and used by everyone. Some years ago, fingerprint readers were only built-in to business laptops, while today they are integrated also into consumer models. Just in the last few years, manufacturers started including fingerprint sensors into mainstream mobile devices. Other options, like face or voice recognition, are also available on smart phones. However, although these technologies enhance security quite well, they are often cumbersome to use: while the built-in camera is still trying to recognize a user's face and performing a liveliness check, they have already entered a Personal Identification Number (PIN) and started to use the phone. If one just wants to check new mail or look something up, these small delays interrupt the workflow and disturb the user. In the end, biometric authentication will be disabled and the phone is susceptible to shoulder surfing and unauthorized access again.

But there is a solution to this problem. A whole group of biometrics is mostly unknown to the public: *Soft biometrics* (also called *behavioral cues* or *implicit authentication* [Sto11]) cannot determine who a person is (identification), but merely who a person is *not* (authentication). Nevertheless, it has its applications as an addition and replacement for hard biometrics. The huge advantage of these authentication methods is that they neither need any action nor cooperation from the user. They do their work in the background – unnoticed, without the need of any additional equipment – but nonetheless enhance security of the whole system. These biometric methods don't require advanced processing (as e. g. fingerprint readers need for extracting minutiae from images), they are low-cost and easy to implement [Tas+14]. The promise is enhanced security without impacting usability. In a survey, 60 % of the respondents wished this very thing: easier forms of mobile authentication [Ali+12]. In this chapter, we talk about what biometrics is, motivate to use keystroke dynamics as biometric features and state the scope and goal of this thesis.

For more information about the general topic of soft biometrics, the interested reader may consult [Kap14].

1.1 BIOMETRICS

The word "biometrics" originates from the two Greek terms *bios* (life) and *metricos* (measure) [Amb05]. Biometric systems are pattern recognition systems. They recognize people by their physiological and behavioral characteristics. The foundation therefore is a human fea-

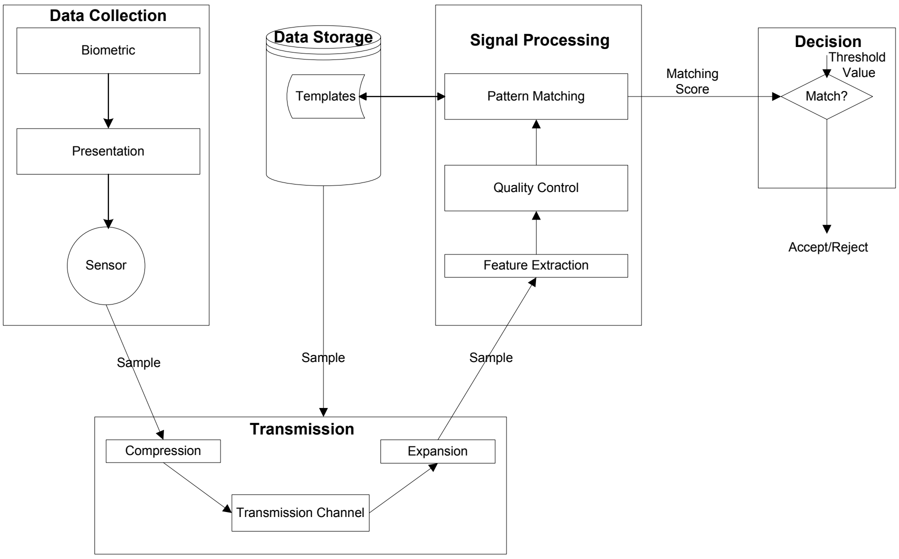


Figure 1: Detailed data flow of a biometric system with the four main modules: data collection, signal processing, decision and data storage [Amb05]

ture that fulfills certain properties for secure *identification* (searching persons with matching features in a database – 1 : n) or *verification* (matching a certain person to given features – 1 : 1). Biometrics also is one of the few technologies that allows *negative recognition*, where it recognizes if a person is someone who they deny being.

1.1.1 *Components*

Every biometric system follows a certain basic structure independent of chosen features and security. Figure 1 shows the way of data through an exemplary system. Its base are four modules [Amb05]: The *Data Collection* unit captures one instance (a sample) of the real-world representation of a biometric feature through some kind of sensor. *Signal Processing* is the most important part of the system: the sample data is separated from noise and the information that is relevant for comparing this specific biometric is extracted (*Feature Extraction*). Through *Pattern Matching*, the current sample data is compared with previously enrolled templates and a score for how well they match is calculated. The *Decision* unit then decides upon this value whether the sample belongs to a valid user or an impostor is trying to access the system. In consequence, the request is accepted or rejected. Finally, in *Data Storage*, the information most significant for a specific user is stored securely in form of templates. These templates are relatively simple representations of the data that was originally captured by the sensor, but they contain the most important bits for authentication.

1.1.2 *Soft Biometrics*

As mentioned in the introduction, strong biometrics such as fingerprints or the iris are broadly adopted in technical systems. Nevertheless, they are still actively developed further, because the methods are good but not perfect. Biometric verification only works when two significant criteria are met: the captured features are gathered from a real person at the time of verification and they match with stored templates. Various circumstances have influence on these criteria, such as noisy data (e. g. through broken sensors or swollen fingers), unfavorable environments or non-universal features (ones that not every person has). Sensors can also be tricked, for example by using portrait photos for facial recognition or silicone molds of fingerprints.

The NIST found that no data of acceptable quality can be extracted from the fingerprints of 2 % of the US-American population [JDN04]

To mitigate these issues, it is increasingly common to add "soft" features that are gathered along the way when capturing the actual biometric data, such as body height, gait, tattoos, etc. In authentication, this data can be used [Amb05]

- to filter the template database and remove irrelevant ones when matching the strong biometrics,
- to combine both types of features, which improves recognition performance and security or
- stand-alone in situations where otherwise no biometric security is present at all.

The most important property of soft features is that they can be used for *implicit* authentication, which means that the user doesn't have to actively do anything in the process. It takes place while they perform their daily activities. Strong biometrics, in contrast, only perform *explicit* authentication, where users have to interrupt their work flow to execute the data collection task before they can proceed.

1.1.3 *Mobile Devices*

With the increasing popularity of smart phones and tablets, many researchers asked whether the proven biometric systems can also be used in mobile environments. These devices create unique challenges and opportunities and require new approaches to the same old problems. The most important factors are limited resources (e. g. Central Processing Unit (CPU) power and memory size) and a limited sensor set. Detailed considerations regarding resources are explained in Section 4.3 (Statistical Classification). As to sensors, in contrast to dedicated or computer-based biometric authentication systems, special equipment such as a fingerprint reader cannot be simply attached to a mobile device. Therefore, implementers have to work with the

FEATURE	CLASSIFICATION	EER [%]	IP	TYPE
Fingerprint [AB14]	Vaulted Fingerprint Verification	7.5	yes	explicit
Hand Gesture [JOT12]	Dynamic Time Warping	0.87	no	explicit
Keystroke Dynamics [ZDJ12]	New Distance Metric[a] + outlier removal	0.84	no	implicit
Signature [Ira+14]	Neural Network	6.9	yes	explicit / implicit[b]
Voice [BCR12]	Vector Quantization	0.83	no	explicit[c]
Face [Bor+11]	Fusion of Beta Wavelet, Facial Curves Shaped Analysis and Iterative Closest Point	1.6	yes	explicit[d]
Iris [LLC14][e]	matching reconstructed signal	FAR: 0.01 %, FRR: 0.69 %	?	?

[a] Distances between samples are calculated by combining Manhattan and Mahalanobis distance.

[b] The type depends on the situation, whether a signature is requested just for authentication purposes or it is required anyway, e. g. for signing a contract.

[c] Voice recognition in general can also be implicit, but the cited paper uses a challenge-based approach.

[d] The authors used a dataset with explicitly captured images, although facial recognition can also be performed in the background during everyday usage.

[e] Unfortunately, we had no access to the paper's fulltext. The given properties are cited from [AAM15].

Table 1: The most common biometric features available on mobile devices with selected approaches and their performance [AAM15]. IP = Impostor Patterns

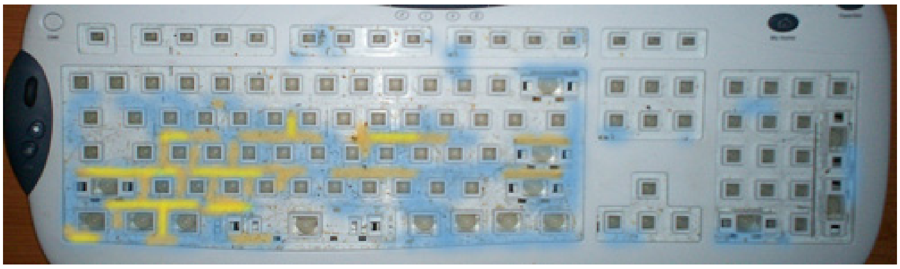


Figure 2: Heat map of the amount of dirt on an opened keyboard. Yellow areas represent peak dirt locations [Taro7]

hardware that is already available. Then again, several devices, for example microphones, cameras and accelerometers, can be expected in nearly every smart phone. Table 1 summarizes the most common biometrics that can be used on mobile devices, selected classification algorithms and the Equal Error Rates (EERs) that can be achieved when using them. They were not tested on mobile devices, however. It's also stated whether the selected method needs impostor patterns (titled "IP"; examples of templates for users that mustn't be granted access in addition to templates for the valid user) and if it is an implicit or an explicit authentication approach.

1.2 MOTIVATION

The way someone types on a keyboard, for example when entering one's password, is one kind of soft biometric feature that allows detecting whether the person sitting in front of a computer is a legitimate user or an impostor. One just has to look at one's own keyboard, which probably wasn't cleaned for an extended period. The residue that accumulates on the surface of each key (and over the time, the plastic of the key caps themselves) has certain shiny spots, where fingers often hit the key, while other keys don't. Figure 2 shows the heat map of dirt distribution Targonski [Taro7] painted while cleaning it. He also noticed that dirt between the keys is more present around often-used keys. It's also visible that e. g. the *F5* function key is more heavily used than the other ones, which are almost residue-free.

With the advent of mobile phones, the concept of keyboard authentication was ported to a mobile environment, beginning with the physical keypad text inputs and advancing to touch inputs on smart phones in recent years. These new input methods, however, provide far more data about a key press than mere timing between the keystrokes. Touch screens are able to detect the angle, size and pressure of a finger, even multiple ones at the same time. Also, smart phones contain a vast array of sensors that measure their surroundings and provide an endless stream of data about the device's orientation in space. In contrast to computers, there are no courses on how to write quickly on touch keyboards and there is no standardized *touch-typing system*. Therefore, every smart phone user teaches himself an individual way to input text quickly. These unique features of mobile devices, however, are in large part still not taken into account, likely because smart phones are relatively new in comparison to regular keyboards.

Back in the 19th century, the telegraph was the dominant form of written long-distance communication. It was observed that telegraph operators, who sent messages all day, were able to identify who of their colleagues was transmitting by listening to their typing rhythm. During World War II, the *Fist of the Sender*, a method for tracking an operator by identifying the rhythm, pace and syncopation of the telegraph keys, was used to follow troop movements. In the 1980s, studies were conducted by the National Science Foundation (NSF) and the NIST in the USA, which established that typing patterns are characteristic for each person [BW12]. Shaffer [Sha78] described in 1978 that typing is a motor programmed skill, meaning that an individual's way of typing is developed by a person over time. Therefore, how someone types cannot be shared, lost or forgotten. Since the movements are organized before they are executed, they are of a *ballistic* nature, which means that they are semi-autonomous and, once they are initiated, cannot be stopped [Sal86]. Both of these properties make keystroke dynamics a suitable soft biometric. Another typical example for a motor programmed skill is gait, which is an active research area for biometric recognition as well. It is possible for humans to recognize a person's gender just by their way of walking [Joh75]. If they know them, they can also determine their precise identity. The same principle can be transferred to automatic recognition.

Every hand movement is divided into fine and gross motor movements. While the former's signals can be stopped until they arrive in the frontal lobe, the latter are executed via the extrapyramidal system and cannot. Further reading may start on [Wik16].

1.3 SCOPE

Banerjee and Woodard [BW12] write that the European Standard *EN 50133-1* [CEN96] requires access control systems to have a False Rejection Rate (FRR) of 1 % and an even lower False Acceptance Rate (FAR) of 0.001 %. For a relation with real-world biometric applications: with iris authentication, which is considered one of the most secure biometric features today, FARs of 0,000001 % and FRRs of virtually 0 % are easily achievable [Dau03]. Since the introduction of the Fingerprint Application Programming Interface (API) in Android 6.0, Google requires hardware manufacturers that want to integrate fingerprint readers into their devices to have FARs of $\leqslant$ 0.002 % and are strongly recommended FRRs of < 10 % [Goo15a]. The best algorithms analyzed by Banerjee and Woodard managed to achieve FARs and FRRs of about 0.1 %, which is very good for a soft biometric system but still doesn't fulfill the requirements for secure biometrics in Europe. These algorithms operate on standard keyboards and often (especially and inherently when using machine learning approaches) they use impostor patterns to train classification. However, in a real-world implementation, people cannot ask random strangers on the street to enter their password, just to get negative samples. Therefore, the question we want to answer in this thesis is: is it possible to substitute impostor patterns with the additional data mobile devices provide while still maintaining a level of security higher than that of already existing statistical methods? We try to find the best combination of touch properties and sensor measurements by capturing usage data from different persons on smart phones in a small user study. It consists of four users providing data over several weeks. We then evaluate different options for the most effective combination of features. More details about the setup and the reasons for why there isn't a large amount of participants can be found in Section 6.1 (User Study). Of course, we don't try to fulfill the criteria of the European Standard with our proposal, simply because soft biometrics are not suitable for high-security environments, as we already mentioned earlier.

2 KEYBOARD AUTHENTICATION

Keyboard authentication systems have the same basic structure as any other biometric system. They require persons to enroll users, create templates and authenticate them. As already described in Section 1.1.2 (Soft Biometrics), such systems can be used in different combinations with strong biometrics. An often used approach, which we also implement, is a two-stage authentication phase: There is a pre-authentication secret that any legitimate user has to provide (in our case, a valid password) to be considered for the second stage, where biometric authentication takes place. Therefore, as with most soft biometric systems, keyboard authentication is not used stand-alone. In this chapter, we describe the general process, which features can be taken for biometric authentication and what types of classifiers can be used.

2.1 BIG PICTURE

Any biometric authentication system is divided into an *enrollment* and an *authentication* phase. Figure 3 shows the way a user has to take through a keyboard dynamics-based authentication system to get a decision on whether they are granted access or not. In the enrollment phase, they enter their password multiple times to allow the system to build a classifier from the features that are extracted from the individual key presses. Depending on the variance of these features and the used classification algorithm, the number of times a password has to be entered before the enrollment phase is finished varies. Nonetheless, this number cannot be arbitrarily high, because developers of

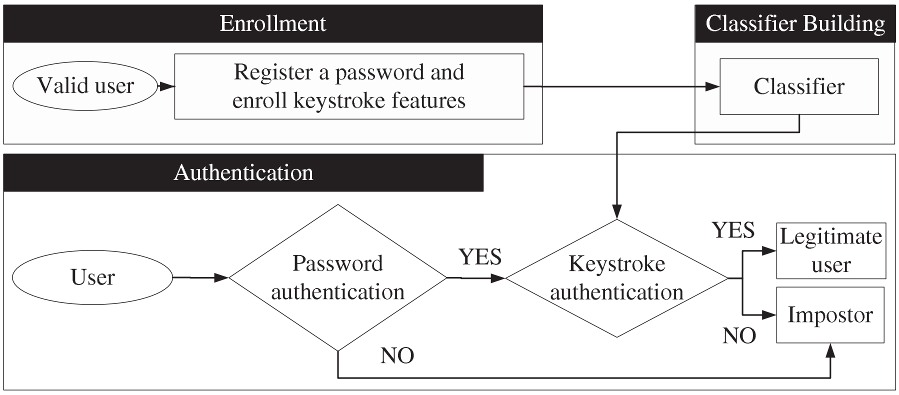


Figure 3: The flow chart of a keystroke dynamics-based authentication system [Tas+14]

such systems have to consider that users get annoyed when they are bothered with tedious tasks for too long. Araújo et al. [Ara+05] found the maximum acceptable number to be ten.

The classifier is the core part of the authentication system, it determines whether an authentication attempt matches the previously built template. There is a vast amount of research on how such an algorithm can be built, ranging from simple statistical methods to advanced machine learning approaches. The biometric performance measures of the whole system, like FAR and FRR, depend mainly on the quality of the classifier. Examples of different systems are given in Chapter 3 (RELATED WORK), where also performance metrics are stated.

In the authentication phase, the user first has to enter a valid (the previously enrolled) password, otherwise they are rejected as an impostor immediately. After that, the keystroke data that was captured during the authentication attempt is fed into the classifier, which calculates a similarity score, derives a binary decision or any other value that indicates whether the given data belongs to the legitimate user. Only if this final decision is positive, the user is granted access.

The described process can be applied to *static* or *dynamic* analysis [MR00]. For static analysis, fixed texts (in most cases, passwords) are used to authenticate users. The same pattern can be used for every attempt and therefore, a high recognition performance is possible. For dynamic analysis, the user's behavior is monitored throughout usage of the device and for every text input. This method has the advantage that it's possible to detect a change of users, for example, when the valid user takes a break and forgets to lock their PC. Although no password input is necessary, the biometric system notices the different typing pattern and can e. g. lock the computer automatically. The big disadvantage of this approach is that it's far more difficult to build effective and reliable classifiers that work with any text input of any length.

2.2 FEATURES

In comparison to touch screens, the features that can be extracted from typing on keyboards are limited, but it is nevertheless enough to perform usable authentication. Figure 4 shows the most common parameters of keystrokes, all based on latencies. The reason is that timing information is the easiest set of features to gather from any keyboard. A *digraph* is the Press-to-Press time (PP) between one and the consecutive key, while a *trigraph* spans three keys instead of two. The Release-to-Press time (RP) is also called the *flight time*, because the fingers are hovering over the keys. On the contrary, the Key Hold Time (HT) is also called *dwell time* [BW12]. Scientists already analyzed user behavior and the parameters of keystrokes that give the

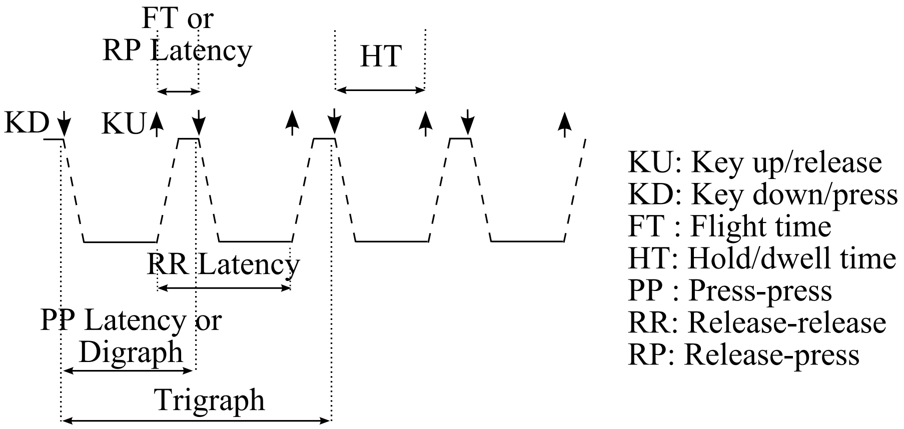


Figure 4: Keystroke timing information [BW12]

best results for biometric authentication. Bergadano, Gunetti, and Picardi [BGP02] found that using trigraphs yielded better results than digraphs or any other *n*-graph, while Robinson et al. [Rob+98] concluded that the HTs are much more important than the inter-key times (Flight Time (FT), PP, Release-to-Release time (RR)). That said, this doesn't seem to be an universal truth, because e. g. Maiorana et al. [Mai+11] found that PP and HT in digraphs work best in their scenario. Because the approach in this thesis is based on their findings, we chose these features as well.

2.2.1 *Mobile Features*

For mobile devices, there is additional information about every touch that can be taken into account. The *pressure* of each key press could already be captured on special keyboards. Saevanee and Bhattarakosol [SB09] found that just by using the pressure of a finger on the screen a very high accuracy could be achieved. Notwithstanding, on capacitive touch screens that aren't pressure sensitive in themselves, it is a virtual value derived from the finger size. This behavior is described in the Android Open Source Project (AOSP) documentation [Goo14c]. Information about the finger *size* was added to the classification e. g. by Tasia et al. [Tas+14], although it was not as effective as the pressure feature. In addition, some touch screens calculate the touch *orientation*, which expresses the angle of the finger on a screen. A feature available on any touch device is the 2D *position* on the pressed key (e. g. near the left upper corner), which amongst other things depends on the hand the user is holding the device in and the used finger(s).

2.2.2 *Sensors*

During our initial research, we found that there already is a considerable amount of work done on keyboard authentication on mobile devices using touch features including pressure, but there was no publication on exploiting the unique properties mobile devices provide: sensors that capture the environment a device is in. Our idea was that users have certain habits of holding their devices when they enter their password. One person may hold a phone in the right hand while typing with the left index finger, while another one might hold it in both hands, typing with the thumbs. Originating from that thought we wanted to enhance the current authentication data with sensor information, which in turn may result in better recognition performance. The Android platform distinguishes between motion, environmental and position sensors [Goo12]. Table 11 in the appendix shows an overview on all available sensors with typical use cases. We were interested in the relative change of positioning in space a phone experiences while the user types, because they might use their device in different positions (e. g. standing, sitting, lying in bed, . . .). This kind of data can only be provided by the motion category, because of which we focused solely on it.

The *accelerometer* measures the acceleration in space (in m/s^2) that is applied to the device – inherently including gravity. Therefore, we also used the *gravity* sensor that is derived from acceleration and indicates just the gravitational force. The exact opposite is the *linear acceleration* sensor that filters gravity from accelerometer readings. Even though all three sensors deliver similar data, we included all of them in our work to see whether ground orientation, acceleration or a combination of both deliver the most discriminant features. The *rotation vector* sensor represents the orientation of a device as an axis around which it is rotated in a certain angle. The X axis is the vector product $Y \times Z$ and points approximately West, the Y axis is tangential to the ground and points toward the geomagnetic North pole while the Z axis points toward the sky and is perpendicular to the ground. Finally, we used the *gyroscope*, which is both available in a calibrated and an uncalibrated version. It measures the rate of rotation (in rad/s) around a device's axes. Gyroscopes have a tendency to drift over time, which means that they yield different motion results than other sensors such as the accelerometer do. This is automatically corrected by the Android system, but developers can also access the uncalibrated gyroscope. It has the disadvantage of not being completely true to reality but has the advantage of providing data with less abrupt jumps introduced by calibration, meaning more smooth and reliable measurements. We included both versions in our study to evaluate which type works best for biometric authentication.

2.3 CLASSIFICATION

The core of good biometric authentication systems is the classifier generated by a specific classification algorithm. It takes the pattern recorded in the current authentication attempt and matches it to a class of patterns that were recorded previously and belong together – in our case the password entries of the user who owns the device. If the pattern similarity to a class is above a certain threshold, the classifier assumes that the class matches. There are numerous approaches to solve this problem with a varying level of complexity, but all the solutions can be categorized into five categories [KAK11]. We annotate the reasons for or against using them in our approach shortly, while an in-depth explanation is given in Section 4.3 (Statistical Classification).

2.3.1 *Statistical Methods*

For statistical methods, measures such as mean, standard deviation and variance, geometric, Manhattan or Euclidean distances in combination with *k*-nearest neighbor approaches are used to find a matching class for a pattern (e. g. searching the template with the minimum mean distance). The most commonly used properties of keystrokes for classification are latency and duration, often also taking digraphs into account. If there are multiple properties to consider, distances are calculated by treating a key press as a vector of numbers. This is the base for *k*-nearest neighbor approaches, where the most fitting enrollment templates are searched for, easily incorporating any distance measure into a simple, understandable and effective classifier. There also have been attempts to calculate a covariance matrix of keystroke latencies to measure the consistency of a user's typing signature. More sophisticated methods use Hidden Markov Models (HMMs) to classify the feature subsets of authentication attempts. As Chang, Tsai, and Lin [CTL12] point out, statistical methods are very resource-efficient and are therefore best suitable for low-power and low-resource mobile devices, even when multiple methods are combined. This advantage can be used without any disadvantages in recognition performance – in a comparison, the best statistical system had an EER of 0.5 % in contrast to 0.43 % for the best (Support Vector Machine (SVM)-based) pattern recognition system [BW12]. Therefore, this was the way to go in our thesis.

2.3.2 *Neural Networks*

Instead of simply executing a sequential set of instructions given by the developer, neural networks explore different hypotheses by themselves and learn the best way to find the most suitable solution to a problem. Similar to the human brain, they build artificial neural pathways through which data flows between processing units (neurons). Neural networks are considered by some to have the greatest potential in biometric classification, but they can be very complex and slow to train. Algorithms such as Deep Learning, which gained popularity in the last years, need vast amounts of processing power to generate satisfactory results. Linear Perceptrons are one of the easier methods to be used for classifying keystroke data. Two separate digraph sets enhance prediction in Back Propagation Neural Networks (BPNNs), while multi-layer neural networks use the time intervals between key presses as features. It's also possible to combine different neural networks for better performance. However, apart from being a rather slow method both during training and evaluation, this approach has a huge disadvantage: it needs positive as well as negative samples for training, which means that in addition to patterns of the legitimate user, it also needs patterns of impostors. This may be alright in academic work or multi-user systems (if every user has the same password), but this is unfeasible in a single-user mobile environment. This is the reason why we abandoned neural networks as possible classification algorithms.

2.3.3 *Pattern Recognition and Learning*

Pattern recognition is the discipline of finding patterns in data and categorizing it into a number of distinct classes. Because the task of a classifier in biometrics comes down to just this, many scientists used and enhanced pattern recognition algorithms to get the best performance for their keyboard authentication system. For example, keystroke duration can be used in a Bayes classifier in combination with Fisher's Linear Discriminant (FLD), which finds a linear combination of features that is either typical for or defines the separation between classes. Implementers also work with Bayes decision rules, a K-means algorithm or a Potential function. Another pattern recognition technique that proved to work quite well is inductive learning. Keystroke patterns can be represented as a time series, so it was proposed to use the Wold decomposition theorem to decompose the input into predictable and unpredictable components. Techniques like Boosting and Random Forests are used for quite some time already as they show good prediction capabilities, although they need very much processing power, just as Deep Learning. Another promising technology in this category are SVMs. They show a remarkable performance, and while most pattern recognition techniques require impostor patterns – just as neural networks do – Schölkopf et al. [Sch+01] first described a single-class SVM that determines whether a sample is similar to a template set or not. Therefore, single-class SVMs might also work in the scenario described in our thesis. Nevertheless, we focus on simpler statistical approaches and leave this topic open for investigation.

2.3.4 *Hybrid Techniques*

Since all the aforementioned classification methods have their unique advantages and disadvantages, it is more and more common for scientists to combine multiple algorithms, so that the weaknesses are compensated. For example, machine learning approaches are often combined with pattern recognition techniques or statistical methods. Different algorithms can be combined either by chaining their outputs so that the next stage works on processed data, or their individual results can be fused, therefore creating an enhanced score.

Search heuristics, such as genetic algorithms, are often used to find optimal solutions for the algorithms' parameters. They work that well for any kind of algorithm optimization, because they can easily handle large databases, provide multiple solutions and can handle various kinds of problems, even non-parametrical ones. Genetic algorithms deliver a good performance especially when used with SVMs. To present a pure evaluation on how sensor data improves recognition performance, we forgo using hybrid techniques and save this area for future research.

3 RELATED WORK

Keyboard authentication has a long history (see Section 1.2 (Motivation)) and therefore, many approaches have already been discussed. Bringing this concept to mobile devices is a more recent development, which is a reason why this area is not as broadly studied. In this chapter, we give a broad overview over the available systems and how they perform. We don't focus just on mobile security but also add "traditional" authentication systems to provide a more thorough overview on what is possible and where the differences and special considerations are for mobile usage. The reports are categorized into three sections based on the input device they work on, and sorted by publication date.

3.1 COMPUTER KEYBOARDS

Keystroke authentication on computer keyboards already exists for a comparably long time and is rather well studied. Many of the approaches show promising results.

3.1.1 *"Computer-Access Authentication with Neural Network Based Keystroke Identity Verification" [Lin97]*

Lin created a neural network-based approach for authenticating users on computer keyboards. He initially planned to use hold time and flight time as the features used for classification. During evaluation, he found that users often press the next key while the previous hasn't been released yet, which caused problems with his method of computing differences. Therefore, he used digraphs as second time feature. He built a three-layered (input I, hidden H and output O) back-propagation neural network. As convergence criterion, the Root-Mean-Square Error (RMSE) was set to a threshold of 0.07. Weight modification was based on error back-propagation and gradient descent. The training set for the classifier was built using three password inputs by each user. From that manually created data, a number of counter-examples were generated by multiplying the values with random data and scaling it by a factor of ten to 0.2. The final set of weights output by the neural network was stored encrypted along with the associated password.

For evaluation, the most efficient number of hidden nodes H was $\mathrm{H} = (\mathrm{I} + \mathrm{O})/2$, because it had a sufficient recognition performance and allowed online training in less than one second. The system was then tested by 90 valid and 61 invalid users. In the first attempt, performance was not optimal and it was discovered that a considerable amount of output weights were near the selected threshold. This caused the author to lower the RMSE threshold to 0.03 and retraining the network. The final evaluation results showed a FAR of 0 % and a FRR of 1.1 %. On a side note, it was also found during the experiments, that typing skill affected the performance of the system: Impostors were allowed to watch valid users input their passwords and imitate them. In this case, FARs were > 0 %. If the valid user was typing proficient (more than 90 words per minute), FARs were five times higher for typing proficient impostors respectively three times higher for non-typing proficient impostors than they were if the valid user was not typing proficient. Unfortunately, the authors don't state any reason for that in their publication. A possible explanation could be that people with good typing skills have more similar and smoother rhythms, while the others press keys in a unique, untrained way that is the most efficient for the individual person.

3.1.2 "User authentication through typing biometrics features" [Ara+05]

Araújo et al. first combined a total of four keystroke metrics for biometric authentication using a statistical classifier. For that purpose, in addition to the typed key code, mean μ and standard deviation δ for digraph, hold time and flight time were calculated and saved as template. If the number of features between enrollment and authentication didn't match (e. g. because a user sometimes used the Caps Lock key instead of pressing Shift multiple times), the attempt was denied immediately. During their study, they found – as already proposed by others – that although a certain amount of enrollment templates is necessary, more than ten template inputs annoy users. This was therefore the number of templates selected for authentication. The threshold for successful authentication was dynamically obtained using the standard deviation, because it was observed that features with higher deviations needed lower thresholds and features with lower deviations needed higher ones. They also implemented an adaption mechanism, where the new sample replaced the oldest one on successful authentication to adapt to changes over time and refine the acceptance threshold.

For their evaluation, 30 users participated on three computers with two different keyboard layouts. They were asked to authenticate themselves as legitimate users between 15 and 20 times (including enrollment inputs). Then, they were requested to authenticate on other users' accounts by telling them the correct password, which was done between 80 and 120 times per person. Finally, the participants were allowed to observe others while they entered their passwords and imitate them afterwards. This was done between 12 and 20 times per person. All of the tasks were distributed over a certain time span and never done all in one session. The analysis showed that the best performance was achieved by using all three metrics at once, resulting in a FAR of 1.89 % and a FRR of 1.45 %. When people were allowed to observe password inputs, the FAR rose to 3.66 %. It was found that passwords with capital letters increased difficulty for impostors, while changing the password to a given string significantly increased FRR to 17.26 % for legitimate users. They had implemented a two-trial authentication, where users had a second chance to successfully login if the biometric data didn't match. If this mechanism wasn't used, FRR increased to 11.57 %. The adaption mechanism decreased both FAR and FRR, but if it also considered impostor patterns to refresh the templates, FAR increased significantly to 9.4 %.

3.2 CELL PHONE KEYBOARDS

Authenticating users by their typing rhythm works quite well for computer keyboards, so the question arises on whether those methods are also applicable to mobile phones. Many of the research papers worked with cell phones where Java Micro Edition (J2ME) applications were installed. Smart phones or even phones with touch screens weren't common at the time.

3.2.1 *"Advanced user authentication for mobile devices" [CF07]*

Clarke and Furnell proposed a keyboard authentication mechanism for traditional cell phones, where users were authenticated during input of telephone numbers, PINs and text messages. They evaluated three different neural networks to compare their performance. The first was a Feed-Forward Multi-Layered Perceptron (FF-MLP), a Best Case Neural Network that took the best performance rate for each user across all configurations and a Gradual Training Algorithm (GTA) that evaluated recognition performance per user at certain training intervals and reset network parameters according to the user's best performance. Feature-wise, they used just one single metric, the flight time for telephone number and PIN input and the hold time for text message input. Analyzing free text generally is a dynamic analysis problem that is comparably difficult to solve. Therefore, they reduced it to static analysis by authenticating just keystrokes of two to six of the most recurrent characters in texts ('e', 't', 'a', 'o', 'n' and 'i'). Each user therefore had five different neural networks, each responding to a different number of common characters, which were dynamically selected depending on the input text. The disadvantage of this approach was that it had to be assumed that every text message contains at least two of these characters.

In the evaluation, 30 users had to enter 30 telephone numbers, PINs and six-digit text messages each, whereas 2/3 of the data was used for training and 1/3 for performance calculation, where the participants acted as impostors. The best algorithm, with an EER of 5 % for telephone numbers, turned out to be the Best Case Neural Network. Unfortunately, because of the large number of iterative tests necessary, it cannot be used practically. The GTA performed second-best, with an EER of 9 and 8 % for PIN and telephone numbers, while the FF-MLP could only achieve 16 and 13 %. Authentication using text messages proved to be more difficult. Results worsened with shorter input texts, because less discriminative data was available. GTA and FF-MLP classified with an EER of 19 respectively 21 %. Because of the bad performance, the authors proposed a biometric framework, where users are continually asked to authenticate again throughout the day. Negative recognition results don't end in a lockout but a gradual readjustment of access levels and more or less rigid monitoring of user activity.

3.2.2 "User authentication using keystroke dynamics for cellular phones" [Cam+09]

Campisi et al. investigated a keyboard authentication mechanism for traditional cell phones that works with text-based password inputs. Numeric inputs were at that time already rather well described, meanwhile it lacked scientific research regarding textual inputs on the keypad of cell phones. The different layout, key shape and response to pressure make findings for traditional keyboards not applicable. Also, in contrast to numerical inputs, texts need multiple key presses per letter. For classification, they used a statistical classifier working with hold time, digraph, flight time and RR latency. From these collected keystroke properties, mean and standard deviation were calculated and used as templates. To perform experiments with user-dependent score normalization, an acquisition distance measure was calculated between all enrollment templates. For authentication, the metrics were captured again and the dissimilarity distances calculated, which were used to generate a global distance by summing them up. Only if this distance was below a certain threshold, the user was authenticated successfully. Normalization was found to be helpful, because the number of keystrokes directly affects the dissimilarity distances. They proposed two different kinds of normalization, a user-independent and a user-dependent one. The simpler, user-independent normalization was just dividing the distance by the number of keystrokes. The user-dependent normalization involved dividing by the variability of the enrollment templates. In the evaluation, they also compared these methods to traditional score normalization methods, such as min-max normalization and double sigmoid score normalization.

The authors specified six passwords with ten characters each, whose letters were equally distributed both over the keys on the keypad and over the number of keystrokes (between one and four) required for the specific character. 30 users then were asked to enter each password ten times on a Nokia 6680 mobile phone. For evaluation, each set of inputs was split into an enrollment set for the first ten inputs and the authentication set for the rest. It was confirmed that the number of enrollment templates significantly improves recognition performance. It started from an EER of 27.02 % for five templates and improved to 16.21 % for ten templates using user-independent normalization. Therefore, it was recommended not to use less than six training samples. When they evaluated their system using the user-dependent normalization method, they could again improve recognition performance to an EER of 14.46 % for ten enrollment samples. Using score normalization approaches nonetheless didn't significantly enhance performance any further. In summary, while user-dependent normalization is an easy method of normalizing data, it proves to be quite effective. On a final side note, it was also found that the password length improves EER with an increasing number of characters.

3.2.3 *"Keystroke dynamics authentication for mobile phones" [Mai+11]*

Two years after their previous paper [Cam+09] on this topic, Maiorana et al. investigated further on the topic of text input authentication on mobile phones. While the evaluation setup was the same as in the previous paper, they employed a new statistical method and tested different refinements. They proposed a classifier that works in scenarios where the number of enrollment templates is low – as it is in real life. As the captured features, they once again used hold time, digraph, flight time and RR latency. The enrolled acquisitions then were compared pairwise and a distance was calculated. After that, it was normalized with respect to password length. As distance metrics, they compared the performance of Manhattan and Euclidean distance. Finally, they proposed four different metrics to characterize keystroke variability of each user:

- mean value of distances to nearest or farthest neighbors,
- of all distances and
- of the distance to the "template keystroke dynamics", which is the acquisition that has the minimum average distance to all others.

They also considered different criteria to select the templates for authentication out of all acquisitions. Minimum Distance (MDIST), Greedy Maximum Match Scores (GMMS) and two different clustering algorithms, Agglomerative Complete Link Clustering (DEND) and fuzzy

C-means clustering, were used. MDIST, which selects the templates representative for a user, and GMMS, however, were modified by the authors to maximize distances between acquisitions. The rationale was that the authentication process works better with templates that carry distinctive information instead of those which reinforce already estimated statistics. Finally, for authentication, the captured data was added to the set of attempts, recalculating the values and finding the distance of the authentication data to the enrollment data. These values then were compared to a threshold for the final decision.

Evaluation took place again on a Nokia 6680 mobile phone, using six passwords with ten characters each. 40 users entered the passwords 20 times during four sessions, with pauses of ten minutes between each session. They had five training attempts before actual enrollment began. Afterwards, the first ten acquisitions were used as enrollment template and the remaining for evaluation. From that data, the best combination of the different authentication metrics was found. First, the pairwise distance metrics were evaluated, which resulted in using Manhattan distances, because EER was 14.72 % instead of 19.68 % for Euclidean distance. Then, the different authentication metrics were evaluated, also considering a Bayes Classifier as well as SVMs and Principal Component Analysis (PCA). However, none of the results could top the mean distance statistics, which was used for the remaining experiments. After this, the best combination of keystroke properties was evaluated, resulting in the use of a combination of digraph and hold time. Then they compared the newly proposed approach to their previous one, where user-dependent metrics were used. In the end, the new algorithm performed better for limited enrollment sets < 10 and had the same performance for greater numbers. Finally, it was tested whether template selection approaches improve authentication. From 12 randomly selected acquisitions per user, a number of templates were selected. It was found that the fuzzy C-means algorithm performed best for $\leqslant 7$ templates, while MDIST and DEND performed best for > 7 templates. That said, it was noted that performance of the fuzzy C-means algorithm strongly depends on the chosen exponential weights, and – because of its fuzzy nature – generally produces varying results. In summary, the best EER that could be reached by combining the optimum set of methods (mean statistics, 10 templates and PP and HT measurements) was 13.59 %.

3.3 TOUCH SCREENS

Although Personal Digital Assistants (PDAs) with touch screens were already known in the early 2000s, smart phones only got popular with the release of the first *iPhone* by Apple in 2007. Therefore, analyzing users' input behavior on non-physical keys is a relatively new field of study.

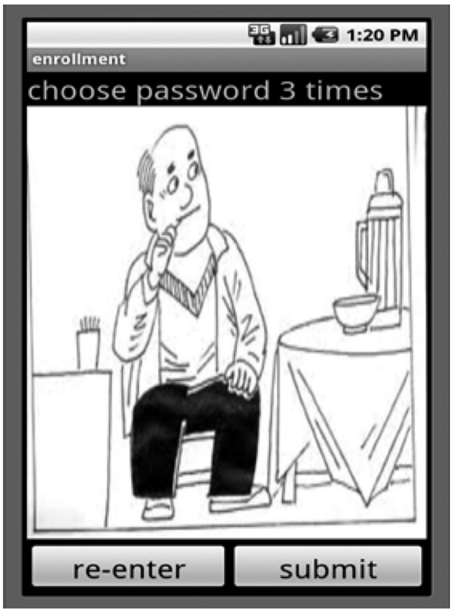


Figure 5: The input UI for graphical passwords [CTL12]

3.3.1 *"Authenticating User Using Keystroke Dynamics and Finger Pressure" [SB09]*

In the advent of the first iPhone and touch screen mobile phones, Saevanee and Bhattarakosol studied the effectiveness of pressure as a measure for authenticating users. They used a laptop touch pad and divided it into 12 sections to simulate a telephone number pad. From the touch data they extracted hold time, flight time and pressure. They found that the pressure of a finger on the selected touch pad model was represented as multiple values over the touch area, hence they used the mean value of each touch. All the values were captured during sweeps every 20 ms. The captured metrics were concatenated into one vector and fed into a Probabilistic Neural Network (PNN), which is similar to *k*-nearest neighbor approaches.

For evaluation, ten participants each entered their phone numbers with ten digits for a total of 30 times in one session. Then, different combinations of the three captured features were tested by splitting the data into ⅔ training and ⅓ evaluation set. They showed that hold time and flight time were bad features both by themselves and in combination. The pressure feature alone and in combination with hold time achieved the best EER of 1 %.

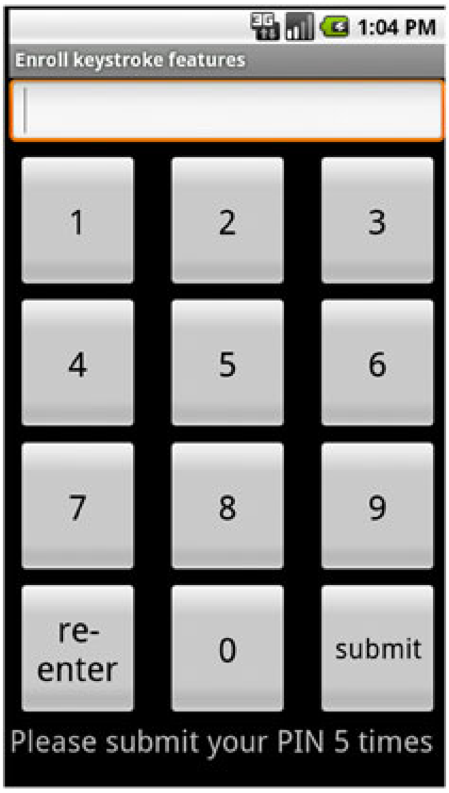
3.3.2 *"A graphical-based password keystroke dynamic authentication system for touch screen handheld mobile devices" [CTL12]*

The password space of the usual PINs on mobile devices in relation to textual passwords on computers is relatively small and they aren't as memorable. Chang, Tsai, and Lin proposed a graphical password that broadens the password space and improves security. In addition, they implemented keystroke authentication by analyzing timing and pressure information. Such a system usually has problems with recognition when users switch between different phones from time to time, because of the different size and layouts of keypads. In their approach, the password input always has the same physical size so that accuracy is not affected. Because of its graphical nature, users don't have to remember all the images, but a meaningful interpretation instead. The system asks users in the setup process to provide their favorite image, which was then scaled to 50 mm × 60 mm and cut into 30 tiles with a length of 10 mm. After that, the user chooses three to six tiles in a certain order. The User Interface (UI) is depicted in Figure 5. The selected squares and their order then represent the graphical password. In the background, the system also captures hold time, flight time and digraph as well as pressure for each tile selection. The enrollment phase requires five training samples. For authentication, the user again selects the same tiles in the same order, which they have to do a certain amount of times to be authenticated successfully. Then, the mean and standard deviation of the features are calculated, as well as the distance to the training samples. Based on a threshold, a decision is made.

100 users took part in evaluation, all of them frequent mobile users. They could freely choose their image and had to provide ten samples. The first five were collected in one session at the same time, while the remaining five were collected over a time span of five weeks on two different phone models. They had different screen sizes on purpose, so that it could be shown that size doesn't affect recognition performance. Ten people then were given the graphical passwords of all 100 participants and had to enter them five times. Evaluation then resulted in an EER of 12.2 % when only using timing information and 6.9 % when combining it with pressure data. It was noted that this method works without having to use any impostor data for training the classifier. The algorithm is suitable also for low-power mobile devices, because classifier building as well as authentication finish in between 1 to 4 ms on average.

3.3.3 *"Two novel biometric features in keystroke dynamics authentication systems for touch screen devices" [Tas+14]*

Two years after their research [CTL12] on graphical passwords, Tasia et al. tried to further enhance and modify their approach, this time relying on the more traditional PIN authentication. In addition to pressure, they also added the finger size as another discriminant feature. Size was chosen because not every person uses the same finger(s) to input their PINs on mobile devices, and while some use the



Please submit your PIN 5 times

Figure 6: The input UI for biometrically enhanced PINs [Tas+14]

fingertip, others use their finger pulp to press on the screen. Also, every individual has different finger thicknesses. For enrollment and authentication, users were presented a 12-button keypad UI with ten digits, one button to re-enter the input and one for submitting the entered PIN, as depicted in Figure 6. Users were asked to enter their PINs five times in a row to be successfully authenticated. The first five input attempts were also used for classifier building. As in their last paper, hold time, flight time and digraph were selected as the relevant timing features. A classifier then was created by calculating mean and mean absolute deviation for all features. For authentication, the average distance between each element of the features was calculated and a decision was made based on a certain threshold.

Before the experiment, all participants were required to familiarize with the mobile device and perform basic operations. The experiment then took place in a classroom, where 100 persons separately had to enter two samples per week of their own PIN with freely chosen length over a period of five weeks. This was done to remove day-to-day typing variability. The first five samples were then used for the enrollment phase, while the remaining five ones were used for authentication and determining FRR. Then, ten randomly selected users were given the PINs of all 100 participants, which they had to enter five times each to produce impostor samples. During the study, it was discovered that most of the people used their right index finger to input the PIN (49 participants), followed by the right thumb (26 participants) and the other thumb, index or middle finger. Evaluation showed that the combination of time features with pressure was most effective and resulted in an EER of 8.4 %, while using all features at once only reached an EER of 10 %. The authors related this result to the security level being high in this mode, causing the FRR to increase. They finally noted that their authentication method is very fast and therefore suitable for mobile use. Enrollment and authentication attempts could be processed in 9 and 3 ms, respectively, even for the longest 8-digit PINs.

3.4 SUMMARY

From all the approaches shown above, we could learn valuable lessons for implementing our own system. The first obvious thing is the selection of features. While there were some experiments in the early days, the timing information extracted from the key presses is pretty much fixed to a subset of HT, PP (digraph), RP and RR in later papers, because they proved to work very well. Regarding recognition performance, statistical methods cannot be considered inferior to other approaches such as machine learning, although they have a wider spread of EERs. This can also be related to the fact that they allow authentication without impostor patterns. Static thresholds for matching templates were found to be bad for performance in the one paper, while they seemed to be alright in the next. In general, it depends on the concrete approach and evaluation helps to find these weaknesses. It's good to work with algorithms that adapt to the data and normalization of features regarding text length is considered a minimum standard.

A very important factor is the number of acquisitions that are used as templates for later authentication. Of course, performance improves with each additional template, but users don't have an infinite amount of patience before they want to use a system for its true purpose. Also, in enrollment mode, the system has to assume that every additional sample comes from valid users, while the probability for impostors trying to gain access rises over time.

Especially for mobile applications, data gathered from the keystrokes doesn't seem to be portable, which results in users having to re-enroll for every device they use. If mobility should be maximized, implementers have to take care of these variances and offer standardized input methods. This, however, bereaves systems of features that distinguish a certain device from all others and potentially gives away performance. Having said that, it was shown that not only keyboards can be used as biometric bearers and the same concepts are also applicable to more graphical approaches such as images or patterns. This would be an interesting field of study for the future, as it strongly emphasizes the multimedial character of smart phones.

Finally, a major observation we made is that none of the studies worked with real-world usage data. In all cases, people were invited to a laboratory environment where they entered their password multiple times. Some authors spread the studies over several days, but we strongly believe that this doesn't depict real user behavior in any way, because they aren't familiar with neither the environment nor the devices. Of course, this is the standard procedure in science, as participants can be monitored and controlled closely. However, we want to study the performance of using sensor data in real use cases that capture the different environments in which mobile devices are used. Therefore, we capture real-world data for our evaluation.

Part II

A NEW COMBINATION OF FEATURES

4 CONCEPT

To create an executable application, one obviously has to choose a programming language and a platform to run it on. We implemented our project in Java on the Android platform because we already had previous experience with both and there are free development tools available. This means that some descriptions in this part are Android-specific and might not apply in the same way to other operating systems. The basic concept – building a classifier from captured sensor data – nevertheless is the same regardless of any implementation details. Therefore, in this chapter we describe the universal fundamentals of our approach to keystroke authentication in a general and abstract matter. It contains a description of the used features, names properties and assumptions regarding the sensor hardware and models the statistical classification process used in this thesis.

4.1 FEATURES

Before we began implementing our project, we thought about the features we wanted to extract from the keystrokes. This would very much influence recognition performance, especially if a very discriminant feature was left out. However, it's not feasible to capture every possible metric from the individual key presses because of performance considerations and space constraints. Therefore, we decided to include the timing features that were deemed most useful by the reviewed papers: PP (digraph) and HT (dwell time). In addition to that, we add properties of the finger touch itself:

- X and Y coordinates of the touch center relative to the touched key's upper left edge,
- size of the area where the finger touches the screen,
- angular orientation (vertical orientation equals to zero) of the touch,
- and normalized physical pressure on the screen.

Finally, we add sensor data as a novelty in our approach. All the sensors in the motion category described in Section 2.2.2 (Sensors) are considered as features, because analyzing the usefulness of each sensor's measurements is an important topic of this thesis. These are accelerometer, gravity, linear acceleration, rotation vector, gyroscope and uncalibrated gyroscope. Depending on the platform and hardware, not all the sensors might be available, but every device we did

FEATURE	DIMENSIONS N	ELEMENTS K
Digraph	1	$k-1$
Dwell Time	1	k
Touch Center	2	k
Size	1	k
Orientation	1	k
Pressure	1	k
Accelerometer	3	k
Gravity	3	k
Linear Acceleration	3	k
Rotation Vector	3	k
Gyroscope	3	k
Uncalibrated Gyroscope	3	k

Table 2: The feature set Δ used in this thesis for keystroke authentication with their dimensions and the number of values

our user study on contains them. Table 2 shows a summary of the used features, represented as the set Δ and the numerical properties that will be important subsequently. The number of dimensions describes how many values one measurement consists of. For example, pressure is only a single value for each key press, while the accelerometer delivers axes in 3D space. The number of elements relative to the K keystrokes describes how many values can be derived thereof. For example, a digraph is a relative number between two key presses. As the first one has no predecessor, it can only be calculated starting from the second one. All the other features are absolute measurements and are therefore available for every single keystroke.

4.2 SENSOR DATA

In Section 2.2.2 (Sensors), we describe the sensors that are available on mobile devices and which of them we use as features in our thesis. For practical approaches and evaluation accuracy, it is important to know about the limitations of these sensors, especially about the resolution of measurements and timing. If a gyroscope can only detect rotations above several degrees, it will not be able to capture the fine movements a person makes when shifting a finger from one key to the next. Then again, if it actually has these capabilities but only performs measurements every second, multiple keystrokes have to share the same data, making them indistinguishable. Typical inter-key timings on traditional keyboards lie around 100 to 300 ms for letters and

HARDWARE	FREQ.	RES.	STDDEV	SENSORS
Accelerometer	**100 Hz** 200 Hz	**12 bits** 16 bits	$\mathbf{0.05\,m/s^2}$	Accelerometer, Gravity, Linear Acceleration, Rotation Vector
Magnetometer	**10 Hz** 50 Hz	**0.6 μT** 0.2 μT	0.5 μT	Rotation Vector
Gyroscope	**100 Hz** 200 Hz	**12 bits** 16 bits	$\mathbf{\frac{10^{-7}rad^2/s^2}{Hz}}$	(Uncalibrated) Gyroscope, Rotation Vector, Gravity, Linear Acceleration

Table 3: Hardware used for gathering the sensor data in this thesis with their measurement frequency, resolution and standard deviation. Bold font indicates Musts while normal font indicates Shoulds.

up to 600 ms for special keys such as numbers [CBT99]. This is different for mobile phone keyboards, because they often have secondary keys assigned. That requires the user to touch the key for a prolonged time. In our study's data, mean inter-key timings are around 300 ms for letters and around 1000 ms for special keys.

Android is the relevant platform for this project, therefore Google's Android 6.0 Compatibility Definition Document [Goo15a] is the reference for these questions. It defines the specifications a hardware manufacturer has to follow for sensors built-in to Android devices. In general, all Android sensors should report an event time in nanoseconds with a maximum synchronization error (to the CPU clock) of 100 ms. New measurements must be reported within $100\,\text{ms} + 2 \cdot \text{sampling rate}$ when the CPU is active. Sensors with a continuous operation mode (which all we used herein support) must have a maximum jitter in periodic data of 3 %. This means that captured sensor data might not have been measured at exactly the same time a key event is fired.

For the hardware used in this thesis, we summarize measurement frequency, resolution and their standard deviation as well as which sensors use it in Table 3. These are the minimum standards we can expect when collecting data for the evaluation. The statements in the Compatibility Definition Document begin with a word marking how strictly the requirements have to be followed. "MUST" defines an absolute requirement and is written in **bold** font in the table. "SHOULD" defines requirements that may be ignored in specific situations after careful consideration and is written in normal font in the table. These definitions follow the Request for Comments (RFC) standards. When using sensors, Android lets the developer also specify a sampling rate in form of a maximum sensor delay. It tells the system the maximum time an app is willing to wait for new data. Typically, the operating system sends events more often, but it might use the whole time slot if resources are sparse. This delay should therefore be set to the maximum feasible value to save power. We decided to use the predefined delay `SENSOR_DELAY_GAME`, which defaults to 20 ms or 50 Hz. Thus, using the inter-key timings described above, a minimum of 15 measurements between pressing and releasing a key for the average keystroke can be expected. This is enough for our purpose.

4.3 STATISTICAL CLASSIFICATION

As already described in Section 2.3 (Classification), one of the most important aspects of any biometric system is the classification process, which matches samples to the templates stored in the database. In this section, we describe the reasons for why we focused on statistical methods in our approach. We step-by-step build the authentication system, starting from the used distance metrics over the variability calculations for the templates and finally arriving at the authentication score. While this chapter presents the algorithms used in our thesis, Section 6.3 (Applying the Classifier) describes which of them work the best and should therefore be preferred. If not noted otherwise, all given formulas originate from [Mai+11].

4.3.1 *Why Statistics?*

When researching papers to use for our biometric authentication approach, we came across several methods that achieve better or worse biometric performances. In Section 2.3 (Classification) we already annotated the various algorithm types with reasons for or against using them, but this section elaborates in more detail on that matter.

Most current mobile devices use Li-Ion batteries, which at the moment provide the best size/capacity tradeoff.

Mobile devices have to carry their own power source for location-independent usage. Energy capacity of the built-in batteries is naturally limited because of the tight space constraints. To maximize battery runtime, very efficient components such as CPUs with deep sleep states are used. They are, however, optimized for energy usage and not processing performance. For software running on these mobile devices, it is of utmost importance that they have to consider power usage and mind resource constraints to keep users satisfied with their experience. For keystroke authentication, this means that some approaches such as machine learning – which are rather resource-demanding – either cannot be used or have to be done outside the device ("in the cloud"). When password data leaves the local environment, security considerations come to mind and have to be handled carefully. We want to focus on improving authentication methods and don't want to complicate matters, therefore we excluded any approaches that can't efficiently be handled on a local device with the available resources.

Another reason for excluding algorithms was the suitability for single-user environments. Pattern recognition techniques usually need multiple classes of patterns from which they can select the most similar to a given template. If authentication in contrast to identification is desired, only one class will be stored in such an environment – that of the legitimate user. Therefore, the most similar one will always be the same. Unlike some pattern recognition approaches that might be able to calculate the similarity of a single pattern to a given template, neural networks don't have this capability. There are other methods which are specifically designed for this task.

Combining the implications of both considerations, we came to the conclusion that statistical approaches best fit our requirements: they consist of simple mathematical formulas that can be calculated on mobile devices quickly and efficiently. Also, they can be effortlessly tailored to use cases with a single user's data for authentication. Maiorana et al. [Mai+11] already proposed and evaluated methods for an environment very similar to ours and they could achieve a good performance with an EER of 13.59 %, so we decided that we would base our approach on their findings.

4.3.2 *Distance Scores*

The distance scores work with the captured data on the most fundamental level. They take single values of the individual features and describe how far away they are in the feature space. A lower similarity score indicates that two given samples' features are near to each other and the user's environment was similar.

For all the following descriptions we define $\mathbf{f}^{\delta}_{u,e}$ as the vector of the individual features $\delta \in \Delta$ that were captured from user u in the e-th acquisition. To visualize the structure of this object, Figure 7 shows an exemplary feature vector for one user with δ features and E enrollment acquisitions (samples). Every acquisition consists of K N-dimensional values $f^{\delta}_{u,e}$. Every one of these corresponds to exactly one key press. The value for K and the number of dimensions is shown in Table 2. A comprehensive summary of the notation used in this thesis can be found in Appendix B (NOTATION).

The distances $D(\cdot,\cdot)$ used in this thesis are either one of the two most common geometrical distances: Manhattan (city block, taxicab) and (Squared) Euclidean distance. They are calculated pairwise between two feature vectors $\mathbf{f}^{\delta}_{u,e}$ and $\mathbf{f}^{\delta}_{u,i}$. Because the text length is freely chosen by the user, they are normalized by sample set size. The analyzed related work also suggests strongly to do so for increased recognition performance.

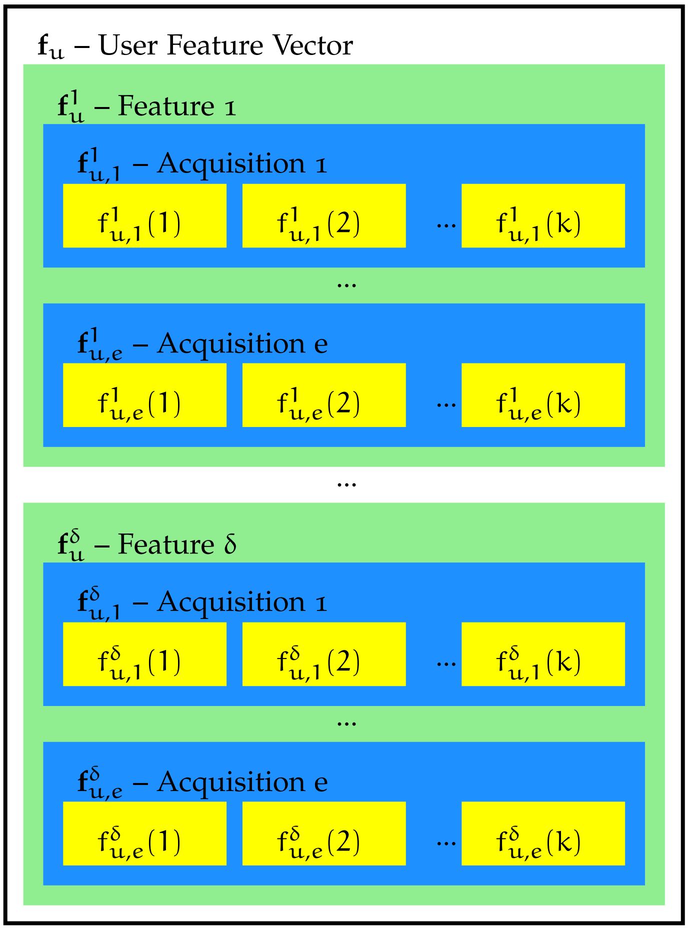

$\mathbf{f}_u$ – User Feature Vector
$\mathbf{f}_u^1$ – Feature 1
$\mathbf{f}_{u,1}^1$ – Acquisition 1
$f_{u,1}^1(1)$
$f_{u,1}^1(2)$
...
$f_{u,1}^1(k)$
...
$\mathbf{f}_{u,e}^1$ – Acquisition e
$f_{u,e}^1(1)$
$f_{u,e}^1(2)$
...
$f_{u,e}^1(k)$
...
$\mathbf{f}_u^\delta$ – Feature δ
$\mathbf{f}_{u,1}^\delta$ – Acquisition 1
$f_{u,1}^\delta(1)$
$f_{u,1}^\delta(2)$
...
$f_{u,1}^\delta(k)$
...
$\mathbf{f}_{u,e}^\delta$ – Acquisition e
$f_{u,e}^\delta(1)$
$f_{u,e}^\delta(2)$
...
$f_{u,e}^\delta(k)$

Figure 7: The feature vector $\mathbf{f}_u$ in a graphical form, showing its individual features δ and their acquisitions $\mathbf{f}_{u,e}^\delta$, which consist of N-dimensional values $f_{u,e}^\delta$.

- Manhattan distance $D_M(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i})$

$$D_M(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i}) = \frac{1}{K} \sum_{k=1}^{K} \left| f^{\delta}_{u,e}(k) - f^{\delta}_{u,i}(k) \right| \tag{1}$$

- Squared Euclidean distance $D_E(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i})$

$$D_E(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i}) = \frac{1}{K} \sum_{k=1}^{K} \left(f^{\delta}_{u,e}(k) - f^{\delta}_{u,i}(k) \right)^2 \tag{2}$$

4.3.3 *Keystroke Variability*

The inter-acquisition distances are then used for calculating the keystroke variability VAR^{δ}_{u} of the user u, which results in a single representative number per feature δ. In Section 6.3 (Applying the Classifier), we evaluate the best of the following four metrics, which should then be used in a real-world application.

- Mean value of the distance of one acquisition to its *nearest* neighbor, MIN^{δ}_{u}

$$MIN^{\delta}_{u} = \frac{1}{E} \sum_{e=1}^{E} \min_{i=1;i\neq e}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i}) \tag{3}$$

- Mean value of the distance of one acquisition to its *farthest* neighbor, MAX^{δ}_{u}

$$MAX^{\delta}_{u} = \frac{1}{E} \sum_{e=1}^{E} \max_{i=1;i\neq e}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i}) \tag{4}$$

- Mean value of all distances between all acquisitions, $MEAN^{\delta}_{u}$

$$MEAN^{\delta}_{u} = \frac{1}{E} \sum_{e=1}^{E} \frac{1}{E-1} \sum_{i=1;i\neq e}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i}) \tag{5}$$

- Mean value of all distances of all acquisitions to the *template keystroke dynamics* $\mathbf{f}^{\delta}_{u,t_u}$ with index $\mathcal{I}(f^{\delta}_{u,t_u}) = t_u$, which has the minimum average distance to all others for this feature, $TEMP^{\delta}_{u}$

$$t_u = \mathcal{I}\left(\min\left[\text{mean}^{E}_{e,i=1;e\neq i} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,i})\right]\right) \tag{6}$$

$$TEMP^{\delta}_{u} = \frac{1}{E} \sum_{e=1;e\neq t_u}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{u,t_u}) \tag{7}$$

4.3.4 *Template Selection*

As argued in Section 2.1 (Big Picture), the enrollment phase cannot be arbitrarily long and the set of templates for authentication has to be fixed at some point. Nevertheless, there is a possibility that the whole set of enrollment templates doesn't yield the same recognition performance as selecting a subset of the most discriminating examples, which describe the user's behavior the best. Also, restricting the template set size results in a higher system performance, because fewer comparisons have to be made and less data has to be stored. We therefore include different template selection approaches that find these representative samples T by clustering all enrollment acquisitions and selecting the most significant ones, which are later used for authentication. This is an already very well-researched process and there are good algorithms, because cluster analysis is an own subtopic of statistics.

Algorithm 1 The MDIST algorithm

```
Initialize $\Delta$, $K$, $E$, $D_{K \times K}$, $AvgD_K$, $T_E$
for all $\delta \in \Delta$ do                    ▷ Calculate pairwise distance scores
    for $m, n \leftarrow 1..K$ do
        $D_{m,n} \leftarrow D(\mathbf{f}^{\delta}_{u,m}, \mathbf{f}^{\delta}_{u,n})$
    end for
end for
for $k \leftarrow 1..K$ do
    $AvgD_k \leftarrow$ mean $D_k$
end for
for $e \leftarrow 1..E$ do                         ▷ Choose $E$ acquisitions
    $T_e \leftarrow \mathcal{I}(\min_{i=e}^{K} AvgD_i)$      ▷ min $\Rightarrow$ MDIST-MIN, max $\Rightarrow$
MDIST-MAX
end for
return $T$
```

Algorithm 2 The GMMS algorithm

```
Initialize $\Delta$, $K$, $E$, $D_{K \times K}$, $T_E$
for all $\delta \in \Delta$ do                    ▷ Calculate pairwise distance scores
    for $m, n \leftarrow 1..K$ do
        $D_{m,n} \leftarrow D(\mathbf{f}^{\delta}_{u,m}, \mathbf{f}^{\delta}_{u,n})$
    end for
end for
for $e \leftarrow 1..E$ do                         ▷ Choose $E$ acquisitions
    $T_e \leftarrow \mathcal{I}(\min \sum_{i=e}^{K} D_i)$      ▷ min $\Rightarrow$ GMMS-MIN, max $\Rightarrow$
GMMS-MAX
end for
return $T$
```

The first two algorithms we test are MDIST [URJ04], depicted in Algorithm 1, and GMMS [Li+08], depicted in Algorithm 2. With MDIST, the N samples are sorted according to their average distance to the others, whereafter E templates with the minimum distance are selected as representative for the user. GMMS minimizes the distance between a set of E templates and the other $(N-E)$ not selected enrollment acquisitions. The rationale behind both algorithms is that there are a number of samples which have maximum similarity and are therefore considered typical. However, Maiorana et al. [Mai+11] proposed modifications to both algorithms, where they don't select the most similar samples but the most dissimilar instead. They argued that it might be beneficial to have templates which represent a broad spectrum of possible behavior. To distinguish both types, we call them MDIST-MIN and MDIST-MAX, respectively GMMS-MIN and GMMS-MAX. Both algorithms deliver deterministic results.

Algorithm 3 The DEND algorithm

```
Initialize $\Delta$, $K$, $E$, $D_{K \times K}$, DEND, $C_E$, $T_E$
for all $\delta \in \Delta$ do                ▷ Calculate pairwise distance scores
    for $m, n \leftarrow 1..K$ do
        $D_{m,n} \leftarrow D(\mathbf{f}^{\delta}_{u,m}, \mathbf{f}^{\delta}_{u,n})$
    end for
end for
DEND $\leftarrow$ Dendrogram$(D)$
for $e \leftarrow 1..E$ do                      ▷ Cut dendrogram at $E$ clusters
    $C \leftarrow$ Cut$(\text{DEND}, E)$
end for
for $e \leftarrow 1..E$ do                      ▷ Select template leaves
    if Leaves$(C_e) = 1$ then
        $T_e \leftarrow C_e$
    else if Leaves$(C_e) = 2$ then
        $T_e \leftarrow$ random $C_{e,1..2}$
    else
        $T_e \leftarrow$ mean $C_{e,1..n}$
    end if
end for
return $T$
```

Algorithm 4 The fuzzy C-means clustering algorithm

```
Initialize $E$, $C_E$, $T_E$
$C \leftarrow$ FuzzyCMeans$(\mathbf{f}_u, E, D(\cdot,\cdot))$
for $e \leftarrow 1..E$ do                      ▷ Select template sample from cluster
    $T_e \leftarrow \min C_{e,1..n}$
end for
return $T$
```

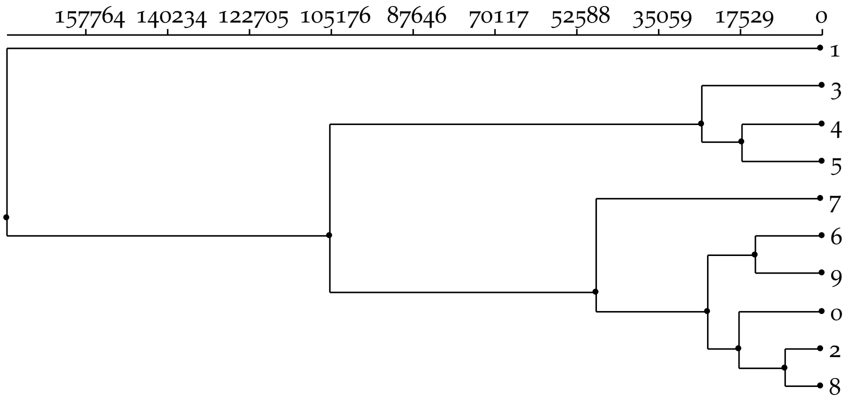


Figure 8: A dendrogram of enrollment acquisitions. On top is the distance between the nodes, the numbers on the right mark the individual enrollments 1 to 10 as leaves.

The second two algorithms we test are DEND [URJ04], depicted in Algorithm 3, and fuzzy C-means clustering [Bez81], depicted in Algorithm 4. DEND uses agglomerative complete link-clustering to create a dendrogram (thereof the name), which is a type of tree. It has the individual samples as its leaves and inner nodes group two children at a time with minimum distance to each other. The root finally groups the two most distant sub-clusters. This results in the leftmost nodes having the greatest similarity to each other, while the rightmost nodes group samples the farthest away. An example extracted from data collected during evaluation is shown in Figure 8. We create the dendrogram by using an external library [BD15], while the template selection is executed as described by the authors. Unfortunately, they didn't mention how the dendrogram has to be pruned to extract exactly K templates. We therefore descend to a level in the tree with at least K nodes and select the rightmost, most distant ones. This conforms with the suggestions by Maiorana et al. [Mai+11] to select the most dissimilar samples. However, it has the disadvantage of potentially preferring outliers. If a selected node is no leaf but it contains exactly two leaves, the template leaf is selected randomly by specification. This leads to non-deterministic outputs. In bigger sub-trees, the leaf with the minimum distance to all others in this node is selected.

Fuzzy C-means clustering originates from data mining and works, similar to SVMs, by dividing an n-dimensional feature space in a way such that the different clusters are separated as well as possible. In contrast to older K-means clustering, its fuzzy properties allow features to be in more than one class with a certain probability. Unfortunately, as with DEND, this adds randomness and the algorithm's output is non-deterministic. It selects E clusters from which the template samples are chosen. These are the ones with the smallest average distance to all others in the cluster. For our implementation we use an external library [Apa16].

4.3.5 *Authentication*

After the distances between all enrollment acquisitions are calculated and the best templates are selected, authentication can take place for every successive sample. Therefore, the distance of the new feature vector $\mathbf{f}^{\delta}_{\tilde{u}}$ captured from the user to be authenticated $\tilde{u}$ relative to the template variability is calculated. Finally, a decision can be made whether the current acquisition belongs to the valid user or an impostor. This is done by comparing the calculated score to a threshold, which has to be determined according to the system's desired security. A common way is to use a threshold such that the EER is reached [Gri08], resulting in an evenly spread FAR and FRR. For high-security environments such as banking applications, the threshold can be set to a value where the FAR is 0.01 %, while for low-security environments such as website logins, it can be set to a value where the FRR is 0.01 %. This means that for ten thousand access attempts, either one impostor or one valid user is classified wrong. Of course, as the metrics are inversely proportional, the number of valid users that cannot or impostors that can access the system also vastly increases.

Which authentication metric $\mathrm{AUTH}_{\tilde{u}}$ to use depends on the variability metric that was calculated for the templates. Of course, they can also be combined to enhance authentication performance, but it has to be considered that resource usage increases as well. In this thesis, we evaluate the performance of single metrics.

$$\mathrm{MIN}_{\tilde{u}} = \frac{1}{||\Delta||} \sum_{\delta \in \Delta} \frac{\min_{e=1}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{\tilde{u},e})}{\mathrm{MIN}^{\delta}_{u}} \tag{8}$$

$$\mathrm{MAX}_{\tilde{u}} = \frac{1}{||\Delta||} \sum_{\delta \in \Delta} \frac{\max_{e=1}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{\tilde{u},e})}{\mathrm{MAX}^{\delta}_{u}} \tag{9}$$

$$\mathrm{MEAN}_{\tilde{u}} = \frac{1}{||\Delta||} \sum_{\delta \in \Delta} \frac{\sum_{e=1}^{E} D(\mathbf{f}^{\delta}_{u,e}, \mathbf{f}^{\delta}_{\tilde{u},e})}{\mathrm{MEAN}^{\delta}_{u}} \tag{10}$$

$$\mathrm{TEMP}_{\tilde{u}} = \frac{1}{||\Delta||} \sum_{\delta \in \Delta} \frac{D(\mathbf{f}^{\delta}_{u,t_u}, \mathbf{f}^{\delta}_{\tilde{u},e})}{\mathrm{TEMP}^{\delta}_{u}} \tag{11}$$

5 IMPLEMENTATION

After deciding about the features that should be used in our approach and the various authentication methods that promise good results, we had to implement it so we could gather data and evaluate it. This project, however, shouldn't only be a means to an end but serve a true purpose instead. We therefore not only propose an exemplary API for apps to use, but also integrate it into the CORMORANT framework [Hin+15], which provides continuous, transparent, extensible, risk-aware and cross-device authentication utilizing multiple biometrics. In this chapter, we discuss the general project structure and design decisions, propose the Biometrics Manager API, describe the statistical classifier implemented in the project and show the example client that was also used for capturing data in the user study. Finally, we talk about the few changes we had to implement for CORMORANT integration and mention some potential security pitfalls.

5.1 PROJECT STRUCTURE

There are many keyboards available for Android devices, which might relate to the fact that custom keyboards are supported since version 1.5, released in 2009 (in contrast to Apple's iOS, which introduced support with iOS 8 in 2014). A variety of them is also open-source, including the AOSP default keyboard (also known as *Google Keyboard*), which ships with every pure Android installation. We therefore didn't see a need for creating our own keyboard and extended an existing one instead. This made it possible for us to focus purely on the biometrics aspect. By selecting the AOSP keyboard, we also gained the advantage of giving users and study participants a keyboard they know and are accustomed to. When beginning implementation, version 4.4 was the latest one, while at the time of writing this thesis, version 7.0 is soon to be released. The differences apart from cosmetic changes are minor and version 4.4 still is the second-most used platform after Android 5 with 30.1 % distribution [Goo16a], so it still is a valid choice.

In the Android source code, the Google Keyboard is internally called LatinIME *and can be downloaded from the Android git repository [Goo14b]. This web server holds the complete Android source code.*

5.1.1 *Hooking the Keyboard*

Setting up the build environment to successfully create a binary of LatinIME from source involved using an Ubuntu Virtual Machine (VM) to build the native library `libjni_latinime` responsible for gesture typing. After that, we investigated the keyboard's functionality and

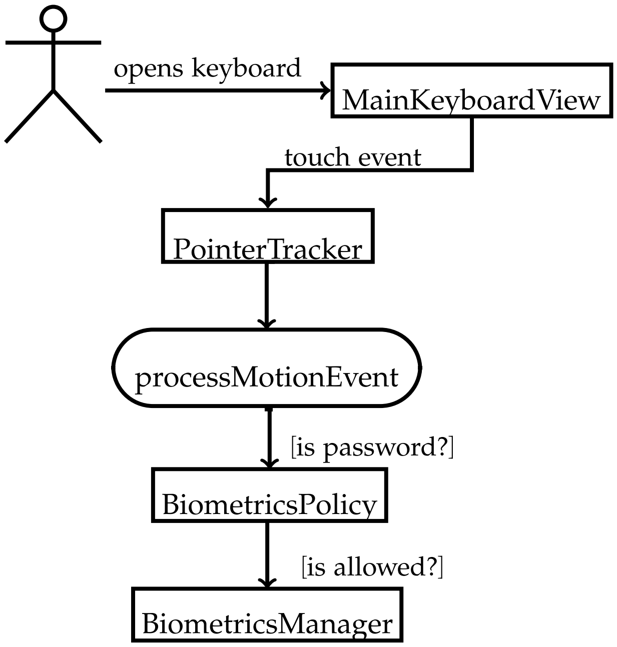


Figure 9: The classes and methods involved in capturing and processing keyboard events for biometric authentication

the classes we had to modify to add biometric authentication functionality. We were looking for places in the code where key press and release events were handled so that we could derive the wanted features from them. We identified `LatinIME` to be the main entry class, which we renamed to *PhonyKeyboard* according to our project name. It contains the typical Android lifecycle methods like `onCreate()` and `onDestroy()`. The keys shown on the keyboard aren't individual objects but merely drawn on the `MainKeyboardView`. This object handles touch events and forwards them to a `PointerTracker`, which is created for each finger that simultaneously touches the screen. It handles all interactions with the keyboard keys, including updating key states (unpressed, pressed, . . .), detecting gestures and secondary key input. For us, the most important method in this class is `processMotionEvent()`, which receives all data about a key event needed for hooking the input flow. This method has a parameter of type `MotionEvent` [Goo09], which is a standard Android class describing movements of fingers, mice, trackballs, etc. According to the set *action*, `PointerTracker` either calls `onDownEvent()` or `onUpEvent()`. After finding the concerned key using the event's coordinates, the keyboard input is finally handled by our `BiometricsManager`. To avoid spying on the user, any keystrokes are filtered before they are handled by the biometrics module: first, it is checked whether the current input field has a password type and second, a `BiometricsPolicy` can apply more advanced techniques (e. g. only allowing certain apps) for ensuring user privacy. The implementation given in our project only allows authentication in the built-in example client. This flow is also depicted in Figure 9.

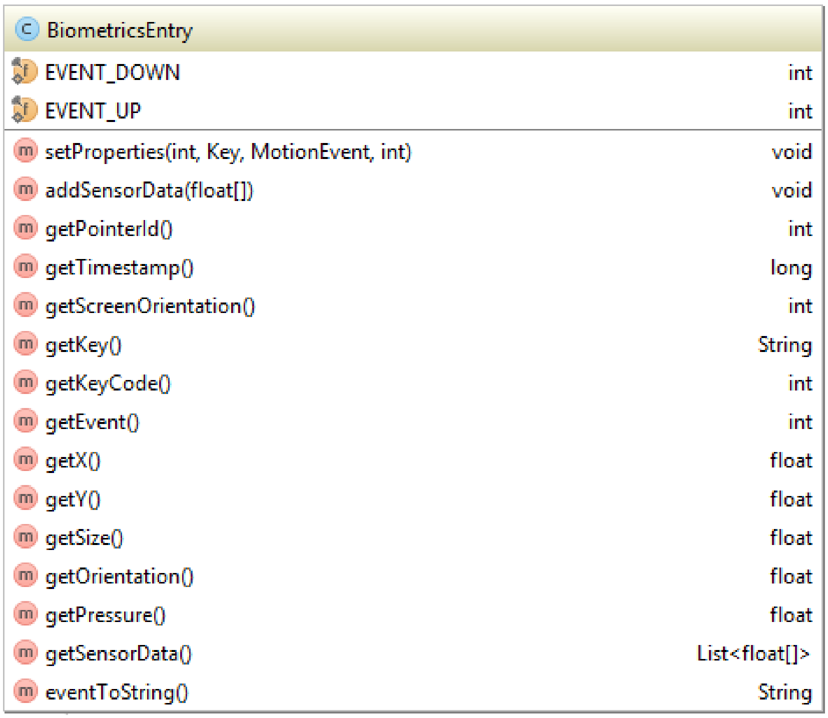
5.1.2 *Biometrics Classes*

As soon as we were able to capture the low-level information about keystrokes, we built our biometric subsystem around it. Class diagrams for the classes mentioned in this subsection can be found in Appendix C. First, we implemented a general abstract `BiometricsManager` class that is the base for all operations on the biometric data. The class hierarchy is depicted in Figure 26. It automatically captures sensor data and provides it to derived classes. If sensors are not available in a specific device, they are automatically hidden from the classifiers. It also acts as a central instance for receiving environment information by them, for example getting the input field's current text and determining the screen orientation. The latter is important to know, because different sets of templates are loaded depending on whether the device is used in landscape or portrait mode. Otherwise, no reliable authentication would be possible when orientation changes. The most important role visible to other apps is the intent-based API it provides, which is described in Section 5.2 (Biometrics Manager API). The class is implemented as a singleton, which emphasizes its central role and guarantees that any keyboard component can access it at any time.

Intents are Android's way of communication between different components or even apps. They represent an app's "intent to do something". Examples are opening another Activity *(dialog) or sending an eMail using a mail app.*

Two classes derive from the abstract Biometrics Manager: `BiometricsLogger` was only used during development. It is a simple wrapper and logs all keyboard events to a Comma Separated Values (CSV) file. Using this class we could get a first impression of how the data received from the keyboard looks like and whether capturing sensor data works. After we could be sure that the hooks in the keyboard don't miss any keystroke events, we started building the second class, `BiometricsManagerImpl`. This one is used for production code and has the capability of loading different classifier classes that are finally used for calculating an authentication score. Apart from forwarding keystroke events and score calculation requests to the classifier, it manages an SQLite database to store data in, which is described in Section 5.1.3 (Database). The classifiers are also given a *biometric context*, which describes the context in which authentication should take place. More information about that can be found in Section 5.2 (Biometrics Manager API).

The information about keystrokes delivered by the keyboard contains many details that aren't necessary for biometric authentication, but those that matter are scattered over several objects. We therefore implemented the `BiometricsEntry`, which contains the necessary information about a keystroke which is forwarded to the classifier. Figure 10 shows its class diagram. The object contains the unique pointer



BiometricsEntry	
EVENT_DOWN	int
EVENT_UP	int
setProperties(int, Key, MotionEvent, int)	void
addSensorData(float[])	void
getPointerId()	int
getTimestamp()	long
getScreenOrientation()	int
getKey()	String
getKeyCode()	int
getEvent()	int
getX()	float
getY()	float
getSize()	float
getOrientation()	float
getPressure()	float
getSensorData()	List<float[]>
eventToString()	String

Figure 10: The `BiometricsEntry` class that holds all information about a keystroke

ID, which identifies a specific finger if multiple ones touch the screen at the same time. This is necessary to associate different keystroke events with the respective keys if a user types so fast that multiple ones are pressed at the same time. It contains

- the event type, which is either *down* or *up*,
- the time stamp in milliseconds at which the event occurred,
- the textual representation and the numeric code of the key,
- the X and Y coordinates of the touch center relative to the upper left corner of the key,
- size, pressure and orientation of the finger on the screen and
- a list of all sensor measurements that were current at the event time.

The entry is forwarded to the `Classifier`, which again is an abstract class to allow maximum flexibility. Its hierarchy is depicted in Figure 27. After the keyboard is opened in a password field, the classifier reads all previous templates from the database, calculates the variability metric described in Section 4.3.3 (Keystroke Variability) and decides whether the new sample will be an enrollment or an authentication acquisition depending on whether template selection has been done. Then, all measurements contained in the entries are collected as keys are pressed. If the backspace key is pressed and password input continues before the text field is empty, the current acquisition is invalidated, because a fluent typing process isn't guaranteed anymore and timings will be messed up. This also occurs when the keyboard is closed during input. When the password was entered completely and the user submits it, a score according to the selected authentication metric described in Section 4.3.5 (Authentication) is calculated. Note that neither the Biometrics Manager nor the classifier know the password for security reasons. The client app signals that it was correct by sending the `BIOMETRICS_GET_SCORE` intent, because otherwise there would be no need for biometric authentication. This intent causes the classifier to associate the captured data with the legitimate user if the score is below the threshold (otherwise, it is rejected), and for example save it to the database as a new template acquisition. More details on the concrete implementations are described in Section 5.3 (Classifier).

Figure 11 shows a summary of the two main actors' lifecycle during biometric keystroke authentication. It begins when the keyboard is opened and `onCreate()` is called. The diagram shows the methods that are executed when the password is entered and a score is requested. Finally, `onDestroy()` is called when the keyboard is destroyed by the system. Note that this lifecycle usually spans multiple text inputs, as the keyboard isn't reinitialized with every text field. It might even be in an active state for a prolonged time, depending on the system's memory usage.

The last major part of the *PhonyKeyboard* app is the example client called *Input Study*. In short, it consists of a main dialog for launching either a test password input, where the keyboard works in normal authentication mode, or a study data input, using which we gathered the data in our study. This mode doesn't perform authentication and only stores the collected acquisitions. More details about this example app can be found in Section 5.4 (Example Client).

5.1.3 *Database*

As previously described, `BiometricsManagerImpl` provides database access for the classifier implementations. Using it, they can store their templates and other management information that has to persist between authentication attempts. The Biometrics Manager doesn't force any structural requirements on the classifiers, but it works with Android's standard concept of *contracts* instead. These are classes that contain Structured Query Language (SQL) statements along with methods for managing tables and updating existing data when a new version of the data structures is available. Normally, this process of

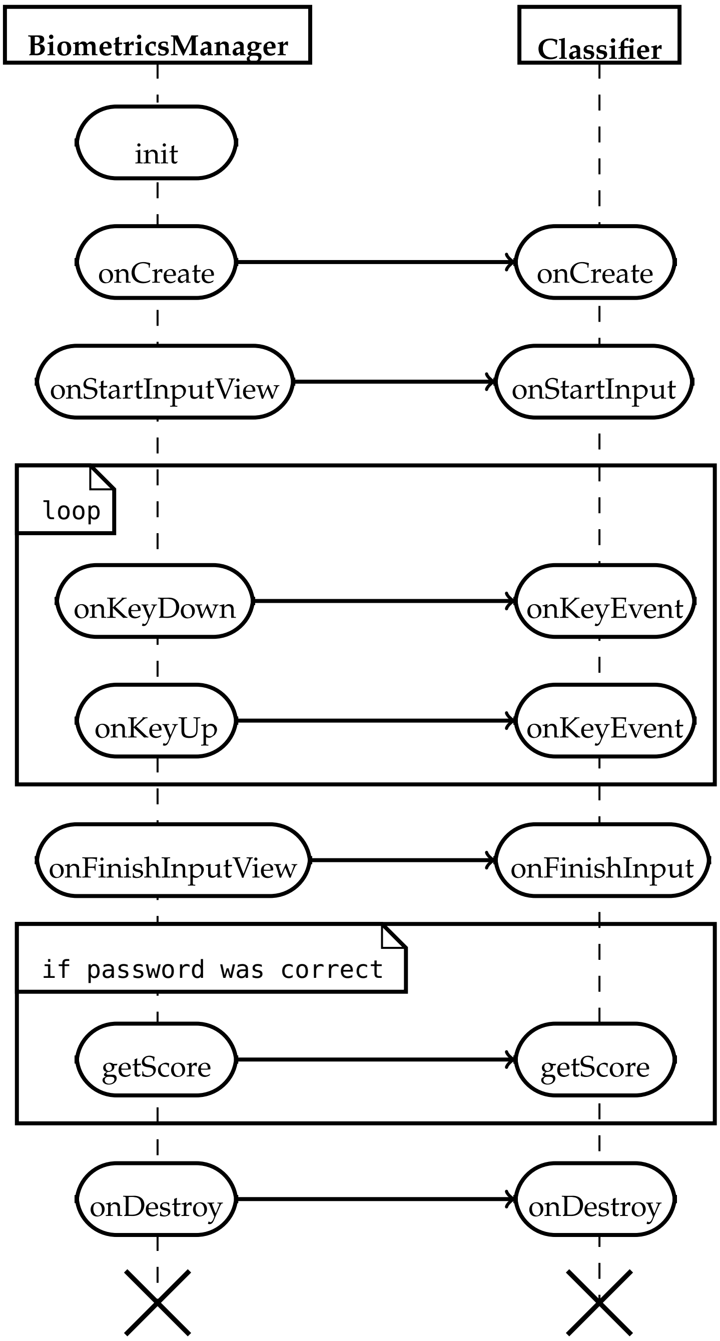
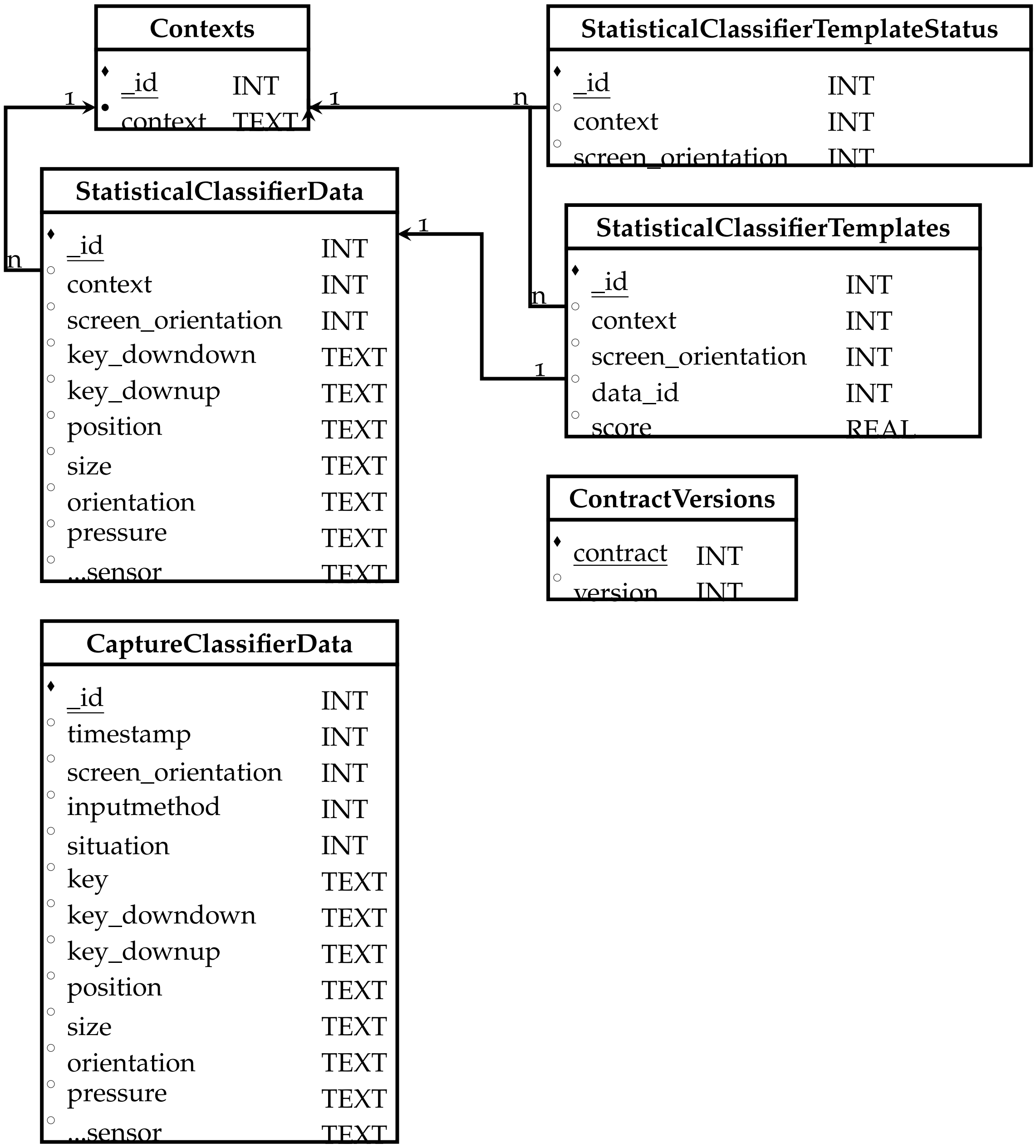


Figure 11: The lifecycle of `BiometricsManager` and `Classifier` when the keyboard is opened, a password is entered, authentication takes place and it is closed again.



Contexts	
_id	INT
context	TEXT

StatisticalClassifierTemplateStatus	
_id	INT
context	INT
screen_orientation	INT

StatisticalClassifierData	
_id	INT
context	INT
screen_orientation	INT
key_downdown	TEXT
key_downup	TEXT
position	TEXT
size	TEXT
orientation	TEXT
pressure	TEXT
...sensor	TEXT

StatisticalClassifierTemplates	
_id	INT
context	INT
screen_orientation	INT
data_id	INT
score	REAL

ContractVersions	
contract	INT
version	INT

CaptureClassifierData	
_id	INT
timestamp	INT
screen_orientation	INT
inputmethod	INT
situation	INT
key	TEXT
key_downdown	TEXT
key_downup	TEXT
position	TEXT
size	TEXT
orientation	TEXT
pressure	TEXT
...sensor	TEXT

Figure 12: A complete overview of the database tables used in the *PhonyKeyboard* project including classifier data tables.

```
// Set a custom biometrics context
EditText password =(EditText)findViewById(R.id.editTextPassword);
password.setPrivateImeOptions("at.jku.fim.phonykeyboard.
    biometricsContext=at.jku.fim.inputstudy.sample");

// Password was entered and is correct, request biometric score
Intent scoreIntent = new Intent("at.jku.fim.phonykeyboard.
    BIOMETRICS_GET_SCORE");
// Low-security environment, use minimum security threshold
scoreIntent.putExtra("at.jku.fim.phonykeyboard.
    BIOMETRICS_SECURITY_LEVEL", 1);
sendOrderedBroadcast(scoreIntent, null, (context, intent) → {
   if (!intent.getAction().equals("at.jku.fim.phonykeyboard.
       BIOMETRICS_GET_SCORE")) {
      return;
   }
   if (getResultCode() == RESULT_OK) {
      int result = getResultExtras(true).getInt("at.jku.fim.
          phonykeyboard.BIOMETRICS_RESULT");
      if (result > 0) {
         completeAuthentication();
      }
   }
});

// Password has changed, clear data
Intent clearIntent = new Intent("at.jku.fim.phonykeyboard.
    BIOMETRICS_CLEAR_DATA");
sendBroadcast(clearIntent);
```

Listing 1: An example code snippet showing the usage of the Biometrics Manager API

calling the respective methods in the right situations is automated by Android, but there is no add-on concept that dynamically adds or removes additional contracts. Therefore, a `BiometricsDbHelper` models this process by keeping track of the currently stored classifiers and their database versions. This metadata is stored in the `ContractVersions` table, while biometric contexts are assigned a permanent unique ID that is saved in `Contexts`. These are the only two types of information the Biometrics Manager has to persist. Each classifier has at least one table in addition to that. The complete database structure is depicted in Figure 12, while the contents of the classifier tables are explained in Section 5.3 (Classifier).

5.2 BIOMETRICS MANAGER API

The keyboard is a self-contained app and needs a way of communicating the authentication workflow to the app that's using it. There-

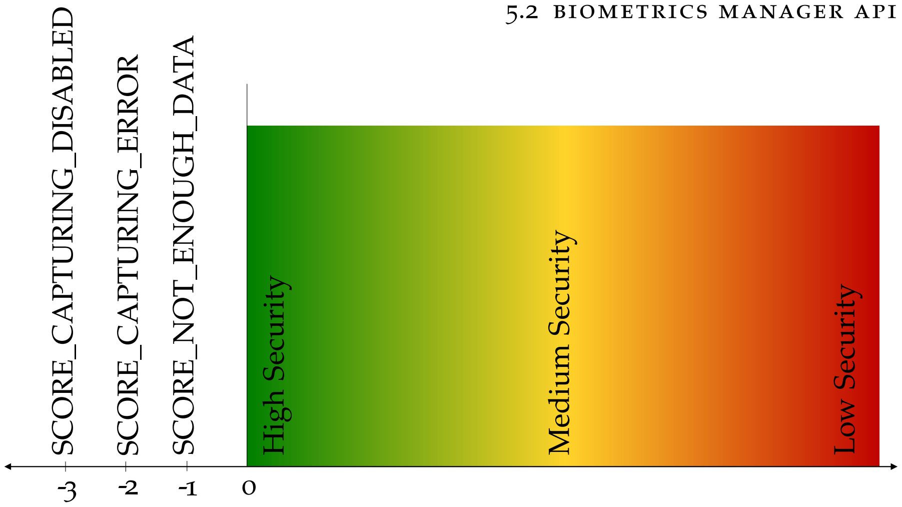


Figure 13: The range of values the score can have, starting from negative numbers for error states to positive numbers for high, medium and low-security environments.

fore, we propose the Biometrics Manager API, an intent-based communication channel between the client and the keyboard. The interface is purposely kept simple to ensure developers can easily integrate it into their existing apps without major changes. An example code snipped can be seen in Listing 1. It consists of two intents, `BIOMETRICS_GET_SCORE` to receive the calculated score after checking the password and `BIOMETRICS_CLEAR_DATA` to reset the templates and data stored in the database when the password changes. This is necessary, because if the new password has a different length than the old one, authentication will fail permanently, because the templates don't match anymore.

`BIOMETRICS_GET_SCORE` has to be implemented by the client as an Android *ordered broadcast*, which means that it expects a result. After requesting the score from the classifier, the Biometrics Manager maps it to an authentication result in an extra field named `BIOMETRICS_RESULT`. Authentication is successful if the score is greater than zero. The intent result also contains a code which defines if the request was executed successfully. There are three states where this is not the case, all resulting in a negative score. Figure 13 shows the range of values.

−1 The set of templates isn't complete yet. The classifier is running in enrollment mode and needs more data for a decision.

−2 An error occurred while capturing the biometric data during typing. This normally occurs when the user presses the backspace key in the middle of entering the password and doesn't completely delete it or if they close and reopen the keyboard while typing. This destroys the typing flow. In rare cases this can also occur when the user enters the password in a way they normally don't, for example by using the symbol keyboard for special characters instead of using the secondary keys. This results in a wrong keystroke count.

−3 Before requesting a score, the BiometricsPolicy is checked whether biometric functionality is enabled. If the score has this value, the client knows that using biometric authentication was denied by the policy.

To ensure maximum flexibility, clients can specify a *security level*, which means that the classifier accepts or rejects the user depending on the individual security requirements. To satisfy these varying requirements, an extra field named BIOMETRICS_SECURITY_LEVEL can be given as a number between zero for maximum and one for minimum security. The default if not stated is 0.5, which corresponds to a threshold that matches the EER. The security level is mapped to the Receiver Operating Characteristic (ROC) curve as described in Section 4.3.5 (Authentication). As it is given with every authentication request, it can also vary every time.

In general, customizing the behavior of a keyboard can also occur using *private IME options* in Android. This is a key-value pair in a string that gets set as a property of the text field. For *PhonyKeyboard*, client apps can set a *biometric context* using the biometricsContext variable that states the context authentication takes place in. If it's not stated, a default context is created by using the app's unique package name. The purpose is to create different biometric profiles for every password the user has, so that no acquisitions of different length or properties pollute the data set. If an app fails to acknowledge this in situations where multiple different passwords have to be entered, capturing errors (score = -2) will occur frequently. Another use case is multiple apps sharing the same account data. It might be desired to keep enrollment times short and avoid having to re-enroll the user in every app. Using the same biometric context, the keystroke templates can be shared between them. In most cases, however, it will not be necessary to define a custom context.

5.3 CLASSIFIER

The base for all classifiers is the abstract class Classifier. It contains several utility methods and inner classes for helping implementations with their data management. Noteworthy is the inner class ActiveBiometricsEntries, which stores *BiometricsEntry* objects for later use. This is important, because the user might touch the screen with multiple fingers at the same time. Therefore, it is critical to associate the correct key presses with their releases. This class makes manual management unnecessary. Provided the current *pointer ID*, it finds matching entries and removes them from memory as soon as they aren't used anymore.

5.3.1 StatisticalClassifier

We implemented two different classifiers. The first, more important one is StatisticalClassifier, which implements authentication as described in Chapter 4 (CONCEPT). When starting input in a password field, it loads the currently stored templates from the StatisticalClassifierData table in the database. If the current environment, characterized by the biometric context and the screen orientation, is contained in the StatisticalClassifierTemplateStatus table, the template selection process has already completed and the classifier starts in authentication mode, where only the selected templates as referenced in StatisticalClassifierTemplates are loaded. The database structure is shown in Figure 12. If there still are templates to be captured, it starts in enrollment mode, which causes all previous acquisitions to be loaded. After this is done, the *keystroke variability* $\mathrm{VAR}_{u}^{\delta}$ is calculated immediately according to the formulas given in Section 4.3.3 (Keystroke Variability).

Note that this subsection doesn't mention the final decision on which distance, variability and authentication metrics to use. This information can be found in Section 6.3 (Applying the Classifier).

After initialization has finished, the user enters their password. With each keystroke, it is checked first if the captured data should be invalidated because the input is partially deleted or the keyboard closed and re-opened in the meantime. Else, if it already is invalid and the input field is now empty, the previously saved data is deleted and authentication can successfully restart. For each keystroke, a new value is added to every feature after the key was released. Key *hold time* and *digraph* are stored as relative numbers between the times of the according events. *Position*, *size*, *orientation* and *pressure* are stored as absolute values for the release event. This ensures that if the finger is repositioned after touching the screen, only the final state is considered. The sensor measurements are stored as the difference between the values at the time of key press and release. The reason is that it's irrelevant what the absolute position of the device in space was at the time the password was entered, but the relative change in positioning is significant. This should be similar every time, independent of how the device is held.

When input is finished, the *authentication score* $\mathrm{AUTH}_{\tilde{u}}$ is calculated according to the formulas given in Section 4.3.5 (Authentication). During this process, it is also verified whether the number of keystrokes and feature values matches that of the saved templates. After the client app has checked the correctness of the password, the Biometrics Manager requests the score from the classifier. According to the given security level, it can determine whether the authentication attempt belongs to the valid user. This is important, because the captured data is stored in the database when authentication is successful to be able to re-evaluate the selected templates. Otherwise, the user's data would be polluted with impostor patterns and recognition performance might suffer. In our current implementation, the template

```
45.5|60.0;53.5|33.25;55.5|44.5;53.0|60.0;4.5|56.0;22.25|51.0
```

Listing 2: The value of the 2-dimensional (X, Y) position field in the database for a password with six characters.

set is fixed after enrollment has finished, but the foundations for dynamic template selection are set.

Finally, the enrollment data, or the authentication data if the calculated score is below the given security level's threshold, is saved to the database. Each of the features is mapped to the database columns in a CSV format, separating each keystroke's values with ";". If they are N-dimensional ($N > 1$), the individual dimensions are again separated with "|". Using this syntax, biometric data for passwords of arbitrary length and features with arbitrarily dimensioned values can be stored. An example for this syntax is shown in Listing 2. If the classifier is in enrollment or authentication mode, no further actions have to be performed. If, however, by adding this sample the required minimum number of acquisitions is reached and therefore, enrollment is completed, *template selection* is executed. During this process, the samples are filtered for most discriminant acquisitions so that the required number of templates T is reached. A reference to these is stored in the `StatisticalClassifierTemplates` table in the database and an entry indicating that template selection has been performed is added to the `StatisticalClassifierTemplateStatus` table. Through this action, the classifier switches to authentication mode.

5.3.2 *CaptureClassifier*

As with the `BiometricsLogger` for testing, we also needed a "dummy" classifier that supports us with collecting data for the evaluation. The `CaptureClassifier` fulfills these requirements. While keystroke variability calculation, authentication and template selection aren't implemented here, keystrokes are processed with the same logic as in the statistical classifier. This means that the same feature values are saved in an internal data structure.

When the Biometrics Manager requests a score, the acquisition is saved to the database in table `CaptureClassifierData`. It has the same structure as the data store of the statistical classifier, but there are additional fields: the *time stamp* at which the sample was collected allows to draw conclusions about the time distribution of participants' inputs. By adding the *keys* pressed, we could observe which keys the inputs were generated with (e. g. secondary key vs. symbol keyboard). As we didn't reset this array when the input field was cleared, we could also find out how often the participants had to restart typing because they made an error. Finally, after each password input they had to fill out a short questionnaire about the participant's environment, i. e. input method and situation. The answers were also saved along with the rest of the data. More information about this questionnaire can be found in Section 6.1.1 (Interaction).

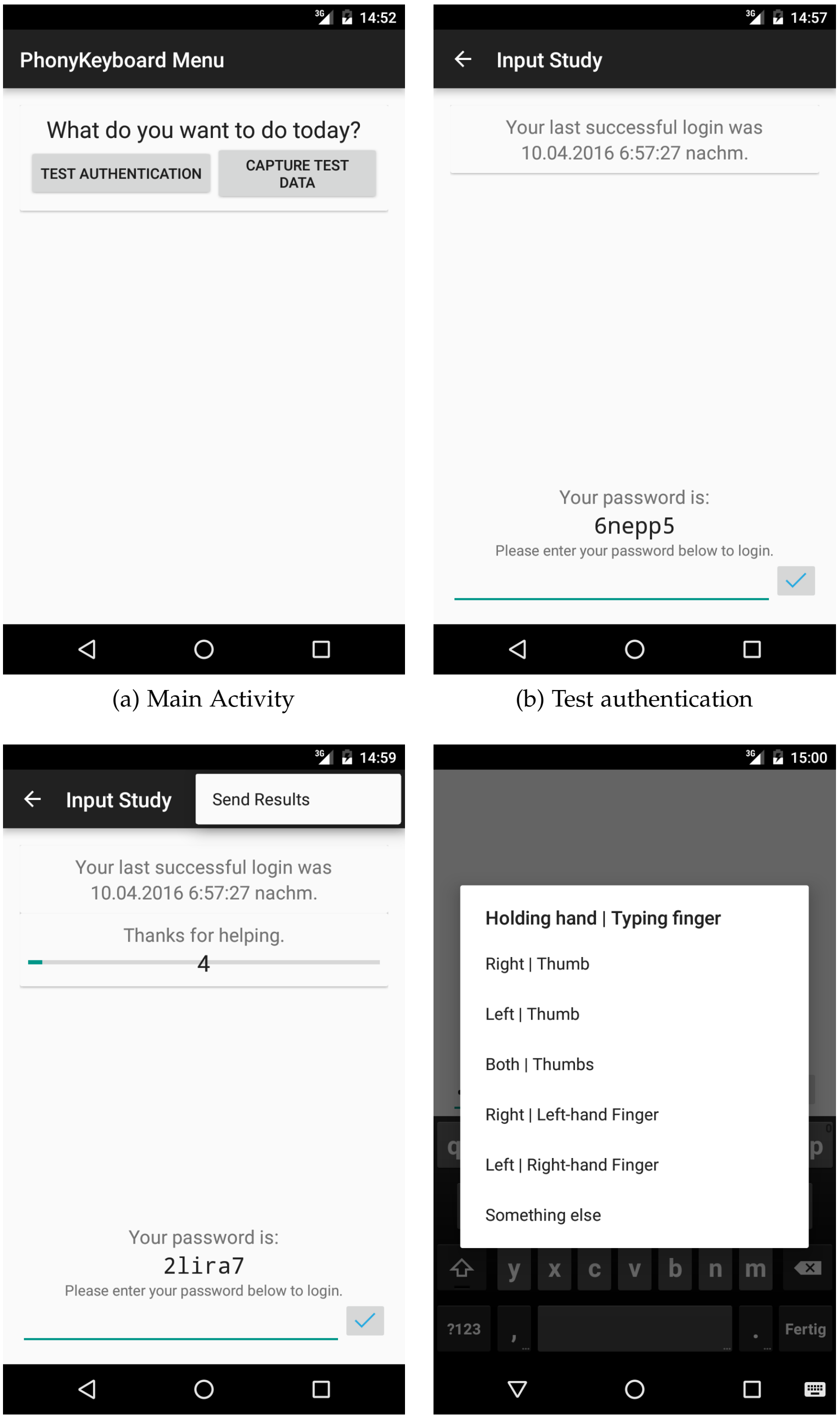
5.4 EXAMPLE CLIENT

To test the classifier and demonstrate an example for using the Biometrics Manager API, we implemented an Android app called *Input Study*. Both the keyboard and the Graphical User Interface (GUI) are packaged in the same file, but the project is configured such that both components get their own icon in the Android launcher. Opening it, the `MainActivity` is shown, which serves as the main menu for selecting the operation mode. Screenshots of the example client are depicted in Figure 14.

Both operation modes are provided by the same activity. The functionality is determined by a flag `isCaptureMode` that is set when opening it.

If *Test authentication* is selected, the statistical classifier is loaded in the keyboard and a random password in the format `/\d\w{4}\d/` (digit – 4-character word – digit) is generated. This is a typical scheme used by many real passwords. To increase realism even more, the word is selected from a German dictionary, so that users can enter it as fluently as possible. If the password is entered correctly, Input Study uses the Biometrics Manager API to receive the biometric authentication result and displays it in a dialog to the user.

If *Capture test data* is selected, the capture classifier is loaded in the keyboard and study mode is started. In this mode, a specific password identical for all participants has to be entered. We chose it to be `2lira7`, because it has insightful spacial properties: the first digit is on the other end of the keyboard than the first two letters, which is also true for the last two letters and the last digit. This forces users to span wider distances across the screen, which increases keystroke latencies. Also, the keys are distributed between the left and right side of the screen, which might tempt participants to use both hands to input the password. As soon as the first authentication attempt was successfully completed in this mode, the user is automatically enrolled in the study. This means that they get a progress bar tracking the number of acquisitions. Also, they get notifications to enter the password in regular intervals. Every time they enter it, they are shown a questionnaire with two questions regarding their current environment. As soon as the authentication was completed 100 times, the study automatically ends. More details regarding the study layout can be found in Section 6.1 (User Study). In the activity's menu, there is an item to send the data that has been captured at that point to the authors. This is implemented as a simple CSV dump of the `CaptureClassifierData` database table, which gets attached to an eMail that is sent via the devices default mail app.



What do you want to do today?

Your last successful login was 10.04.2016 6:57:27 nachm.

Your password is:

6nepp5

Please enter your password below to login.

Your last successful login was 10.04.2016 6:57:27 nachm.

Thanks for helping.

Your password is:

2lira7

Please enter your password below to login.

Holding hand | Typing finger

Right | Thumb

Left | Thumb

Both | Thumbs

Right | Left-hand Finger

Left | Right-hand Finger

Something else

(a) Main Activity

(b) Test authentication

(c) Input study with open menu

(d) Questionnaire after study input

Figure 14: Screenshots of the example client showing the main activity with its mode selection menu, the two different modes and one of the questions that are asked after the password was entered in study mode.

5.5 CORMORANT INTEGRATION

Providing a new API for one app, which has to be implemented in every other app that wants to use it, hinders distribution of this one app, because the benefits have to exceed the cost of implementation. The CORMORANT framework [Hin+15] combines various biometric approaches and even spans multiple devices, which means that there are huge benefits for security-conscious developers that want to make their app more secure. Supporting this framework is easy for a biometric authentication app, because of which we decided to integrate our system into CORMORANT.

After including the `cormorantapi` library, there were just two minor adjustments to make to our project. The first was to declare a new permission in the `AndroidManifest` for signaling CORMORANT that this app contains an authentication plugin. Also, we had to declare the service used for communication between framework and app. Different properties define its behavior, whereby the most important ones are

- `pluginType`, which tells CORMORANT whether the service provides risk assessment or confidence-based authentication functionality,
- and `implicit`, which is a flag indicating whether authentication is performed implicitly in the background or explicitly by requesting the user to perform some action.

The final additions in this file are depicted in Listing 3.

Secondly, we had to implement the `AbstractConfidenceService`, which is called when the framework requests biometric scores. The method needed for that is `onDataUpdateRequest`, which in our case simply forwards the last calculated score from the Biometrics Manager. The returned value needs not be interpretable outside the specific authentication plugin and normalization is automatically done by the framework, which means that there is no customization required by the developer. In addition to pulling the score, pushing is also supported. Therefore, the service has to be bound and `publishConfidenceUpdate` called. The service's code can be found in Listing 4.

5.6 SECURITY

Today, in any software, security is a topic during development that shouldn't be neglected. This is even more true for programs that promise protection, because otherwise the user gets a false sense of security. We therefore describe some attack vectors that could be used for exploiting our app. This is by no means a detailed security analysis.

Antivirus scanners, which are regarded to be essential for Windows systems today, are regularly found to introduce vulnerabilities that wouldn't exist in "unprotected" operating systems.

Communication via the Biometrics Manager API takes place using *implicit* broadcast intents. Because of their nature, any app can receive, execute and alter them. This means that a malicious app could for example intercept the score request by the client and accept any authentication attempt. By specifying the class name the intent should be sent to and therefore converting it to an *implicit* intent, this problem can be prevented.

Sharing the same biometric context can be useful if multiple apps share the same login data (e. g. the *Facebook* and *Messenger* apps). This, however, also opens the system to manipulation. If a malicious app uses the same context as the one it wants to gain access to, it can feed the classifier with the templates of an impostor or even delete the template set, resulting in completely disabled biometric authentication. For apps that don't specify a custom context, this could be mitigated by forbidding the use of custom contexts that match the package name of another app. For others, this is an inherent issue of sharing biometric data and should be considered when depending on it.

All relevant pieces of information, such as keystroke templates, authentication metadata and so on, are stored in an unencrypted SQLite database on the device memory. Normally, access permissions protect this file stored in the app's private directory. However, if root access is granted or the memory is read physically, attackers can modify this database, inject rouge templates matching an impostor, model artificial keystrokes or reset the data and force enrollment mode, where no security is provided. There could also be the possibility of using the collected templates to infer the typed keys, even if this information isn't saved. This could be prevented by moving the database to a secure location, as noted in Section 7.2.2 (Android's Trusted Execution Environment).

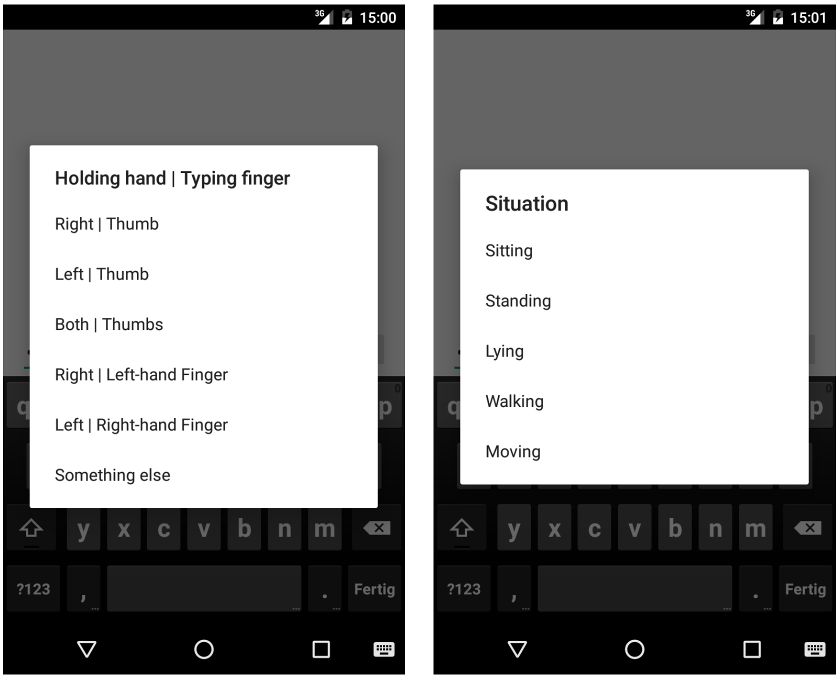
6 EXPERIMENTS

The most important work when proposing novel scientific methods is evaluating their usefulness. This is needed on the one hand for the authors themselves to find out whether their approach is an advancement to the previous ones, and on the other hand for the scientific community and interested readers to verify the results by recreating the experiments. We have investigated various metrics and algorithms that might be able to give good results, but the final choice of which combination to use can only be made after evaluating them. In this chapter, we describe our user study layout and procedure and deliver insight into the captured data. We apply the classifier to the data and find out which metrics should be used in a real-world system and which EER can be expected. Finally, we determine the classification thresholds for different security needs, calculate how the mobile devices battery life is influenced by capturing sensor data while using the keyboard and show how the accuracy of the measurements influences the final authentication result.

6.1 USER STUDY

To be able to evaluate the best parameters for our classifier, we need a vast amount of data of different users. Variations in device usage over time and getting used to entering the password predefined by us meant we had to carry out the study over a prolonged period. Also, as different people handle their device in unique ways, we had to acquire multiple participants. Finally, we wanted to find out whether the templates gathered on one device can be taken to others, which would require only a single enrollment phase per user instead of per user and device. Therefore, we added a control group.

It was an utmost concern for us to gather real-life usage data from our participants. Normally, studies are carried out in laboratories, where people are assigned a predefined setup in which experiments take place. They are closely controlled and monitored to ensure that as much usable data as possible is gathered in a short time. While this method is convenient, it has the disadvantage of not being true to life. People use their mobile phones in different places and environments, they walk or lie down when they type, use left, right or both hands, and so on. If we brought people to the laboratory for one or more sessions of intense password-typing, we wouldn't get nearly as much insight into their behavior than if we studied the users "in the wild". Also, time and effort increase this way, because of which we had to



(a) Input method (b) Situation

Figure 15: Screenshots of the example client showing the two questions with their answers that are asked after the password was entered in study mode.

restrict our set of participants to a small number. We were aware of the difficulties with this decision and that errors might occur – which did. But in the end, the effort payed off and we can present results that mirror users' reality.

6.1.1 *Interaction*

For our evaluation, we add a *capture mode* to our example client, as described in Section 5.3.2 (`CaptureClassifier`). A progress bar shows the participants how many inputs they already contributed and how far away they are from the target. After each successful password entry, two questions are shown, which help us unveil possible connections in the data: Holding hand and typing finger (combined to the *input method*) as well as *situation*. Using the answers we can investigate whether the hand the device is held in or the finger that is used for typing influence the measurements. By asking for the situation, we can see if the position the user is in (lying, standing, moving by tramway, ...) shows up in the data. The user interface for these questions and the possible predefined answers can be seen in Figure 15. To speed up the process for the participants, answers are given and cannot be entered manually.

As soon as a user has entered the first password in the capture mode, they are notified to enter it again in regular intervals until the needed amount is reached. To get samples of a user's activities throughout the day, notifications are shown every 5 hours, while they are suppressed between midnight and 7 AM to avoid annoying participants. This results in an average of 3 samples per day. Also, if users interact with their device in a period of 30 minutes before a new notification would appear, they get notified immediately. If a notification was dismissed without entering the password, it reappears every time the participant uses the device. This increases chances of getting as many acquisitions as possible in a shorter time frame. If no notifications would be shown, users might forget to enter data or spread the samples unevenly throughout the study period.

6.1.2 *Participants*

We selected a total of four participants for our study, three of whom (P1, P3 and P4) owned the same smart phone, a *LG Nexus 4*. The other one (P2) was added to the control group to see whether templates can be shared between devices. He used a *Samsung Galaxy S4*. All participants were right-handed. Everyone got sent a copy of the app, which they had to install on their devices. They got an introduction about what to do and how the app works. They were reminded to treat the input notification like they would a text message: pay attention to it as soon as they can, but don't act if it's inconvenient at the time. Also, they were told to send intermediary results about once a week, in case data would be lost (e. g. because of theft or malfunctions). The task was to collect at least 100 complete samples, resulting in a total of 400 acquisitions.

6.2 CAPTURED DATA

When we first got results back from our participants, we noticed that all the collected sensor measurements were zero. Investigating this issue, we found that there was a bug with collecting sensor data in the Biometrics Manager, which provided the same measurements over and over again. Therefore, we had to do a second study phase with the corrected app. For this phase, we also added the questionnaire at the end of each input, because the already captured data suggested that there might be connections to the input environment.

6.2.1 *Remarks*

During the study, none of the participants entered their password in landscape mode. The following experiments are therefore solely done on portrait mode data. We don't expect that the results would be much different.

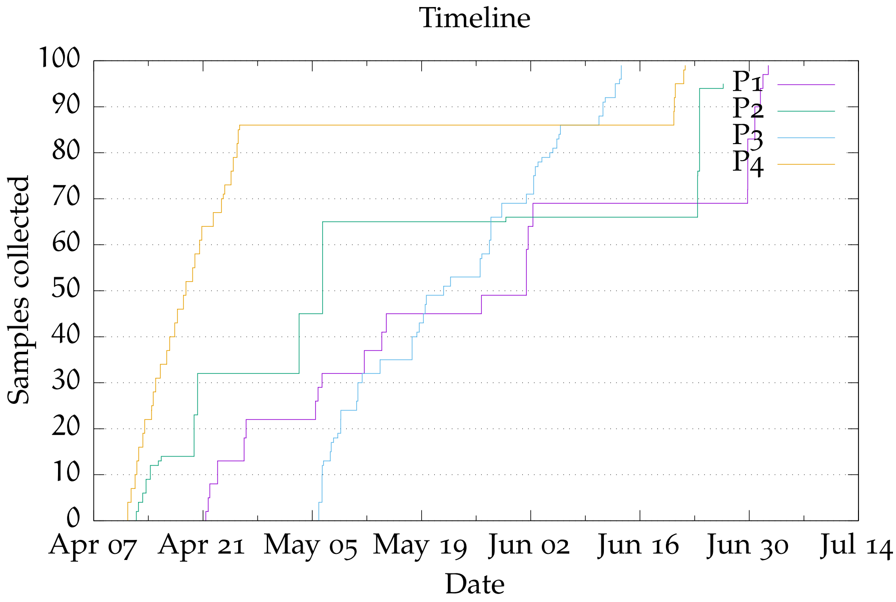
Looking at the raw CSV files, we noticed that for the data captured on the Nexus 4, the uncalibrated gyroscope and rotation vector sensors sometimes didn't deliver data, which resulted in empty columns. We didn't find another bug in the app, so we suspect a problem in the device firmware. A reboot fixed the issue every time, but the affected participants were asked to capture more data so we could ensure that 100 complete, working samples were available for each.

After getting the first data of P2, we wondered why he had two additional keystrokes per sample compared to the other participants. At first, we thought that somehow the enter key would be captured as well, but as we talked to him, we discovered that he was used to typing numbers on the symbol keyboard instead of using the secondary keys. All the others didn't, so we asked him to use secondary keys from that point on to get comparable data.

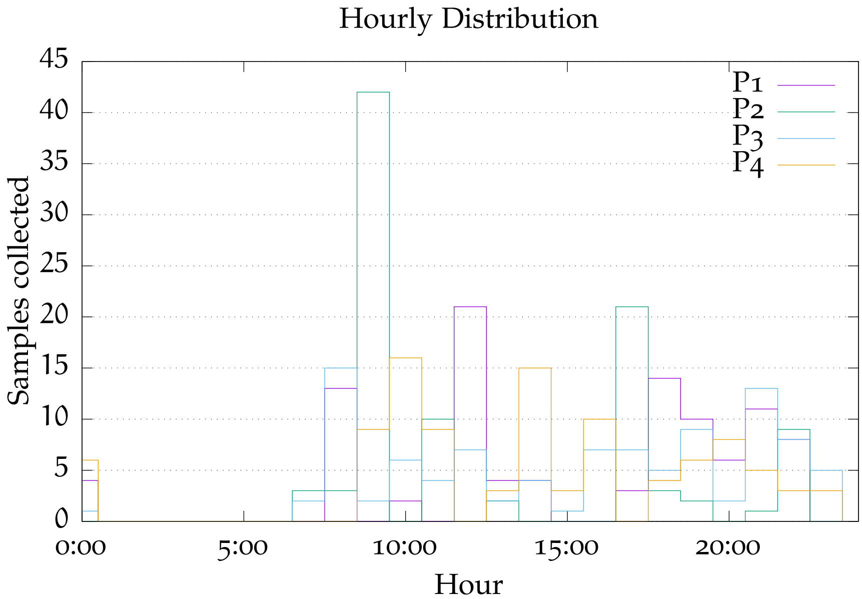
Comparing the data captured on the two different phones, we discovered the difference the built-in hardware makes. For example, on the Nexus 4, the touch size is a discrete instead of a continuous value and has only coarse increments. It seems that finger orientation isn't captured, as the value is always zero. On the Galaxy S4, the pressure value is always one, which makes this feature non-discriminant. Touch size is captured more exactly, although orientation also isn't available. This means that the finger orientation effectively isn't evaluable in our scenario.

6.2.2 *Insight*

Before we apply the classifier and decide which features and metrics to use, we want to look at the unprocessed data delivered by the participants and analyze it. First of all, its interesting to know about the morale of the participants. They weren't monitored, so they weren't forced to follow the notifications and input the password. Figure 16a shows that this concern proved to be real. Especially P2 entered almost 20 samples on one day and generally had a very bursty behavior. In June, it is visible for P1 that he had difficulties with the bug that prevented some sensors to be captured. Therefore, he had to enter many more passwords (a total of 162) to get the required number of complete samples than the rest of the participants. For P4, there is a long flat line visible where the phone was being repaired and he could therefore not provide any acquisitions. In general, P3 and P4 were following the study guidelines best. Figure 16b shows that the collected samples are distributed rather evenly throughout the day. Only P2 seems to have preferred times after breakfast at 8 AM and before dinner at 4 PM. There is a lack of data, however, in the afternoon between 1 and 3 PM. Combining both observations, we can



(a) Timeline



(b) Hourly distribution

Figure 16: Timeline of the amount of samples that have been collected by the participants at a certain date (16a), and hourly distribution at which they were captured (16b).

	SITTING	STANDING	LYING	WALKING	MOVING
RIGHT THUMB	120	50	38	16	2
LEFT THUMB	8	4	11	2	0
BOTH THUMBS	81	18	6	2	15
LEFT FINGER	3	3	0	0	0
RIGHT FINGER	3	0	11	0	0
OTHER	3	0	0	0	0

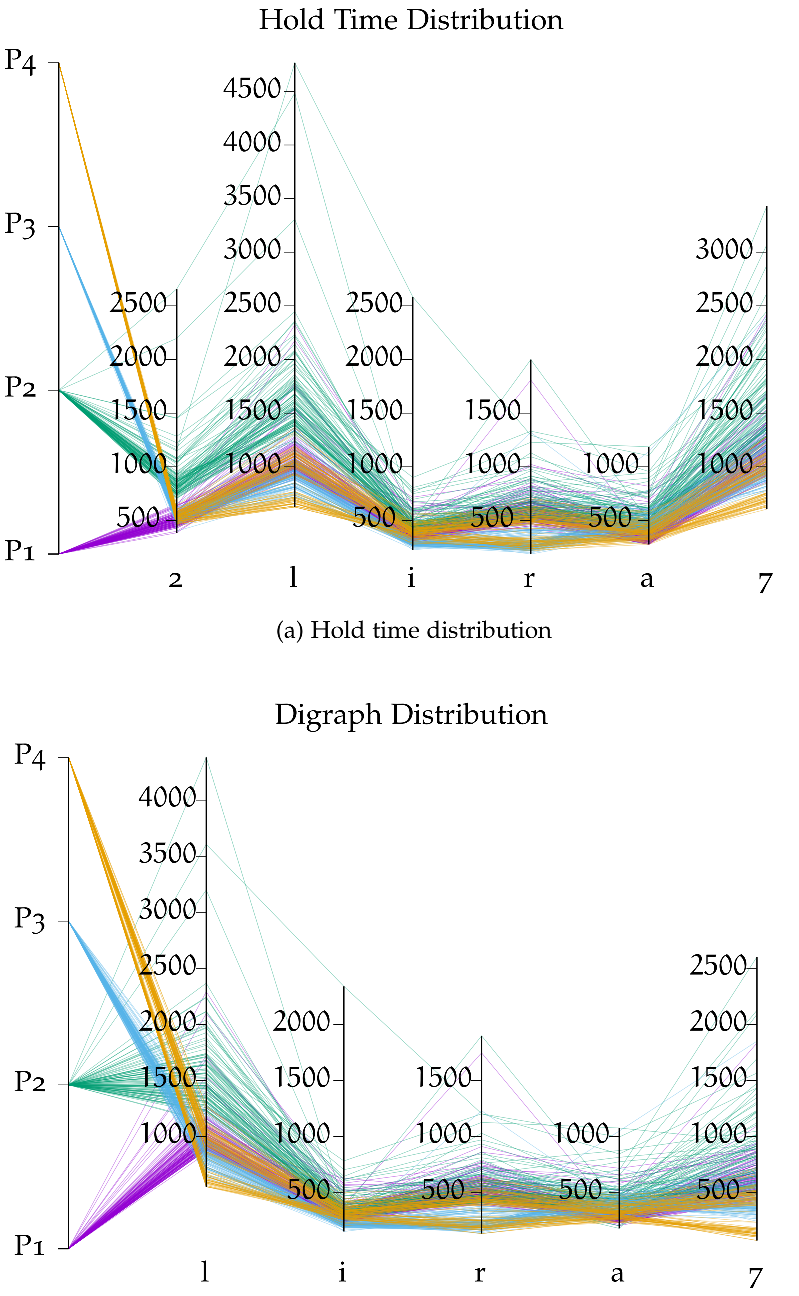
Table 4: Number of samples for the input methods and situations that were asked in the participant questionnaire

draw the conclusion for evaluation, that the variability of the acquisitions over time varies between the participants.

Another aspect worth having a look at is the collected data itself: it often helps to visualize information, because humans themselves are very good pattern matchers. Figure 17 shows the distribution of hold times and digraph per user for each character. The digraph plot begins only at the second character, because it is a relative metric between two keys. While both figures seem chaotic at the first sight, they quickly reveal properties of each participant's typing pattern. For example, P1 and P4 show relatively consistent typing patterns at the lower end of the scale, meaning that they typed rather quickly and didn't stay long on one key. P2 was consistently slower than the other participants and also had the most extreme outliers. P4 has two larger clusters for the "l" and the "r" as well as the "7" key. Unsurprisingly, it can be seen that using the secondary keys results in longer hold times. What's interesting, however, is that P2 had longer hold times for the first letter than for the secondary number key. Also, both graphs show overall similar timing distributions. In general, a psychological study on these findings might bring up reasons for this behavior.

The captured sensor data is difficult to visualize, because it's in 3D space. We therefore only plot the measurements for the letter "i", as can be seen in Figure 18. Most of the values for this sensor are centered around zero, although there are some measurements scattered over the X and Y axes, while the Z distribution is rather small. This means that the phone isn't moved much in height, but there is a distinct clockwise rotation. This, we suppose, is the result of the "i" key being to the upper left of the previous "l" key. Graphs for the rest of the sensors can be found in Figure 28 in Appendix E.

Before evaluation, we want to know whether the metadata participants marked their acquisitions with (input method and situation) also are visible as different categories in captured data. As it can be seen in Figure 19, with the numbers listed in Table 4, most of the pass-



(a) Hold time distribution

(b) Digraph distribution

Figure 17: The distribution of hold times (17a) and digraph (17b) in milliseconds over the password characters per user

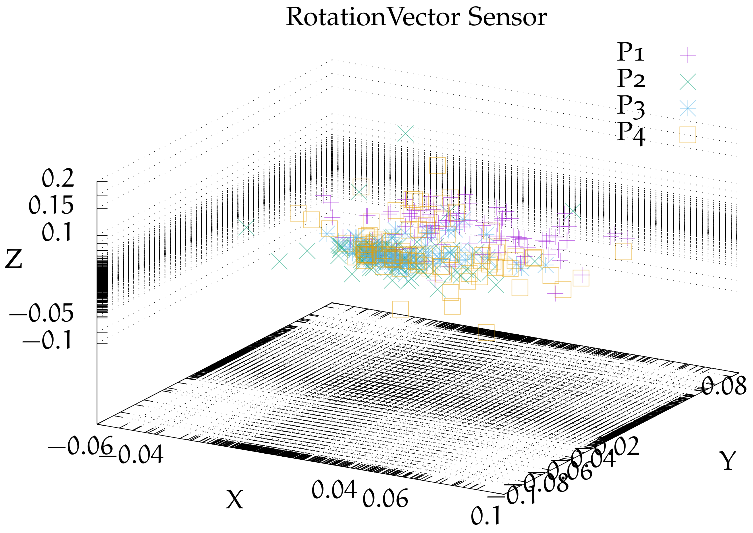


Figure 18: Measurements of the *rotation vector* sensor for the letter "i" from all participants (unitless)

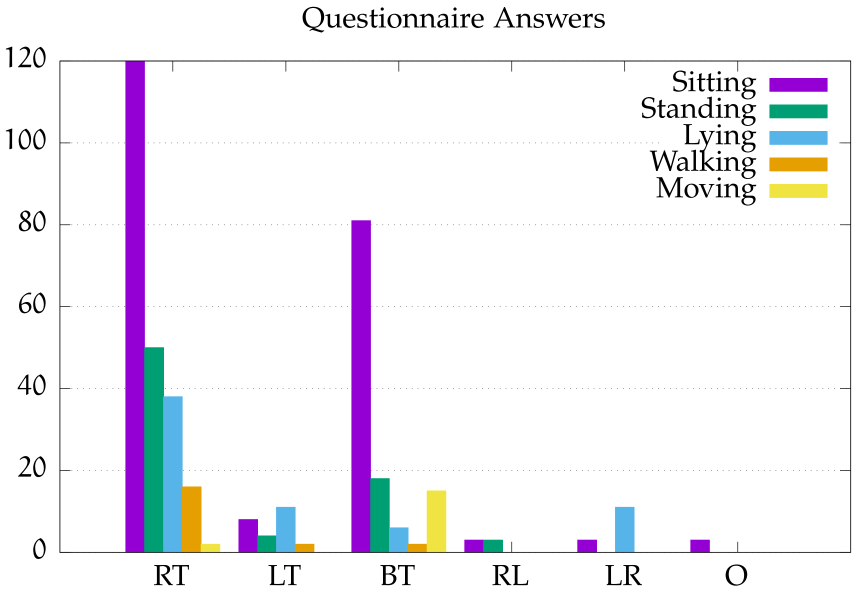
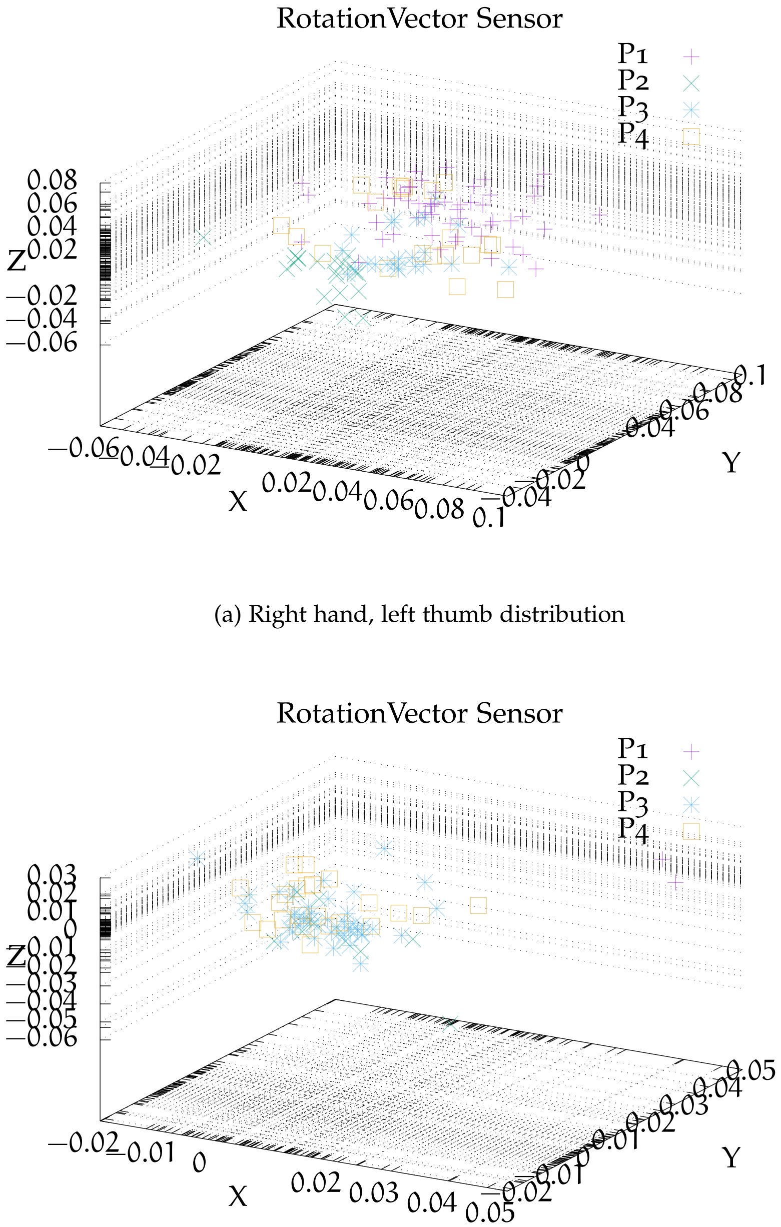


Figure 19: Answers of the study participants regarding the situation they used a specific input method in (either right, left or both hands using thumb (RT, LT, BT), right hand using left finger and vice versa (RL, LR) or other)



(a) Right hand, left thumb distribution

(b) Both hands, both thumbs distribution

Figure 20: Comparison of the rotation vector measurements (unitless) when using the right hand (20a) or both hands (20b) to type while sitting

word samples were input using the thumb of the right hand (297) or while sitting (292). This isn't surprising, because all the participants are right-handed and people normally don't move while using their devices. Only a total of 47 samples were collected while walking or moving (e. g. cycling, going by tramway,. . .). It's also obvious from the graph that the second-most used input method is holding the device in both hands while typing with the thumbs (137 samples), except when lying down, where the different methods are more evenly spread. This is probably because people regularly change positions to avoid over-straining an arm. We have the most data for the right hand and both hands while sitting, so we plotted them for comparison in Figure 20. It can be seen that sensor data really varies between input methods. Z axis distribution is far less while using both thumbs and also on the other axes, movement is less than when holding the phone just in one hand. This is logical, since the device has double the stabilization when it's held in both hands.

6.3 APPLYING THE CLASSIFIER

For faster development and greater processing power, we ported the essential classes from an Android app to a standard Java program. This was generally not difficult, because Android uses many classes that are also available in Java distributions, but we had to re-implement the database functionality that would normally be provided by the framework. In the desktop implementation, databases are only held in memory so we can assure that no data from previous runs alters evaluation results. The main class for the program is `StatisticalClassifierEvaluation`, which provides a command line interface so that different functionality can be called without altering the code. It is able to calculate the scores for single users, perform parameter evaluation or plot graphs of the raw data that are featured in this thesis.

Most of the work happens in optimization mode using the `StatisticalClassifierOptimizer` class. It takes every parameter of the classifier and iterates through the complete range of possible values while keeping all others fixed. This is done in cooperation with a `EvaluationParams` class, which holds all settings for the different aspects of the classifier. The parameters we optimize are the

- used distance function $D(\cdot,\cdot)$ (either $D_M(\cdot,\cdot)$ or $D_E(\cdot,\cdot)$) (two values),

- used variability VAR_u^δ and authentication metric $AUTH_{\tilde{u}}$, which consist of the two associated metrics MIN_u^δ and $MIN_{\tilde{u}}$, MAX_u^δ and $MAX_{\tilde{u}}$, $MEAN_u^\delta$ and $MEAN_{\tilde{u}}$, and $TEMP_u^\delta$ and $TEMP_{\tilde{u}}$ (four values),

$D(\cdot,\cdot)$	EER
Manhattan	10.94 %
Squared Euclidean	**10.44 %**

Table 5: EER of the two distance metrics

- acquisition set size for how many enrollment samples have to be collected before template selection is performed (98 values),
- template selection algorithm used to select the best templates (seven values),
- size of the template set that is used in authentication (nine values) and finally
- sensors that are used in the classifier (64 values).

We don't optimize acquisition and template set size over all possible values, because having less than two enrollment templates isn't feasible. Finding the absolute optimum would be a very time-consuming task where $2 \cdot 4 \cdot 98 \cdot 7 \cdot 9 \cdot 64 = 3\,161\,088$ combinations would have to be checked, so we start by using the results from Maiorana et al. [Mai+11] and iteratively optimize the parameters in the above order, every time taking the best result, where EER is minimum. If varying parameters doesn't have a big influence anymore, we have reached a minimum. The tables and figures below reflect the performance of the individual parameters while all others were fixed to the final – best – selection. In our evaluation, we want to stay true to our promise and again want to be as realistic as possible. Therefore, the methods are evaluated by training the classifier using the study acquisitions in the order they were captured. The training set consists of the first few samples, while the rest of them is used for testing. We take the data of each participant as the valid user once, while all others act as impostors. For the evaluation, we ignore the data of P2 as the control group, resulting in a total of three valid users with 300 positive samples and six impostors with 900 negative samples.

6.3.1 *Distance Metric*

Already the first experiment shows that the results of Maiorana et al. aren't directly applicable to our scenario. While they concluded that the Manhattan distance results in the best EER, in our case, as shown in Table 5, *Squared Euclidean* works better with an EER of 10.44 % instead of 10.94 %. It's a small but noticeable difference.

VAR_u^{δ}	$\mathrm{AUTH}_{\tilde{u}}$	EER
MIN_u^{δ}	$\mathrm{MIN}_{\tilde{u}}$	**10.44 %**
MAX_u^{δ}	$\mathrm{MAX}_{\tilde{u}}$	15.09 %
MEAN_u^{δ}	$\mathrm{MEAN}_{\tilde{u}}$	12.78 %
TEMP_u^{δ}	$\mathrm{TEMP}_{\tilde{u}}$	12.43 %

Table 6: EER of the four keystroke variability and authentication metrics

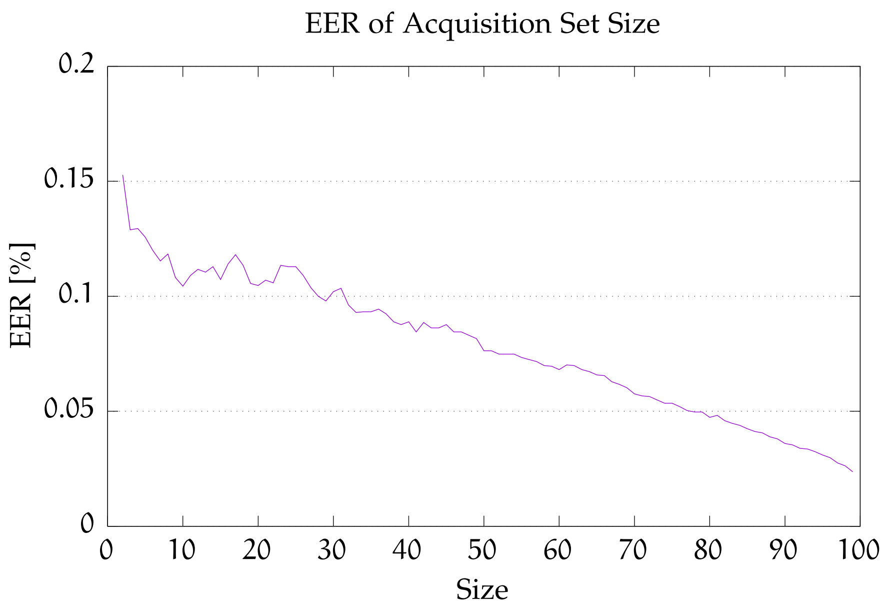


Figure 21: EERs of different acquisition set sizes

6.3.2 *Keystroke Variability Metric*

For the keystroke variability and authentication metric, we could reproduce the results of Maiorana et al., where MIN_u^{δ} and $\mathrm{MIN}_{\tilde{u}}$ were the best formulae. This is true as well in our approach, where it outperforms the other methods by at least 1.99 %, as shown in Table 6.

6.3.3 *Acquisition Set Size*

The acquisition set size is difficult to absolutely optimize, because, as we already mentioned in Section 2.1 (Big Picture), users don't have an arbitrary amount of time and patience to train the system. We nevertheless want to know how effective the authentication process gets with increasing enrollment set size. In Figure 21, we show that we could achieve a minimum EER of 2.37 % when using 99 samples for enrollment and one last sample in authentication. Of course, this result could be expected, because the templates then are very well fitted

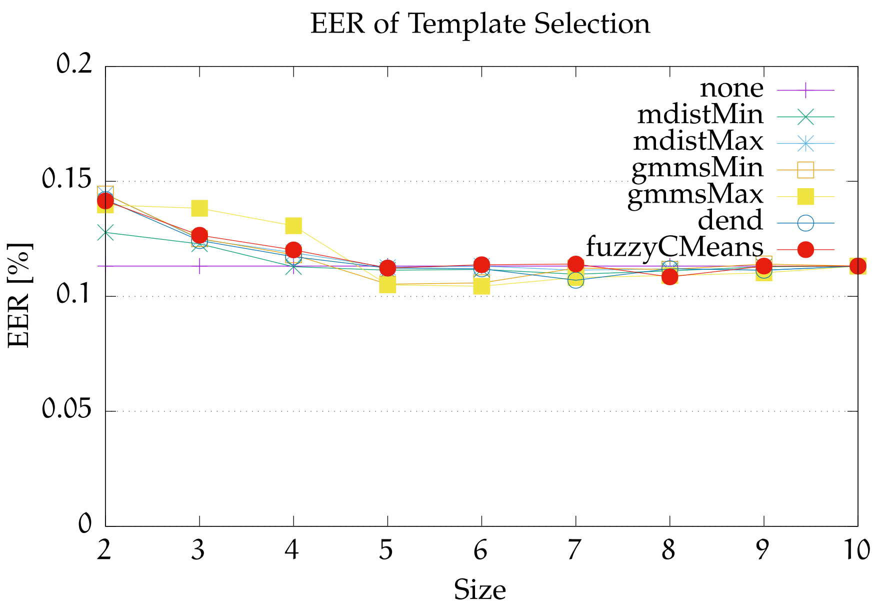


Figure 22: EERs of the template selection algorithms at specific template set sizes

to the user. However, an interesting aspect to note is that while performance stabilizes at about 30 samples and then linearly decreases, there is a local minimum at the beginning of the graph, which conveniently lies at a *set size of ten* with an EER of 10.44 %. This means that we can use exactly the maximum feasible number of enrollment acquisitions in our system and get a good performance.

6.3.4 *Template Selection*

After we have fixed the acquisition set size, we want to determine whether template selection is advantageous to performing none, which

TEMPLATE SELECTION	EER	SET SIZE
none	11.32 %	10
MDIST-MIN	10.96 %	7
MDIST-MAX	11.14 %	7, 9
GMMS-MIN	10.53 %	5
GMMS-MAX	**10.44 %**	6
DEND	11.05 %	9
Fuzzy C-means	10.85 %	9

Table 7: Peak EER of the six different template selection algorithms with the optimum template set size

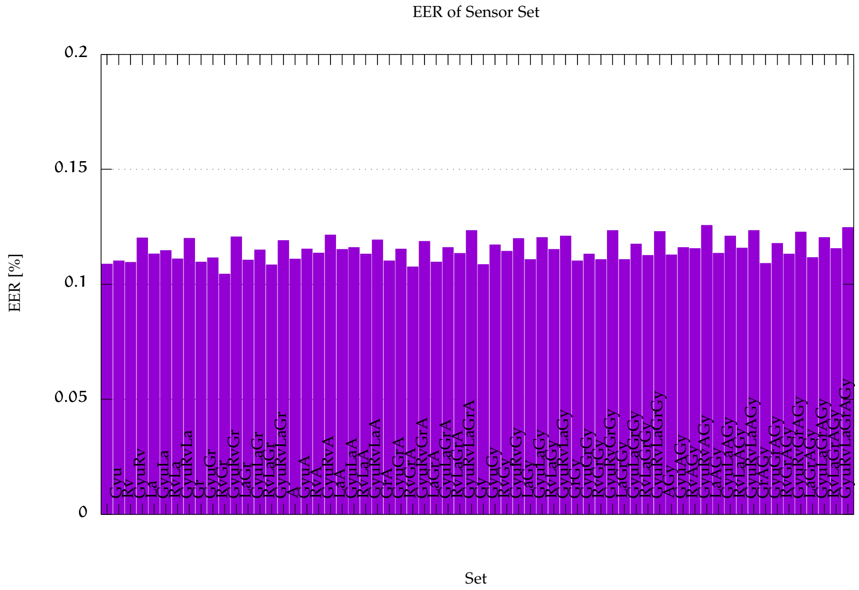


Figure 23: EERs of different sets of used sensors

already results in EERs of 11.32 %, and how many templates should be used. We optimize both template set size and selection function in one go, because different approaches have their optimum at different sizes. We depict in Figure 22 that for small set sizes up to four the algorithms aren't satisfactory, but soon after that, they enhance EERs. The most promising algorithm is *GMMS-MAX*, which again is different to Maiorana et al., where MDIST-MAX had the best results. In our system, it is the second-worst, achieving an EER of 11.14 % with seven or nine templates. Table 7 summarizes the maximum EERs obtainable with the different selection methods and their optimum set size.

6.3.5 *Sensor Selection*

The most interesting evaluation for our approach is: which (if any) set of sensors should be used to improve keystroke dynamics-based authentication systems? To answer this question, we calculated the powerset $\mathcal{P}(\Delta)$ of the set of all sensors and evaluated each element thereof. As can be seen in Figure 23, the more sensors are used in general, the worse results get. Using all sensors results in an EER of only 12.47 %, while the best performance is reached by using the *rotation vector* sensor in combination with the *gyroscope*. As we already showed in Figure 18, clusters can also be seen in the rotation vector visualization and the data in Figure 28c in Appendix E, showing gyroscope measurement distribution, also is widely spread. This makes clustering easier than if all measurements have similar values. If the accelerometer is added as a third sensor, the second-best EER of 10.76 % can be achieved. However, not using any sensors at all al-

SENSOR SET	EER
Rotation Vector, Gravity	**10,44 %**
Rotation Vector, Gravity, Accelerometer	10,76 %
Rotation Vector, Gravity, Linear Acceleration	10,85 %
Gyroscope	10,86 %
{}	10,88 %
Gravity, Accelerometer, Gyroscope	10,91 %
Rotation Vector	10,96 %
Gravity	10,97 %
Gravity, Accelerometer, Linear Acceleration	10,97 %
Uncalibrated Gyroscope	11,02 %

Table 8: The ten sensor sets with the best performance, sorted by EER

PARAMETER	VALUE	EER
Distance	Squared Euclidean	7.63 %
Variability	MIN	7.66 %
Acquisition Set Size	99	1.77 %
Template Selection	GMMS-MIN	**7.31 %**
Template Set Size	5	7.31 %
Sensors	Rotation Vector, Gravity	7.66 %

Table 9: Peak EERs of the optimization process when including the control group

ready results in 10.88 % EER, which is the fifth-best. Table 8 shows the ten best sensor sets sorted by EER.

We also try commonly used random evaluation, where we randomly sort samples for training and testing and perform evaluation 400 times, so that statistically, every sample was the first once and we can get a comparison to using the data sequentially. When run with the same parameters as before, the EER rises to 15.53 %. This is likely due to the fact that half of the participants entered their samples in batches after a certain time (compare to Figure 16a), having less varying data. This results in a high probability of training the classifier only with similar samples, not taking into account different environments.

6.3.6 *Control Group*

The control group was added to the study for identifying whether the templates captured during enrollment are portable and can be synchronized between different devices of varying size. Although we already conclude in Chapter 3 (RELATED WORK) that this probably isn't the case, we want to test this using our own data. As the Samsung Galaxy S4 used by P2 has a 4.99" screen and the LG Nexus 4 only 4.7", differences in input behavior are expected. We therefore also add P2 to the optimization process and see how EERs change. As the best performance we can achieve using the same parameters as before is 7.66 %, which makes a difference of 2.78 %, it is obvious that the data has considerable differences that can be used for better discrimination by the classifier. In Table 9 we state the best EERs that optimization of the individual parameters could achieve on this data set. As we already mentioned in Section 6.2.1 (Remarks), there are a few differences in the data that can be traced to the used hardware. As conclusion, the enrollment templates might be portable if the devices have similar hardware and screen sizes, but for the general case, it's not possible and re-enrollment has to take place on each device. Otherwise, the templates would get tainted and higher thresholds would be needed, which in turn enhances FAR.

6.4 CONCLUSION

In summary, we were able to show that touch screen keyboards have different requirements to biometric authentication systems and the approaches from physical keyboards cannot be used directly. For including sensor data into authentication, the method shown in our thesis works, but it only improves performance by 0.44 %. This proves that sensors can help identify valid users better, but it also means that approaches of including them other than taking the relative value at the time of key release have to be investigated as well. For example, Giuffrida et al. [Giu+14] use a sliding window over the sensor measurements of accelerometer and orientation sensor in combination with statistical methods, which results in an EER of 0.08 %.

If used in real-life systems, the threshold for authentication has to be set. As mentioned in Section 4.3.5 (Authentication), we discuss multiple ones for different security needs here. As enabling sensors during inputs also influences battery life of the device, we give an estimation on how much this can be. Also, Android requires sensors to have a certain minimum resolution. We discuss how changing it can influence authentication results.

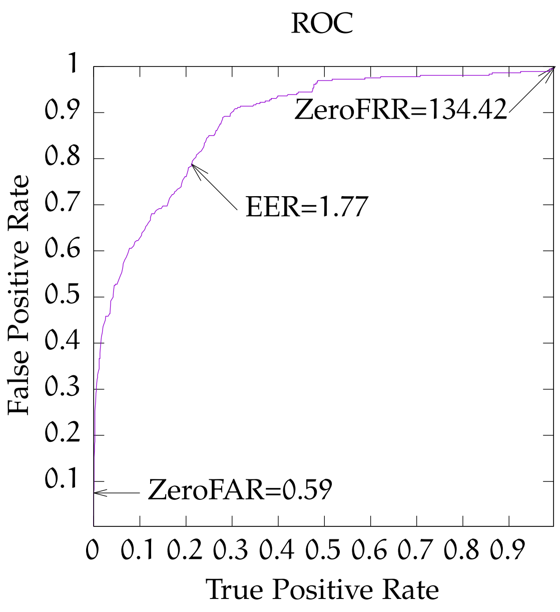


Figure 24: ROC of the statistical classifier proposed in this thesis with thresholds for ZeroFAR, EER and ZeroFRR.

6.4.1 *Thresholds*

As we described in Section 4.3.5 (Authentication), three thresholds are normally used, one for high security at 0.01 % FAR, medium security at the EER and low security at 0.01 % FRR. All of these values can be determined from the ROC, which is depicted for our system in Figure 24. Unfortunately, we don't have enough data to calculate these metrics with sufficient precision, therefore we use the next higher threshold. The *high-security threshold* should be set to 0.6, resulting in a FAR of 0.084 %. The *medium-security threshold* should be set to 1.77, resulting in an EER of 10.44 % and finally, the *low-security threshold* should be set to 133.99, resulting in a FRR of 0.28 %. High and low-security threshold both lie at the points where the first impostor attempt is allowed or the last valid user attempt is rejected.

6.4.2 *Battery Life*

We researched the maximum power consumption of the sensors we found to be most appropriate for our authentication approach defined in the Android Compatibility Definition Document [Goo15a]. By combining it with mobile phone usage data, we can get a rough estimation on how much additional power is needed. According to [Mar13], 18- to 20-year olds (the most smart phone-savy age group) sent and received 3853 text messages per month in 2013. Let's assume that for security reasons a password has to be entered beforehand every time. For the average-complexity password used in our evalua-

SAMPLING	EER
0.001	10.79 %
0.01	10.73 %
0.1	10.70 %
1	10.85 %

Table 10: EER of our system when rounding sensor data to a certain number of decimal places

tion, participants needed a mean of seven seconds to complete input. As the rotation vector used in our system combines accelerometer, magnetometer and gyroscope data, it combines energy usage of all of them and, per compatibility definition, mustn't consume more than 4 mW. This results in

$$\frac{3853 \cdot 12}{365} \cdot 7s \cdot 4mW \approx \frac{3547mWs}{3600} \approx 1mWh \tag{12}$$

Assuming a typical Li-Ion battery with 3.7 V and a capacity of 3000 mAh, the sensors need

$$\frac{1mWh}{3700mV} = 0.00027Ah \cdot 1000 = 0.27mAh \tag{13}$$

The battery therefore discharges at an additional hourly rate of

$$\frac{0.27}{3000}mAh = 0.009\% \tag{14}$$

Given the added security and the fact that other user activities like playing games or simply turning the screen on need more power, this is negligible.

6.4.3 *Data Accuracy*

Another interesting topic is how sensor resolution and timing influences recognition performance. As shown in Section 4.2 (Sensor Data), Android sensors are allowed a certain tolerance both in reporting timing and in measurement accuracy. Killourhy and Maxion [KM08] studied the influence of reduced clock timing on digraphs. Windows XP has a clock resolution of 15.625 ms, while the X11 server under UNIX delivers keystroke time stamps with a 10 ms resolution. They then compared the Windows XP clock with a high-resolution clock of 1 ms and multiple derived resolutions between 20 ms and 20 seconds. They showed that while resolutions up to 15 ms only have marginal influence (4.2 % higher EER), it starts increasing dramatically after 50 ms.

We replicate this experiment and apply it to sensors by sampling the data captured by the participants with 0.001, 0.01, 0.1 and whole unit steps. Then, we calculate the EER again and see whether it increases. Table 10 shows the results. Interestingly, while rounding it to three positions after decimal point increases the EER by 0.35 %, performance gets better when rounding more until it gets worse again in the most extreme case of whole number rounding. The results don't worsen as much as with Killourhy and Maxion, but considering that using no sensor data at all already achieves an EER of 10.88 %, this is a huge difference.

7 SUMMARY

In this thesis we proposed a novel sensor-enhanced keystroke dynamics authentication system for mobile devices. We summarize the approach and our findings in this chapter and talk about future work that could be done to improve it.

7.1 SUMMING UP

After giving a general introduction to biometrics in Chapter 1, we sketched the important components along the way of samples though the system. We established that soft biometrics are an individual group and, although they cannot identify but only authenticate users, security can be enhanced by them. They can filter template databases for strong biometrics and add additional features to the system for cases where it is unsure about the decision. Also, it's possible to use them stand-alone in situations where strong biometrics are unsuitable, for example when no user interaction is feasible. We summarized the types of biometrics that can be used on mobile devices: fingerprint, hand gesture, keystroke dynamics, signature, voice, face and iris. By analyzing the classification methods, whether they need impostor patterns and if authentication can be carried out implicitly, we put the stated EERs into perspective. Our motivation for the approach was that already in the 19th century, telegraph operators could identify colleagues by their typing rhythm and keystroke authentication systems have subsequently been built for computer keyboards. However, as there is comparably little research regarding touch keyboards and even less of them use sensors, which are available on any smart phone today, we investigated a novel combination of features. Our goal was to undercut the EER of 13.59 % that was reached by Maiorana et al. [Mai+11] using statistical methods, whose approach we based our work on. It was obvious, however, that we would not fulfill the criteria of the European Standard *EN 50133-1* for secure access control systems, which mandates a FAR of 0.001 % and a FRR of 1 %, as soft biometrics generally aren't designed to be used on their own in high-security environments.

In Chapter 2 we described how keystroke authentication systems can enhance password authentication by providing an extra security layer after the correct password has been entered. We mentioned the stages such a system has, that enrollment shouldn't take more than ten samples to avoid annoying the user [Ara+05] and that analysis can either be performed in a static or a dynamic way. The difference is that static analysis is performed on a pre-specified text while dynamic analysis works with any textual inputs. As this is more difficult and results in worse recognition performance, we focused on static analysis. After describing the traditional keystroke timing information used in most systems – hold time and digraph – we mentioned features unique to touch screens, for example finger orientation and size as well as the position on the key. Following that we did an analysis of the sensors available on mobile devices. We found that motion sensors are best suited for our purpose. Additionally, we investigated different approaches for classification, starting from statistical methods and neural networks over pattern recognition and learning to hybrid techniques. We stated the advantages and disadvantages and decided to use statistics because of their resource-effectiveness and ability to work without impostor patterns.

By analyzing related work in Chapter 3, we learned valuable lessons for our system's design. We categorized the papers into the three types: computer, cell phone and touch screen keyboards. For every system, recognition performance is given to compare them and get a feeling for what can be expected. Finally, we summarized our findings:

- hold time and digraph are the most promising timing features,
- static thresholds for all users aren't bad for performance in all cases,
- keystroke data isn't necessarily portable between devices and
- methods used for keystroke authentication can also be applied to graphical approaches with images and patterns.

Finally, we noticed that studies normally are carried out in laboratories, where captured data will likely differ from usage in real life. We therefore decided to base our evaluation on real usage data.

As our system was programmed on the Android platform using Java, we introduced the theoretical concepts behind it in Chapter 4. We talked about the features we used and how many values can be extracted from one sample. We stated that we would use the accelerometer, gravity, linear acceleration, rotation vector and calibrated as well as uncalibrated gyroscope as the sensors used in authentication. Additionally, we described the minimum requirements Android imposes on the hardware that captures these sensors' measurements and estimated that the accuracy is sufficient. Finally, we introduced the methods our classification approach uses to

- calculate the distance between samples (Manhattan and Squared Euclidean distance)
- determine the variability of the acquisitions by calculating the mean distance to certain other ones (nearest and farthes neighbour, all others or a template keystroke dynamics)

- select the best templates from enrollment to enhance authentication (MDIST and GMMS as well as DEND and fuzzy C-means) and finally
- authenticate users by calculating a single score value taking into account the variability metrics from before.

For implementation on the Android operating system in Chapter 5, we listed the modifications to the *Google Keyboard* to integrate our biometric authentication approach. We introduced the central instance, the Biometrics Manager, how it collects sensor data and communicates with the classifier. The lifecycle of both classes and the database structure that holds metadata and the captured acquisitions was depicted. We proposed the Biometrics Manager API, that can be used by client apps to request an authentication result, state a biometric context for sharing templates between apps and clear data when the password changes. After that, we documented the inner workings of the statistical classifier we implemented and how it differs from the capture classifier later used for collecting data in our user study. We also showed the example client included in our project and its user interface. Finally, we integrated our approach into the CORMORANT framework [Hin+15] which integrates multiple biometrics and provides a central API for client applications to used. Also, we sketched possible security flaws in our system, like intercepting authentication requests via the Biometrics Manager API and abusing stored template information.

In the last Chapter 6 we described the methodical approach to evaluating our system. We reiterated the importance for us to use real-world data in the experiments and that it complicated data acquisition, because of which we only had four participants in our study. We mentioned how interaction with the system in study mode works and that captured input data is enhanced with manually entered input method and situation information. We also added a control group to evaluate whether the templates are portable between devices. Furthermore, we analyzed properties of the collected raw data and stated that we had to redo the study because of a bug in providing sensor data to the classifier. We found that half of the participants followed instructions to regularly enter the given password badly, resulting in batches of data. However, we could also show that clusters can be seen when the input method and situation are taken into account and that most passwords were entered using either the right or both hands and typing with the thumbs. After that, we iteratively optimized our system by finding out which combination of the methods described in the concept works best. By combining the Squared Euclidean distance with minimum variability metric, an acquisition set size of ten and selecting six templates using the GMMS-MAX algorithm, we could obtain a minimum EER of 10.44 % when using rotation vector and gravity sensor data. However, as using no sensor data at all already resulted in an EER of 10.88 %, we concluded that calculating the relative measurements between key press and release doesn't provide enough discriminatory information and further research is needed regarding other methods. We also showed that the keystroke templates indeed aren't portable between devices. Finally, we determine the optimum thresholds in the classifier to be 0.6 for high-security environments, resulting in a FAR of 0.084 %, 1.77 for medium-security environments, resulting in an EER of 10.44 % and 133.99 for low-security environments, resulting in a FRR of 0.28 %. By calculating the power needed for capturing sensor data, we showed that the influence on battery life is marginal. We also investigated how accuracy of sensor data affects our recognition performance and concluded that it worsens EERs by 0.41 % in the worst case, which is above using no sensor data at all.

7.2 FUTURE WORK

A Master Thesis is naturally time-limited to ensure the student is finished at some point. Biometric methods, however, are a field of study where additional work always is possible. If one approach doesn't yield the expected result, there are plenty of others to try. If one feature doesn't bring the expected success, another one can be used instead, and so on. This is also true for the thesis on hand. As we showed in Section 6.3.5 (Sensor Selection), using sensors to enhance keystroke authentication does enhance performance, but the way we implemented it isn't the best. Therefore, primary future work should focus on different ways of incorporating sensor measurements. It might be beneficiary to use sliding windows, as Giuffrida et al. [Giu+14] proposed, or to capture more measurements per keystroke. As we calculated in Section 4.2 (Sensor Data), an average of 15 values could be used instead of the single one in this approach.

We currently differentiate template sets according to screen resolution, but there potentially need to be more of them. An example is when users have different ways of entering the same password in certain situations, using secondary keys one time and symbol keyboards another. This changes the amount of keystrokes and isn't considered in the current implementation. Any deviation in sample size is treated as an impostor attack. Of course, adding more template sets would also result in longer enrollment times and the question remains, whether new template sets should still be allowed after a prolonged amount of time, for example, because the user varies their behavior.

This is also related to a similar problem: in our approach, we did a one-time feature selection, but it's possible that user behavior changes over time. The selection process can be executed offline while nobody uses the keyboard. Therefore, regular updates of the template

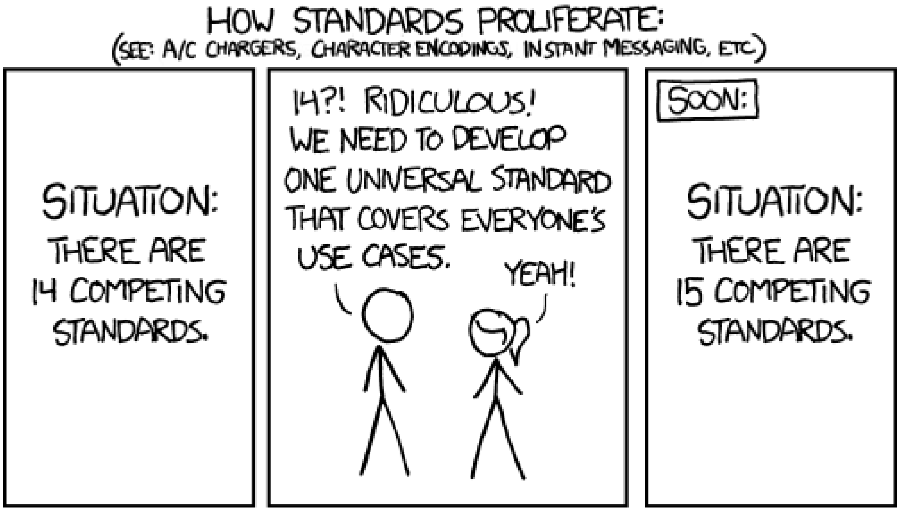

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Figure 25: Comic illustrating the consequences of trying to introduce new standards because existing ones don't seem to fit [Mun11]

set could on the one hand improve recognition performance and on the other hand adapt to behavioral changes over time.

For real-world systems, the keyboard UI would have to be improved as well. A first-time wizard should introduce the user to the additional features the keyboard provides and how they can use it. This could take place similarly to the setup wizard shown after installing the app. Additionally, there should be biometric settings where the user can check which apps have data stored and clear it on demand. An enhanced biometrics policy could ask every time a new app wants to perform biometric authentication, at least when shared biometric contexts are used and the templates can potentially be abused.

Another open decision is the API choice. On the one hand, in Section 5.2 (Biometrics Manager API), we proposed our own method for accessing authentication results from client apps. This, however, has the consequence that potentially few apps will use it, because integration is an additional effort with little gain. This can also be seen with additional hardware specific to single device models, such as secondary screens, or even standards, as Figure 25 illustrates. The more future-proof decision would therefore be to rely on the CORMORANT framework, which by far offers a greater amount of features and therefore more incentives to integrate it.

Finally, the system could be ported to other operating systems like Apple's iOS and considerations regarding secure template storage should be made. Both aspects are described below.

7.2.1 *Apple iOS*

Before the release of iOS 8 in 2014, no custom keyboards could be integrated to Apple's mobile devices because of security concerns. As this has changed since then, a port would be possible from a purely practical point of view. Of course, as apps have to be written in Objective C or Swift there, this would require huge amounts of work. To make it easier, the base keyboard could again be an open-source project already designed for the specific operating system, such as *Slidden* [BG14].

iOS also offers different sensors than Android, based on accelerometer and gyroscope: in addition to the raw readings, a *device motion* fused sensor provides attitude, rotation rate (= rotation vector), acceleration (= linear acceleration) and magnetic field data. Another evaluation would need to find out whether the data these sensors deliver is comparable to those in Android.

Sensor update frequency can be defined in iOS up to 100 Hz or 10 ms. This would be high enough, but unfortunately, Apple doesn't seem to release information about the individual sensor's frequency, resolution and standard deviation. Therefore, a comparison to Android regarding the expected accuracy is not possible.

7.2.2 *Android's Trusted Execution Environment*

With Android 6.0, Google added a hardware-backed keystore to the framework. Using the Android Keystore system, access to a Trusted Execution Environment (TEE) running on a separate hardware chip is possible. It prevents unauthorized extraction of key data stored therein and mitigates unauthorized access [Goo14a]. Unfortunately, this only works with cryptographic operations, but it could at least be used to securely encrypt and decrypt the templates in the database and therefore prevent abuse.

However, in the same version, user-authentication-gated cryptographic keys were added [Goo15b]. This allows accessing securely stored keys through authentication tokens generated either via PIN, pattern or password authentication (*Gatekeeper* component) or via fingerprint (*Fingerprint* component). Both Gatekeeper and Fingerprint only send the inputs/samples to a counterpart in the TEE, which securely compares them and generates the token. No secrets ever leave the secure environment. Unfortunately, this system is not extendable. In the TEE runs an operating system called *Trusty* [Goo16b], which is encapsulated and only communicates via Inter-process Communication (IPC). Unfortunately, all Trusty programs are developed by Google, which also builds the environment and cryptographically signs it with their key. This signature is then verified by the device bootloader. No third-party programs are supported in the architecture.

In conclusion, Google already included secure biometric key and template storage for fingerprints. Unfortunately, as long as they don't add additional biometrics, we cannot use this conveniently secure environment. The decision to deny developer access is understandable, as every additional program running in the TEE increases attack surface and unrestricted access would open the system to malicious code.

Part III

APPENDIX

A ANDROID SENSORS OVERVIEW

This appendix contains an overview of the sensors available in Android, whether they are based on hardware and how they are commonly used.

SENSOR	TYPE	DESCRIPTION	COMMON USES
Accelerometer	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
Ambient Temperature	Hardware	Measures the ambient room temperature in degrees Celsius (°C).	Monitoring air temperatures.
Gravity	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all,three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
Gyroscope	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
Light	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.

SENSOR	TYPE	DESCRIPTION	COMMON USES
Linear Acceleration	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
Magnetic Field	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
Orientation	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).	Determining device position.
Pressure	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
Proximity	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
Relative Humidity	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.

SENSOR	TYPE	DESCRIPTION	COMMON USES
Rotation Vector	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
Temperature	Hardware	Measures the temperature of the device in degrees Celsius (°C). Replaced with the *Ambient Temperature* sensor in Android 4.0.	Monitoring temperatures.

Table 11: An overview of all sensors available on the Android platform [Goo12]

B NOTATION

We use a consistent naming scheme throughout the thesis to ensure that every formula and algorithm is as understandable as possible. Through our research we found it helpful to have a concise overview at all the conventions. The notation here therefore should be considered as a kind of "dictionary".

VARIABLES

E Number of template acquisitions selected from the enrollment acquisitions

e A template acquisition

K Number of values (keystrokes) of an acquisition, shown in Table 2

k A value (keystroke)

N Number of elements, shown in Table 2

n An element

u A legitimate user

$\tilde{u}$ A user to be authenticated

COLLECTIONS

Δ The set of features, shown in Table 2

δ One feature contained in feature set Δ

$D_{N \times N}$ Matrix of pairwise distance scores between all acquisitions

$\mathbf{f}_u$ Feature vector containing all features δ of user u

$\mathbf{f}^{\delta}_{u,e}$ One acquisition of feature δ for user u

$f^{\delta}_{u,e}$ One N-dimensional value (keystroke) in an acquisition of feature δ

T Set of selected templates

t_u Index of the template keystroke dynamics of a feature, which has the minimum average distance to all others

FUNCTIONS

$D(\cdot,\cdot)$ Distance function

$D_M(\cdot,\cdot)$ Manhattan (city block) distance

$D_E(\cdot,\cdot)$ Squared Euclidean distance

$\mathcal{I}(V_e)$ Index of element e in vector V

$\mathcal{P}(\Delta)$ Powerset of set Δ

$AUTH_{\tilde{u}}$ Authentication metric

MAX_u^δ Mean value of the distance of one acquisition to its farthest neighbor

$MAX_{\tilde{u}}$ Authentication metric based on MAX_u^δ

$MEAN_u^\delta$ Mean value of all distances between all acquisitions

$MEAN_{\tilde{u}}$ Authentication metric based on $MEAN_u^\delta$

MIN_u^δ Mean value of the distance of one acquisition to its nearest neighbor

$MIN_{\tilde{u}}$ Authentication metric based on MIN_u^δ

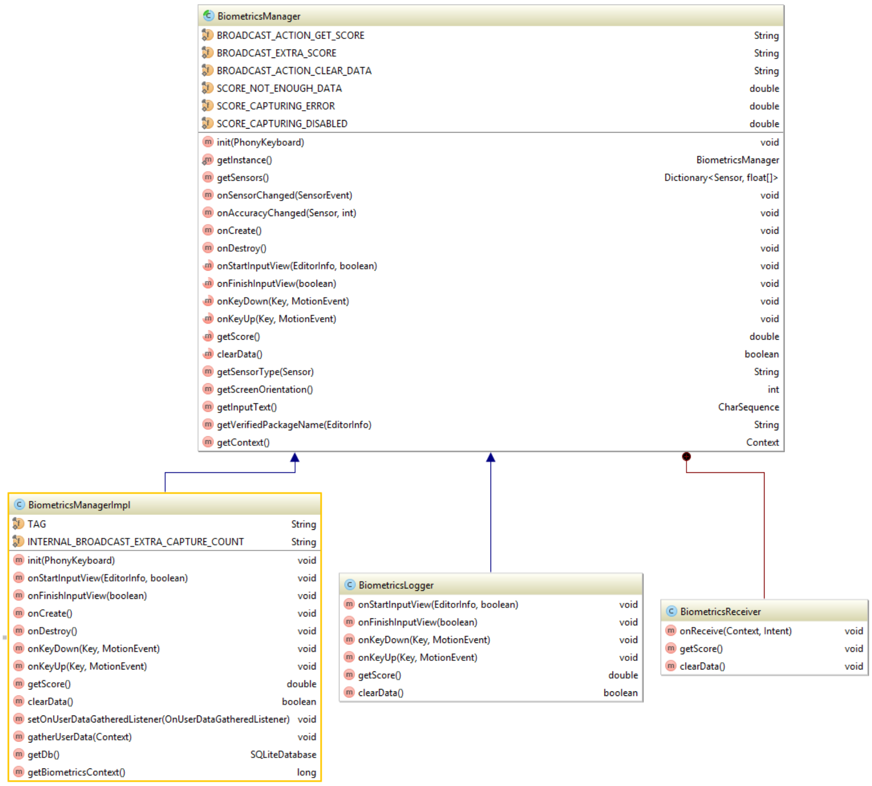
$TEMP_u^\delta$ Mean value of all distances of all acquisitions to the template keystroke dynamics with index t_u, which has the minimum average distance to all others for this feature

$TEMP_{\tilde{u}}$ Authentication metric based on $TEMP_u^\delta$

VAR_u^δ Variability metric

C
CLASS DIAGRAMS

This appendix contains the class diagrams of the most interesting classes in the *PhonyKeyboard* app. They were too large to be included in the main text without impeding readability, but they are referenced in the text where necessary.



BiometricsManager	
BROADCAST_ACTION_GET_SCORE	String
BROADCAST_EXTRA_SCORE	String
BROADCAST_ACTION_CLEAR_DATA	String
SCORE_NOT_ENOUGH_DATA	double
SCORE_CAPTURING_ERROR	double
SCORE_CAPTURING_DISABLED	double
init(PhonyKeyboard)	void
getInstance()	BiometricsManager
getSensors()	Dictionary<Sensor, float[]>
onSensorChanged(SensorEvent)	void
onAccuracyChanged(Sensor, int)	void
onCreate()	void
onDestroy()	void
onStartInputView(EditorInfo, boolean)	void
onFinishInputView(boolean)	void
onKeyDown(Key, MotionEvent)	void
onKeyUp(Key, MotionEvent)	void
getScore()	double
clearData()	boolean
getSensorType(Sensor)	String
getScreenOrientation()	int
getInputText()	CharSequence
getVerifiedPackageName(EditorInfo)	String
getContext()	Context

BiometricsManagerImpl	
TAG	String
INTERNAL_BROADCAST_EXTRA_CAPTURE_COUNT	String
init(PhonyKeyboard)	void
onStartInputView(EditorInfo, boolean)	void
onFinishInputView(boolean)	void
onCreate()	void
onDestroy()	void
onKeyDown(Key, MotionEvent)	void
onKeyUp(Key, MotionEvent)	void
getScore()	double
clearData()	boolean
setOnUserDataGatheredListener(OnUserDataGatheredListener)	void
gatherUserData(Context)	void
getDb()	SQLiteDatabase
getBiometricsContext()	long

BiometricsLogger	
onStartInputView(EditorInfo, boolean)	void
onFinishInputView(boolean)	void
onKeyDown(Key, MotionEvent)	void
onKeyUp(Key, MotionEvent)	void
getScore()	double
clearData()	boolean

BiometricsReceiver	
onReceive(Context, Intent)	void
getScore()	void
clearData()	void

Figure 26: The `BiometricsManager` class hierarchy

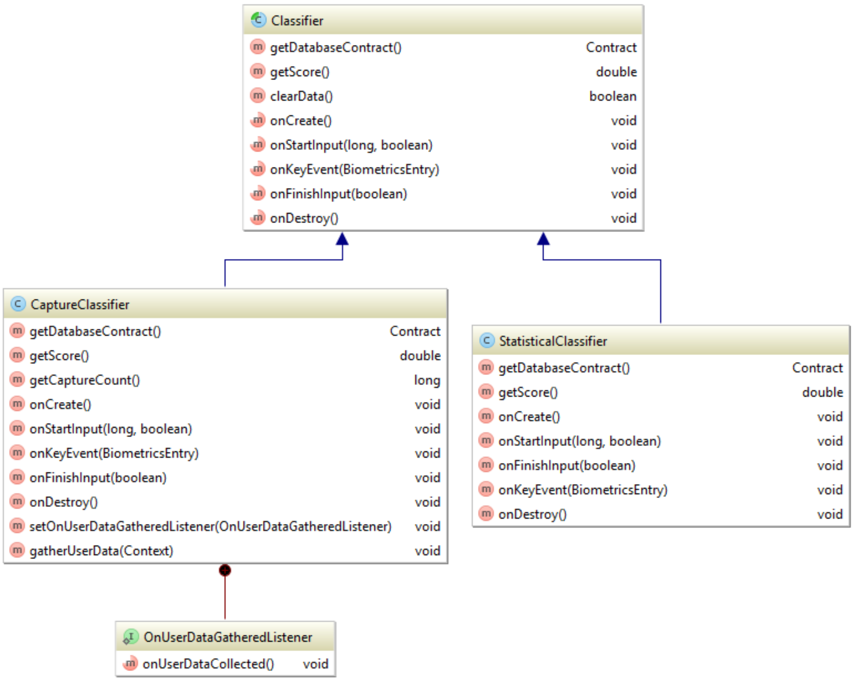

Classifier
getDatabaseContract() Contract
getScore() double
clearData() boolean
onCreate() void
onStartInput(long, boolean) void
onKeyEvent(BiometricsEntry) void
onFinishInput(boolean) void
onDestroy() void
CaptureClassifier
getDatabaseContract() Contract
getScore() double
getCaptureCount() long
onCreate() void
onStartInput(long, boolean) void
onKeyEvent(BiometricsEntry) void
onFinishInput(boolean) void
onDestroy() void
setOnUserDataGatheredListener(OnUserDataGatheredListener) void
gatherUserData(Context) void
StatisticalClassifier
getDatabaseContract() Contract
getScore() double
onCreate() void
onStartInput(long, boolean) void
onFinishInput(boolean) void
onKeyEvent(BiometricsEntry) void
onDestroy() void
OnUserDataGatheredListener
onUserDataCollected() void

Figure 27: The `Classifier` class hierarchy

D CORMORANT INTEGRATION

For integrating our project into the CORMORANT framework, we had to make a few adjustments to our code. These snippets are included in this chapter.

```
<manifest package="at.jku.fim.phonykeyboard.latin"
   xmlns:android="http://schemas.android.com/apk/res/android">
   ...
   <!-- permission declaration -->
   <uses-permission android:name="at.usmile.cormorant.
       REGISTER_AUTH_PLUGIN" />
   ...
   <!-- service declaration -->
   <service
      android:name=".biometrics.CormorantAuthenticationService"
      android:enabled="true"
      android:exported="true"
      android:permission="at.usmile.cormorant.permission.
          READ_PLUGIN_DATA">
      <meta-data
         android:name="apiVersion"
         android:value="1" />
      <meta-data
         android:name="pluginType"
         android:value="confidence" />
      <meta-data
         android:name="title"
         android:value="PhonyKeyboard" />
      <meta-data
         android:name="description"
         android:value="Sensor—enhanced keystroke dynamics
             authentication" />
      <meta-data
         android:name="implicit"
         android:value="true" />
   </service>
   ...
</manifest>
```

Listing 3: The code snippets added to AndroidManifest.xml for integrating into the CORMORANT framework

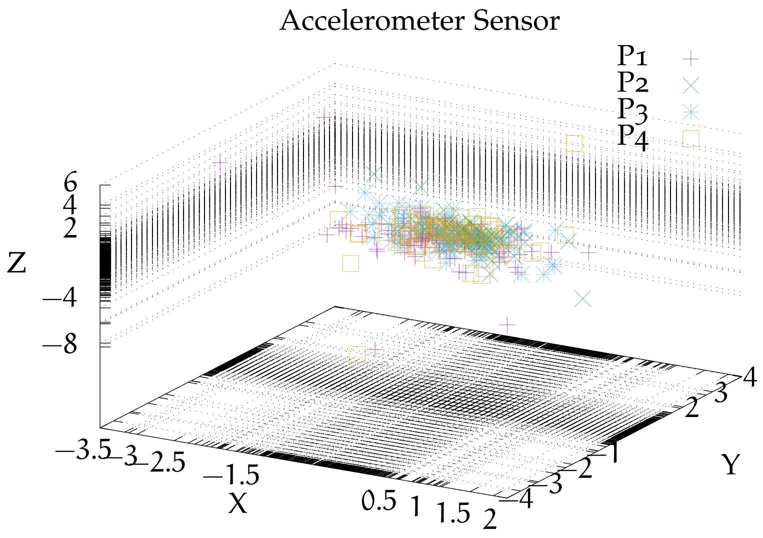
```
public class CormorantAuthenticationService extends
    AbstractConfidenceService {
   @Override
   protected void onDataUpdateRequest() {
      BiometricsManager manager;
      try {
         manager = BiometricsManager.getInstance();
      } catch (IllegalStateException e) {
         manager = null;
      }
      double score = manager != null ? manager.getScore() :
          BiometricsManager.SCORE_CAPTURING_DISABLED;
      publishConfidenceUpdate(score);
   }

   protected void publishConfidenceUpdate(double score) {
      publishConfidenceUpdate(this, score);
   }

   public static void publishConfidenceUpdate(
       AbstractConfidenceService service, double score) {
      StatusDataConfidence confidence = new StatusDataConfidence
          ();
      if (score < BiometricsManager.SCORE_NOT_ENOUGH_DATA) {
         confidence.status(StatusDataConfidence.Status.UNKNOWN);
      } else if (score == BiometricsManager.SCORE_NOT_ENOUGH_DATA
          ) {
         confidence.status(StatusDataConfidence.Status.TRAINING);
      } else {
         confidence.status(StatusDataConfidence.Status.
             OPERATIONAL);
      }
      confidence.confidence(score > 0 ? score : 0);
      service.publishConfidenceUpdate(confidence);
   }
}
```

Listing 4: The new CormorantAuthenticationService used for communication between the *PhonyKeyboard* app and CORMORANT

E

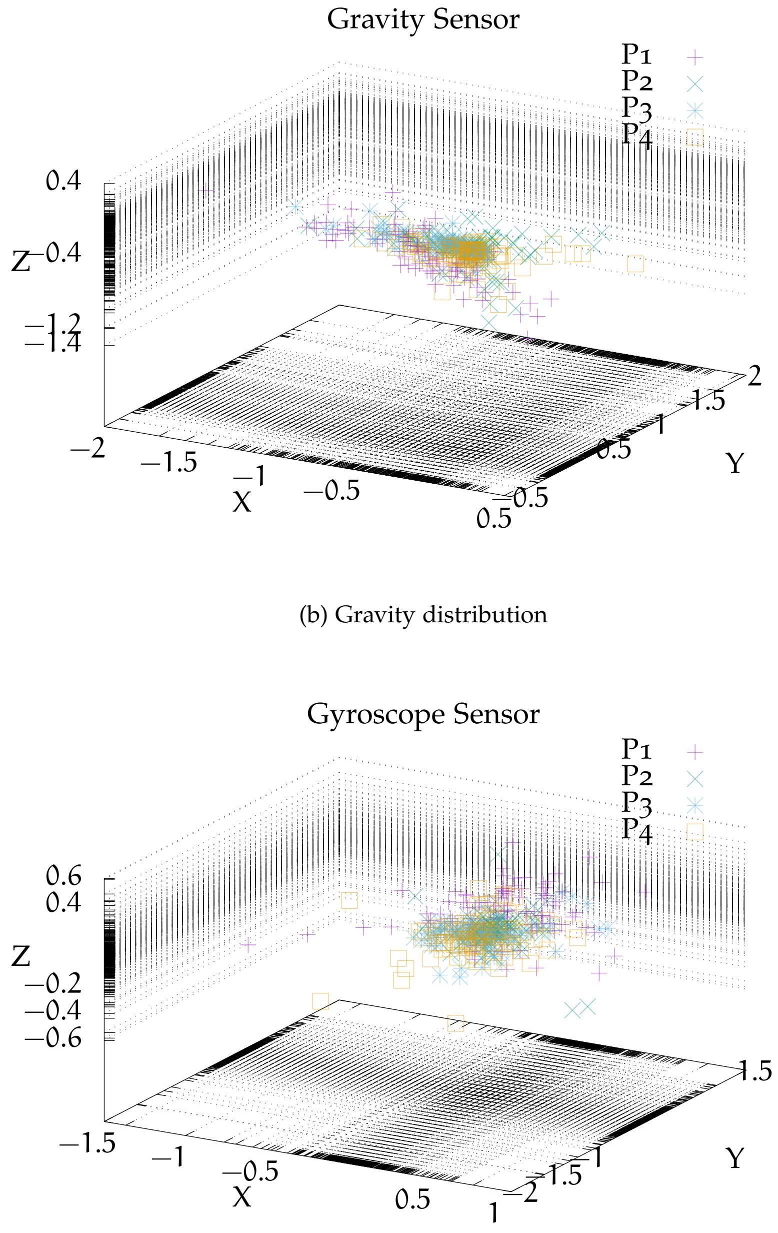
DATA GRAPHS

All graphs regarding the data we collected in our user study, which we didn't have any space for in Chapter 6 (EXPERIMENTS), are collected here.



(a) Accelerometer distribution

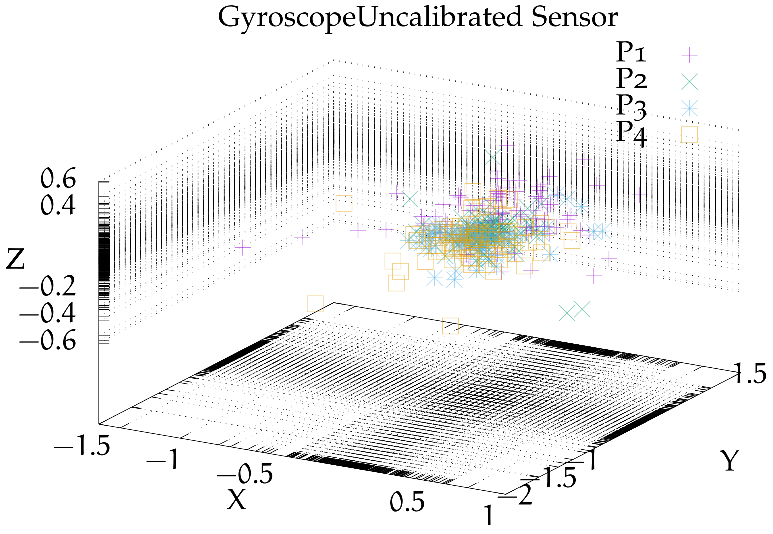
Figure 28: Measurements of the accelerometer in m/s^2 for the letter "i" from all participants



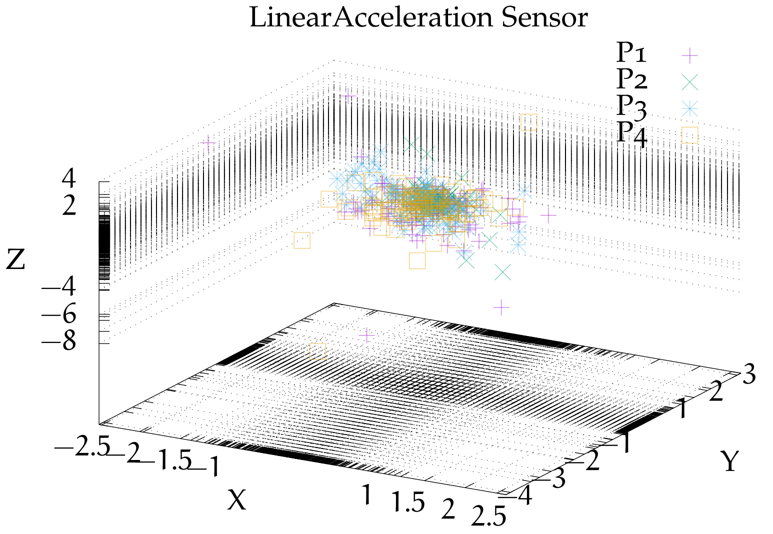
(b) Gravity distribution

(c) Gyroscope distribution

Figure 28: Continued: Measurements of the gravity sensor in m/s^2 and the gyroscope in rad/s for the letter "i" from all participants



(d) Uncalibrated gyroscope distribution



(e) Linear acceleration distribution

Figure 28: Continued: Measurements of the uncalibrated gyroscope in rad/s and the linear acceleration sensor in m/s^2 for the letter "i" from all participants

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LISTINGS

ACRONYMS

API	Application Programming Interface
AOSP	Android Open Source Project
BPNN	Back Propagation Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DEND	Agglomerative Complete Link Clustering (named after the resulting dendrograms)
EER	Equal Error Rate
FAR	False Acceptance Rate
FF-MLP	Feed-Forward Multi-Layered Perceptron
FLD	Fisher's Linear Discriminant
FRR	False Rejection Rate
FT	Flight Time (see PP)
GMMS	Greedy Maximum Match Scores
GTA	Gradual Training Algorithm
GUI	Graphical User Interface
HMM	Hidden Markov Model
HT	Key Hold Time (dwell time)
IPC	Inter-process Communication
J2ME	Java Platform, Micro Edition
MDIST	Minimum Distance algorithm
NSF	National Science Foundation

NIST	National Institute of Standards and Technology
PCA	Principal Component Analysis
PDA	Personal Digital Assistant
PIN	Personal Identification Number
PNN	Probabilistic Neural Network
PP	Press-to-Press time (digraph)
RFC	Request for Comments
RMSE	Root-Mean-Square Error
ROC	Receiver Operating Characteristic
RP	Release-to-Press time (flight time)
RR	Release-to-Release time
SQL	Structured Query Language
SVM	Support Vector Machine
TEE	Trusted Execution Environment
UI	User Interface
VM	Virtual Machine

BIBLIOGRAPHY

[AAM15] Sarah N Abdulkader, Ayman Atia, and Mostafa-Sami M Mostafa. "Authentication systems: principles and threats." In: *Computer and Information Science* 8.3 (July 2015), p. 155. DOI: 10.5539/cis.v8n3p155. URL: http://www.ccsenet.org/journal/index.php/cis/article/view/47833.

[Ali+12] P.N. Ali Fahmi, Elyor Kodirov, Deok-Jai Choi, Guee-Sang Lee, A Mohd Fikri Azli, and Shohel Sayeed. "Implicit authentication based on ear shape biometrics using smartphone camera during a call." In: *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Oct. 2012, pp. 2272–2276. DOI: 10.1109/ICSMC.2012.6378079. URL: http://www.wellaware1.com/docs/earbiometrics_cell%2006378079.pdf.

[AB14] Hamdan Alzahrani and Terrance E. Boult. "Remote authentication using vaulted fingerprint verification." In: ed. by Ioannis A. Kakadiaris, Walter J. Scheirer, and Christoph Busch. International Society for Optics and Photonics, May 2014, 90750K. DOI: 10.1117/12.2053126. URL: https://digitool.library.colostate.edu/bitstream/handle/10976/166582/Alzahrani_uccs_0892D_10169.pdf?sequence=1&isAllowed=y.

[Amb05] Parvathi D. Ambalakat. "Security of Biometric Authentication Systems." In: *21st Computer Science Seminar*. Hartford, CT, USA, 2005, pp. 1–7.

[Apa16] Apache Software Foundation. *Commons Math: The Apache Commons Mathematics Library*. 2016. URL: https://commons.apache.org/proper/commons-math/.

[Ara+05] Lívia C. F. Araújo, Luiz H. R. Jr. Sucupira, Miguel G. Lizárraga, Lee L. Ling, and João B. T. Yabu-Uti. "User authentication through typing biometrics features." In: *IEEE Transactions on Signal Processing* 53.2 (2005), pp. 851–855. DOI: 10.1109/TSP.2004.839903. URL: http://www.cmlab.csie.ntu.edu.tw/~ipr/ipr2006/data/misc/01381785.pdf.

[BCR12] M. Baloul, E. Cherrier, and C. Rosenberger. "Challenge-based speaker recognition for mobile authentication." In: *nternational Conference of the Biometrics Special Interest Group (BIOSIG)*. IEEE, 2012, pp. 1–7. URL: https://hal.archives-ouvertes.fr/file/index/docid/998932/filename/acti-baloul-2012-1.pdf.

[BW12] Salil Partha Banerjee and Damon L. Woodard. "Biometric Authentication and Identification Using Keystroke Dynamics: A Survey." In: *Journal of Pattern Recognition Research* 7.1 (2012), pp. 116–139. DOI: 10.13176/11.427. URL: http://www.jprr.org/index.php/jprr/article/view/427/167.

[BD15] Lars Behnke and Michel Daviot. *hierarchical-clustering-java*. 2015. URL: https://github.com/lbehnke/hierarchical-clustering-java/.

[BGP02] Francesco Bergadano, Daniele Gunetti, and Claudia Picardi. "User authentication through keystroke dynamics." In: *ACM Transactions on Information and System Security* 5.4 (Nov. 2002), pp. 367–397. DOI: 10.1145/581271.581272. URL: http://www.di.unito.it/~gunetti/curriculum/p367-tissec-1.pdf.

[Bez81] James C. Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Plenum Press, 1981, p. 256.

[Bor+11] Mohamed Anouar Borg, Salwa Said, Boulbaba Ben Amor, and Chokri Ben Amar. "Information fusion for 3D face recognition." In: *2011 International Conference on Image Information Processing*. IEEE, Nov. 2011, pp. 1–6. DOI: 10.1109/ICIIP.2011.6108980.

[BG14] Daniel Brim and Ivan Grachyov. *Slidden: An open source, customizable, iOS 8 keyboard.* 2014. URL: https://github.com/Brimizer/Slidden.

[Cam+09] P. Campisi, E. Maiorana, M. Lo Bosco, and A. Neri. "User authentication using keystroke dynamics for cellular phones." In: *IET Signal Processing* 3.4 (2009), p. 333. DOI: 10.1049/iet-spr.2008.0171. URL: http://www.comlab.uniroma3.it/Campisi/Pub_PDF/CamMaiNerIET09.pdf.

[CEN96] CENELEC. *EN 50133-1:1996; Alarm systems. Access control systems for use in security applications. Part 1: System requirements*. 1996.

[CTL12] Ting-Yi Chang, Cheng-Jung Tsai, and Jyun-Hao Lin. "A graphical-based password keystroke dynamic authentication system for touch screen handheld mobile devices." In: *Journal of Systems and Software* 85.5 (2012), pp. 1157–1165. DOI: 10.1016/j.jss.2011.12.044.

[CF07] N. L. Clarke and S. M. Furnell. "Advanced user authentication for mobile devices." In: *Computers & Security* 26.2 (2007), pp. 109–119. DOI: 10.1016/j.cose.2006.08.008. URL: http://pearl.plymouth.ac.uk/bitstream/handle/10026.1/1101/CKARKE%20NL_2004.pdf.

[CBT99] O. Coltell, J.M. Badfa, and G. Torres. "Biometric identification system based on keyboard filtering." In: *Proceedings IEEE 33rd Annual 1999 International Carnahan Conference on Security Technology (Cat. No.99CH36303)*. IEEE, 1999, pp. 203–209. DOI: 10.1109/CCST.1999.797915. URL: http://gtorres.forofue.com/docs/carnahan99.pdf.

[Dau03] John Daugman. "The importance of being random: statistical principles of iris recognition." In: *Pattern Recognition* 36.2 (2003), pp. 279–291. DOI: 10.1016/S0031-3203(02)00030-4.

[Giu+14] Cristiano Giuffrida, Kamil Majdanik, Mauro Conti, and Herbert Bos. "I Sensed It Was You: Authenticating Mobile Users with Sensor-Enhanced Keystroke Dynamics." In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, 2014, pp. 92–111. DOI: 10.1007/978-3-319-08509-8_6. URL: http://dimva2014.isg.rhul.ac.uk/slides/Giuffrida-dimva-2014-talk.pdf.

[Goo09] Google. *MotionEvent | Android Developers*. 2009. URL: https://developer.android.com/reference/android/view/MotionEvent.html (visited on 2016-07-29).

[Goo12] Google. *Sensors Overview | Android Developers*. 2012. URL: https://developer.android.com/guide/topics/sensors/sensors_overview.html (visited on 2016-05-06).

[Goo14a] Google. *Android Keystore System | Android Developers*. 2014. URL: https://developer.android.com/training/articles/keystore.html.

[Goo14b] Google. *LatinIME*. 2014. URL: https://android.googlesource.com/platform/packages/inputmethods/LatinIME/ (visited on 2016-07-29).

[Goo14c] Google. *Touch Devices | Android Open Source Project*. 2014. URL: https://source.android.com/devices/input/touch-devices.html#pressure-field (visited on 2016-05-05).

[Goo15a] Google. *Android 6.0 Compatibility Definition Document | Android Open Source Project*. 2015. URL: http://source.android.com/compatibility/android-cdd.html#7_3_sensors (visited on 2016-07-11).

[Goo15b] Google. *Authentication | Android Open Source Project*. 2015. URL: https://source.android.com/security/authentication/index.html (visited on 2016-07-29).

[Goo16a] Google. *Platform Versions | Android Developers*. 2016. URL: https://developer.android.com/about/dashboards/index.html#Platform (visited on 2016-07-20).

[Goo16b] Google. *Trusty TEE | Android Open Source Project*. 2016. URL: https://source.android.com/security/trusty/index.html (visited on 2016-07-29).

[Gri08] Griaule Biometrics. *Equal Error Rate (EER) | Griaule Biometrics*. 2008. URL: http://www.griaulebiometrics.com/en-us/book/understanding-biometrics/evaluation/accuracy/matching/interest/equal (visited on 2016-06-28).

[Hin+15] Daniel Hintze, Rainhard D. Findling, Muhammad Muaaz, Eckhard Koch, and René Mayrhofer. "Cormorant: towards continuous risk-aware multi-modal cross-device authentication." In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers - UbiComp '15*. New York, New York, USA: ACM Press, 2015, pp. 169–172. DOI: 10.1145/2800835.2800906. URL: https://usmile.at/sites/default/files/publications/Hintze_15_CORMORANT:TowardsContinuous_posterpaper.pdf.

[Ira+14] Vahab Iranmanesh, Sharifah Mumtazah Syed Ahmad, Wan Azizun Wan Adnan, Salman Yussof, Olasimbo Ayodeji Arigbabu, and Fahad Layth Malallah. "Online Handwritten Signature Verification Using Neural Network Classifier Based on Principal Component Analysis." In: *The Scientific World Journal* 2014 (2014), pp. 1–8. DOI: 10.1155/2014/381469. URL: http://downloads.hindawi.com/journals/tswj/2014/381469.pdf.

[JDN04] Anil K. Jain, Sarat C. Dass, and Karthik Nandakumar. "Can soft biometric traits assist user recognition?" In: *Proceedings of SPIE* 5404 (2004), pp. 561–572. DOI: 10.1117/12.542890. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.2055.

[JOT12] Je-Hyoung Jeon, Beom-Seok Oh, and Kar-Ann Toh. "A system for hand gesture based signature recognition." In: *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*. IEEE, Dec. 2012, pp. 171–175. DOI: 10.1109/ICARCV.2012.6485153.

[Joh75] Gunnar Johansson. "Visual Motion Perception." In: *Scientific American* 232.6 (June 1975), pp. 76–88. DOI: 10.1038/scientificamerican0675-76.

[Kap14] Philipp Kapfer. *Soft Biometrics*. Tech. rep. Linz: Department of Computational Perception, JKU Linz, 2014, pp. 1–20. URL: https://cloud.kasimir-wg.eu/f/3470a8ca83/?raw=1.

[KAK11] M. Karnan, M. Akila, and N. Krishnaraj. "Biometric personal authentication using keystroke dynamics: A review." In: *Applied Soft Computing* 11.2 (2011), pp. 1565–1573. DOI: 10.1016/j.asoc.2010.08.003.

[KM08] Kevin Killourhy and Roy Maxion. "The Effect of Clock Resolution on Keystroke Dynamics." In: *Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 331–350. DOI: 10.1007/978-3-540-87403-4_18. URL: https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/maxion/pubs/KillourhyMaxion08a.pdf.

[Li+08] Yong Li, Jianping Yin, En Zhu, Chunfeng Hu, and Hui Chen. "Score Based Biometric Template Selection and Update." In: *2008 Second International Conference on Future Generation Communication and Networking*. Vol. 3. IEEE, Dec. 2008, pp. 35–40. DOI: 10.1109/FGCN.2008.208.

[Lin97] Daw-Tung Lin. "Computer-access authentication with neural network based keystroke identity verification." In: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 1. IEEE, 1997, pp. 174–178. DOI: 10.1109/ICNN.1997.611659.

[LLC14] Jin Liu, Ting ting Liu, and Bin ru Chen. "A Novel Iris Verification System Based on Feature Extraction." In: *Proceedings of the 2013 International Conference on Electrical and Information Technologies for Rail Transportation (EITRT2013)-Volume I*. Springer Berlin Heidelberg, 2014, pp. 441–449. DOI: 10.1007/978-3-642-53778-3_43.

[Mai+11] Emanuele Maiorana, Patrizio Campisi, Noelia González-Carballo, and Alessandro Neri. "Keystroke dynamics authentication for mobile phones." In: *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*. New York, New York, USA: ACM Press, 2011, p. 21. DOI: 10.1145/1982185.1982190. URL: http://www.cs.uml.edu/~xinwenfu/Classes/91.661.201/MobilePhoneSecurityPapers/11.Mariorana.SAC.Keystroke%20dynamics%20authentication%20for%20mobile%20phones.pdf.

[Mar13] MarketingCharts. *18-24-Year-Old Smartphone Owners Send and Receive Almost 4K Texts per Month*. 2013. URL: http://www.marketingcharts.com/online/18-24-year-old-smartphone-owners-send-and-receive-almost-4k-texts-per-month-27993/ (visited on 2016-07-27).

[Met15] Max Metzger. *Vulnerability found in McAfee, Kaspersky and AVG anti-virus softwares*. 2015. URL: http://www.scmagazineuk.com/vulnerability-found-in-mcafee-kaspersky-and-avg-anti-virus-softwares/article/458993/ (visited on 2016-07-24).

[MR00] Fabian Monrose and Aviel D. Rubin. “Keystroke dynamics as a biometric for authentication.” In: *Future Generation Computer Systems* 16.4 (2000), pp. 351–359. DOI: 10.1016/S0167-739X(99)00059-X. URL: http://www1.cs.columbia.edu/~hgs/teaching/security/hw/keystroke.pdf.

[Mun11] Randall Munroe. *xkcd: Standard*. 2011. URL: https://xkcd.com/927/ (visited on 2016-07-29).

[Rob+98] J. A. Robinson, V. W. Liang, J. A. M. Chambers, and C. L. MacKenzie. “Computer user verification using login string keystroke dynamics.” In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 28.2 (1998), pp. 236–241. DOI: 10.1109/3468.661150.

[SB09] H. Saevanee and P. Bhattarakosol. “Authenticating User Using Keystroke Dynamics and Finger Pressure.” In: *2009 6th IEEE Consumer Communications and Networking Conference*. IEEE, 2009, pp. 1–2. DOI: 10.1109/CCNC.2009.4784783.

[Sal86] Timothy A. Salthouse. “Perceptual, cognitive, and motoric aspects of transcription typing.” In: *Psychological Bulletin* 99.3 (1986), pp. 303–319. DOI: 10.1037/0033-2909.99.3.303. URL: http://faculty.virginia.edu/cogage/publications2/Pre%201995/Perceptual%20Cognitive%20and%20Motoric%20Aspects%20of%20Transcription%20Typing.pdf.

[Sch+01] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. “Estimating the support of a high-dimensional distribution.” In: *Neural computation* 13.7 (2001), pp. 1443–1471. DOI: 10.1162/089976601750264965. URL: http://www.perso.telecom-paristech.fr/~clemenco/Projets_ENPC_files/OneClasslong.pdf.

[Sha78] L. H. Shaffer. “Timing in the motor programming of typing.” In: *Quarterly Journal of Experimental Psychology* 30.2 (1978), pp. 333–345. DOI: 10.1080/14640747808400680.

[Sto11] Tobias Stockinger. “Implicit Authentication On Mobile Devices.” 2011. URL: http://www.eislab.fim.uni-passau.de/files/publications/2011/stockinger2011ImplicitAuthenticationOnMo
pdf.

[Tar07] Tony Targonski. *Heatmap distribution of dirt in a keyboard | CompSci.ca/blog*. 2007. URL: http://compsci.ca/blog/heatmap-distribution-of-dirt-in-a-keyboard/ (visited on 2016-05-07).

[Tas+14] Cheng-Jung Tasia, Ting-Yi Chang, Pei-Cheng Cheng, and Jyun-Hao Lin. "Two novel biometric features in keystroke dynamics authentication systems for touch screen devices." In: *Security and Communication Networks* 7.4 (Apr. 2014), pp. 750–758. DOI: 10.1002/sec.776.

[URJ04] Umut Uludag, Arun Ross, and Anil Jain. "Biometric template selection and update: a case study in fingerprints." In: *Pattern Recognition* 37.7 (2004), pp. 1533–1542. DOI: 10.1016/j.patcog.2003.11.012. URL: http://isrc.ccs.asia.edu.tw/yourslides/files/48/UludagRossJain_TemplateSelection_PR2004.pdf.

[Wik16] Wikipedia. *Motor cortex*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Motor_cortex&oldid=723583673 (visited on 2016-06-05).

[ZDJ12] Yu Zhong, Yunbin Deng, and Anil K. Jain. "Keystroke dynamics for user authentication." In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, June 2012, pp. 117–123. DOI: 10.1109/CVPRW.2012.6239225. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.389.3959&rep=rep1&type=pdf.

Curriculum Vitae

Person

Name:	Philipp Kapfer
Adress:	Drouotstraße 8/14 4020 Linz
Telephone:	0660 / 1434130
eMail:	philipp@kapfer.co.uk
Citizenship:	Austria
Birth Date:	June 17, 1989

Education

- 4 years Primary School (1995–1999)
- 4 years Secondary School (1999–2003)
- 5 years Commercial Academy Rohrbach – Sector Digital Business (2003–2008)
- Bachelor Program Informatik at Johannes Kepler University Linz (2009–2013)
- Master Program Computer Science at Johannes Kepler University Linz (2013–2016)
- Master Program Law and Business Aspects for Technicians (since 2015)

Experience

- GRZ IT Center Linz – Print Operating (July 2006)
- XorteX eBusiness GmbH, Neufelden – Web Developer (July–October 2008)
- Austrian Red Cross, St. Veit – Civil Service (November 2008–July 2009)
- Austrian Red Cross, St. Veit – Holiday Replacement in EMS (August 2010)
- XorteX eBusiness GmbH, Neufelden – Web Developer (July–September 2011 and 2012)
- Johannes Kepler University Linz – IT Technician (November 2012–June 2016)

Projects

- Creation of a staff roster management tool for a bakery
- Development of the web interface for a heating control

- Contribution to open-source projects, eg ACRA, FOG, ownCloud
- Development of an information and navigation system for rescue operations
- Planning and building an image-controlled TV background illumination
- School Leaving Project: Customer Area and Data Accounting for an Internet Service Provider
- Bachelor Thesis: „A Multi-Touch User Interface for Supporting Collaborative Situation Awareness”
- Master Thesis: „PhonyKeyboard: Sensor-enhanced Keystroke Dynamics Authentication on Mobile Devices“

Hard Skills – Soft- & Hardware

- **Operating Systems**
  Windows XP–10, Windows Server 2003–2012 R2, Debian / Ubuntu /OpenSuSE Linux
- **Programming**
  Java (EE), Android, C#, PHP, C
- **Database Systems**
  MySQL, Microsoft SQL Server, SQLite
- **Hardware**
  PC, Server, Raspberry Pi, Network Switches, Arduino

Soft Skills – Languages

- German – Mother Tongue
- English – C1
- Spanish – B1

Honorary Posts & Hobbies

- Honorary Paramedic (since 2008)
- Honorary Librarian (since 2005)
- Hobbies: Bicycling, Swimming, Reading, Computers & Technology

STATUTORY DECLARATION

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
This printed thesis is identical with the electronic version submitted.

Linz, August 2016

Philipp Kapfer