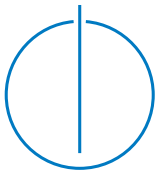
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Pattern Recognition with Smart Devices as Personal Authentication Factor

Philipp Fent



DEPARTMENT OF INFORMATICS

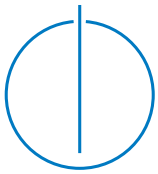
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Pattern Recognition with Smart Devices as Personal Authentication Factor

Mustererkennung mit Mobilgeräten als persönliches Identifikationsmerkmal

Author:	Philipp Fent
Supervisor:	Prof. Dr. Uwe Baumgarten
Advisor:	Nils T. Kannengießer, M.Sc, Prof. Senjun Song, Ph.D.
Submission Date:	15. February 2016



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15. February 2016 Philipp Fent

Acknowledgments

Nils Kannengießer and Prof. Senjun Song as advisors Prof. Dr. Uwe Baumgarten as Supervisor The Chair of Operating Systems to provide test devices Many unnamed proofreaders. TODO: Acknowledgments

Abstract

Every person displays characteristic patterns of behavior, that can be used to verify her or his identity. With the rise of personal smart devices, e.g. smartwatches or smartphones, these patterns can be recorded and analyzed.The resulting characteristics can be used as an additional factor in Multi-Factor Authentication or used as an intrusion detection system by reporting anomalies. In this paper, we analyze the patterns for motion, determined by measuring acceleration. We then evaluate how to efficiently, accurately, and practically extract behavioral patterns, identifying individual users. Furthermore, we developed Android smartphone and smartwatch app prototypes demonstrating the identification capabilities.

TODO: conclusion

Menschen besitzen charakteristische Verhaltensmuster, durch die sie eindeutig identifiziert werden können. Durch ständig mitgeführte Mobilgeräte, z.B. Smartphones oder Smartwatches, können diese Muster aufgezeichnet und analysiert werden. Die daraus abgeleiteten Merkmale können als zusätzlicher Faktor bei Multi-Faktor-Authentifizierungsverfahren oder als Angriffserkennungssystem eingesetzt werden. Im Rahmen dieser Arbeit werden diese Muster anhand von Beschleunigungssensoren, hinsichtlich effizienter und präziser Verhaltensmustererkennung, analysiert. Dies wird durch einen Prototypen einer Android Smartphone und Smartwatch App demonstriert.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**
1.1 Vision . . . 1
1.2 Single user licenses . . . 2
1.3 Multi-factor authentication . . . 3
1.4 Smart mobile devices . . . 3
1.4.1 Smartphones . . . 5
1.4.2 Smartwatches . . . 5
1.4.3 Smart-rings . . . 5
1.5 Pattern recognition . . . 5
1.5.1 Preprocessing . . . 5
1.5.2 Feature extraction . . . 5
1.5.3 Classification . . . 5

2 Approaches **7**
2.1 Key stroke pattern recognition . . . 7
2.1.1 Related work . . . 7
2.1.2 Identification of keystrokes based on acceleration data . . . 8
2.1.3 Features of keystrokes . . . 11
2.1.4 Adaption to phones . . . 11
2.1.5 Adaption to watches and keyboards . . . 13
2.2 Gait recognition . . . 16
2.2.1 Related work . . . 16
2.2.2 General limitations . . . 17
2.2.3 Conclusion . . . 18

3 Implementation **19**
3.1 Platform identification: Device vs. Server 19
3.2 General purpose Android acceleration pattern detection library 20
3.2.1 Sensor recording in Android 21
3.2.2 Sensor measurement framework 21
3.2.3 Preprocessing of SensorData 22
3.2.4 Feature extraction from SensorData 23
3.2.5 Classification and machine learning 24
3.3 Data storage and processing 24
3.3.1 SQLite Database 24
3.3.2 Background verification of patterns 25
3.4 App prototypes 25
3.4.1 Android application 25
3.4.2 Android Wear application 25
3.4.3 Limitations 25

4 Evaluation and Interpretation **26**
4.1 Test setup 26
4.2 Rejection rate vs. accuracy tradeoff 26

5 Conclusion **27**
5.1 Current state 27
5.2 Future prospects 27

Glossary **28**

Acronyms **29**

List of Figures **30**

Listings **31**

Bibliography **32**

1 Introduction

Modern computer systems are facing TODO. Users are one of the most commonly exploited things around computers, especially in social engineering and fishing. Most systems currently rely on a standard combination between username and password. Already in 2009, Aloul et al. described the most common security concerns with passwords:[1] “Users tend to use easy-to-guess passwords, use the same password in multiple accounts, write the passwords or store them on their machines, etc. Furthermore, hackers have the option of using many techniques to steal passwords such as shoulder surfing, snooping, sniffing, guessing, etc.”

Furthermore, many users tend to stay logged into services with their mobile devices, despite not having appropriate security measures for their devices. On most Android phones, full disk encryption is not enabled by default, which leads to another attack vector for identity theft.

Aloul et al. introduced a system of One Time Passwords (OTPs) using mobile phones, which significantly improves security by introducing a second factor of authentication. However, for authentication situations on smartphones themselves the OTP mechanism is rendered pretty much useless, as the second factor is in fact on the same device.

[6]

1.1 Vision

The whole vision of this thesis, is to provide a way to easily detect individual users by a short authentication sequence based on acceleration patterns. Even though this does not qualify as cryptographically secure authentication, behavioural patterns and keystroke recognition can be used as biometric authentication aids. For example, Bhargav-Spantzel et. al.[5] described a system to extract cryptographic biometric keys from biometric data and how this can be combined with additional other proofs of identity to provide strong authentication.

With acceleration pattern recognition, we will make a authentication mechanism

with zero additional user interaction possible. This allows for higher frequency user re-authentication without disturbing and annoying the user. That means, instead of prompting the user with a login screen every 24 hours, we can measure his or her acceleration patterns every time sensitive information is accessed. Therefore, we can not only provide basic login authentication, but also provide a way for users to stay authenticated for longer sessions or even detect when someone else hijacks a valid session. Since the time between two authentication requests can almost be arbitrarily small, an attacker who gets control over the current session will get a new authentication request relatively quickly, thus minimizing the potential damage.

1.2 Single user licenses

Another application of this technique is to identify individual users, even if using the same device. This might not only be useful in terms of individualizing the software according to the current user, but also for tracking of software usage.

A common licensing model for software are per-user licenses, i.e. n licenses for n users of the software. However, this license model currently cannot be enforced, since the software is installed on a single physical device, which may be shared among users. This led to most software companies licensing their software per-installation instead of per-user. As of 2016, many users tend to have multiple devices and also want to use their licenses on multiple devices. This resulted in a trend to bind software licenses to user accounts instead of devices. This trend is also prevailing in modern Software as a service (SaaS) models, which do not require installation of the software on end-user devices anymore. A possible circumvention of these account bound licenses is account sharing. This imposes a real problem, not only for software licensors, but also for other access providers. For example a consumer research from Parks Associates[24] reports, that “6% [of video streaming users] are exclusively using shared accounts to access subscription”.

The methods described in this theses can give a powerful way to detect individual users sharing physical devices, as well as sharing individual accounts and thus reduce copyright infringements that could not even be detected beforehand.

1.3 Multi-factor authentication

Multi–factor authentication (MFA) is a technique to enhance security in access control situations. It combines multiple forms of authentication mechanisms, based on conceptually different approaches: Knowledge, e.g. passwords or PINs; possessions, e.g. keys or bank cards and biometric characteristics, like fingerprints or, as in our approach, behavioural patterns.

A typical authentication attempt with MFA is only successful, when all needed factors are present. The most common example for MFA is banking, where one needs to be in possession of the banking card and needs to know the card's PIN. However, an attack vector targeting this system is copying the banking card while the attacked person does not notice his card being copied. This attack vector is also possible with biometric characteristics and even relatively easy, as many biometric traits are publicly visible. Fingerprints have proven to be copyable with low cost[12] and new high resolution cameras allow to photograph fingerprints and eyes in high enough quality to spoof many scanners[13]. These attacks also can be adapted to other authentication systems based on visible biometric traits, such as iris recognition or Android's Face Unlock.

For biometric authentication to be sufficiently secure, the traits need to be intrinsic, i.e. not publicly visible, and hard to copy. Acceleration based motion detection matches these requirements, as recording of these patterns is only possible with physical access to the authentication device or very close monitoring of all body movement of the user.

1.4 Smart mobile devices

Smart devices are electronic devices, that feature wireless communication, e.g. WiFi or Bluetooth. Smart *mobile* devices are smart devices, that are typically worn or kept in close proximity to the user. This usage usually results in small form factors and little weight. These devices are most often commodity devices and used frequently. Therefore, smart mobile devices are ideal to provide authentication, since the authenticating user is accustomed using the device.

Common examples for smart mobile devices are smartphones and smartwatches.
TODO: Füllbild beschreiben

Figure 1.1: Examples for smart mobile devices that are worn in close proximity of the user (from left to right): A OnePlus One smartphone, a Sony SmartWatch 3, Smarty Ring concept design

1.4.1 Smartphones

Smartphones are the most capable of the smart mobile devices discussed herein. Smartphones usually have numerous wireless communication possibilities and thus function are a personal data-hub, to which other personal devices connect and communicate over. Typical connections for Smartphones are: Cellular network (e.g. GSM, UMTS), WiFi, Bluetooth, Near Field Communication etc. Smartphones are also packed with sensors, which can be utilized by programmers of Apps and typically include acceleration as well as gyroscopic sensors for movement detection.

TODO: Android, IOS, WindowsPhone -> Android biggest target audience

1.4.2 Smartwatches

Smartwatches, bild mitte, beginn von Android Wear, adaption smartphone/smartwatch -> alles android

TODO: Operating systems: more differenciated. Android, WatchOS, Tizen, PebbleOS Market share from [15]

1.4.3 Smart-rings

Future music, no real sensors, maybe special developement necessary to identify users with accelerometers?

1.5 Pattern recognition

TODO: hier schaubild einfügen

1.5.1 Preprocessing

1.5.2 Feature extraction

Dynamic time warping

1.5.3 Classification

Support Vector Machine TODO Schaubild für SVM

k-Nearest Neighbor TODO Schaubild für KNN
Bayes classifier
Neuronal networks
Machine learning with encog
Tensorflow

2 Approaches

2.1 Key stroke pattern recognition

Timings and patterns in key strokes individual characteristics, that can serve as biometric user identification. Individual typing patterns can be extracted from typing samples and can later be used to verify a users identity. The patterns, so called keystroke dynamics, are usually extracted via the key down / up events. Dholi and Chaudhari[11] classified several features from those events, as shown in Figure 2.1: The interval between two key presses, the dwell time of a single key press, the latency between consecutive keystrokes, the flight time and the time from up to up.

2.1.1 Related work

The idea of authentication by keystroke timings started as early as 1980 with Gaines et al.[14] to evaluate the effectiveness experimentally. In their experiment, the scientists gave seven professional typists, i.e. secretaries, a text to type and examined their patterns in typing. Gaines et al. looked at the time to type pairs of successively typed letters, so called “digraphs”. From five of these digraph times, all of the seven typists in this study could be identified.

In 1997, Dieter Bartmann presented PSYLOCK[4], a system that analyzes the keystroke rhythm of text input and identifies users according to these rhythms. The system claimes to work with an arbitrary text of ca. 100 characters. PSYLOCK uses an approach based on statistical models in combination with support vector machines.

In 1999, Monrose and Rubin[21] proposed a new approach at keystroke identification: Continuous keystroke verification. Previous attempts to keystroke verification only used static verification, i.e. they verified the characteristics only at specific times, for example during login. Monrose’s and Rubin’s system monitors the user’s typing behaviour continuously and thus can provide a significant improvement of security. The basic approach in their classification algorithm is clustering feature sets, determined through factor analysis, with a K-Nearest Neighbor approach.

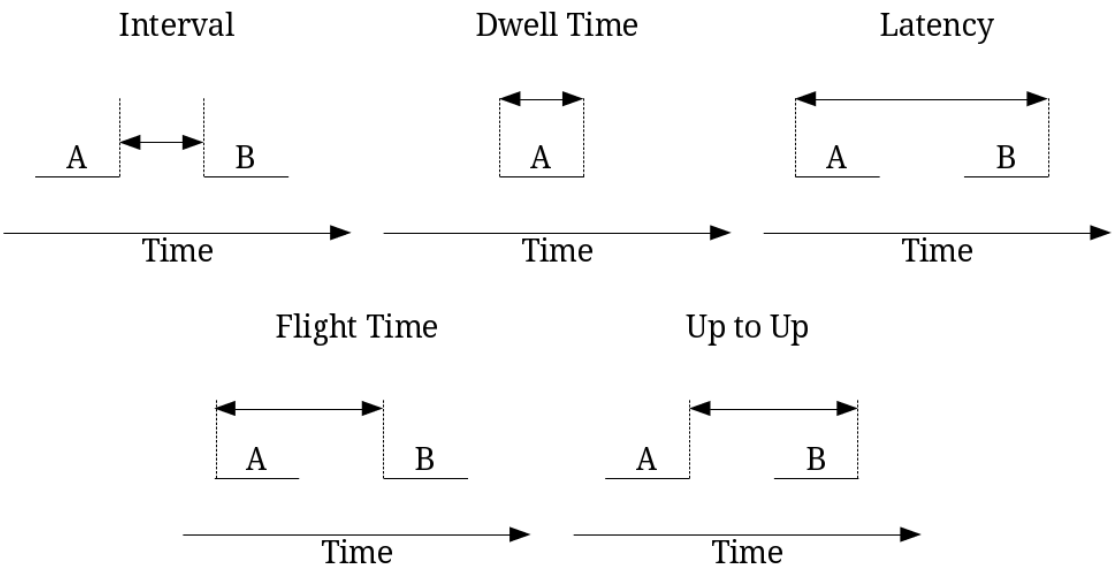


Figure 2.1: The five temporal features of keystroke dynamics[11]

Clarke and Furnell[9] published a paper on identifying mobile phone users using keystroke analysis in 2006. In this paper, users were identified by typical handset interactions: entering telephone numbers and writing text messages. For this paper, Clarke and Furnell compared several multi-layered neuronal networks of different flavour, all of which performed approximately the same. Since the form factor of mobile phones has drastically evolved in the last 10 years, their approach is largely obsolete for modern smartphones lacking physical buttons.

2.1.2 Identification of keystrokes based on acceleration data

Since smartphones lack physical buttons, but contain several high accuracy motion sensors, correlating taps on the screen and keystrokes is a natural assumption. Typical character input on phones is done via on-screen keyboards controlled via taps on the screen.

With identifying keystroke patterns with acceleration sensor data, we are not limited to simple character and text input, but can also identify arbitrary tap sequences, that might occur in authentication scenarios. For example reading emails or messages might be

even more secure-worthy than writing messages. Our approach can also extract and match patterns to authenticate users in simple navigation flow.

As a side note, typically the access to the on-screen keyboard is significantly restricted for apps to hinder keyloggers from sniffing passwords. However, access to acceleration sensors is pretty much unrestricted and can also be done in web-browsers such as Google Chrome via an Javascript API[7].

In 2012, Miluzzo et al.[20] introduced *TapPrints*, a mechanism to extract the location of screen taps solely from accelerometer and gyroscope data. With a machine learning approach, the system is able to detect taps with a bagged decision tree classifier to almost 100%. Furthermore they were even able to guess individual letters in about 50% of the cases.

For the scope of this bachelor's thesis, a complex training with machine learning algorithms to detect taps is unnecessary. We can detect individual taps via a simple peak detection algorithm, as described by Palishkar et al.[23]. An example for peak recognition on the time series of sensor measurements is shown in Figure 2.2. Eventual inaccuracies in identification of individual taps is not necessarily considered negative to the overall pattern recognition scheme, because those peaks are also part of the user's individual behaviour.

Palishkar's peak detection algorithm identifies peaks (or spikes) in a given time-series of values. These peaks represent keystrokes or simply "values of interest" in our acceleration data. between the detected peaks, no disturbance in the phones acceleration is found, i.e. the user does not touch or move the phone. A data point in our data is a *local* peak, if it is a maximum value within a defined window and not too many other points in the window have similar values.

Palishkar's algorithm is a parametric algorithm, that can be adapted to the individual structure of the time series data. The algorithm takes two parameters additional to the time series: the window size k around the peak to detect and a stringency h that rejects "low" peaks based on Chebyshev's inequality. Chebyshev's inequality states that for a random variable X with mean μ and standard deviation σ, $P[|X - \mu| \geq h\sigma] < \frac{1}{h^2}$. Palishkar recommends a typical h with $1 \leq h \leq 3$ for peak detection. We can then make a sophisticated guess, that a peak x with $|x - \mu| < (h * \sigma)$ is "small" in a global context and thus not an interesting peak.

The window size k restricts the amount of peaks detected within k data points. To optimize the peak detection, we need to adapt k to the typical time of a key press. Is k too small, we might face the problem of detecting back-swings in the sensor data as additional peaks, is k too big, subsequent keystroke might not be recognized. We

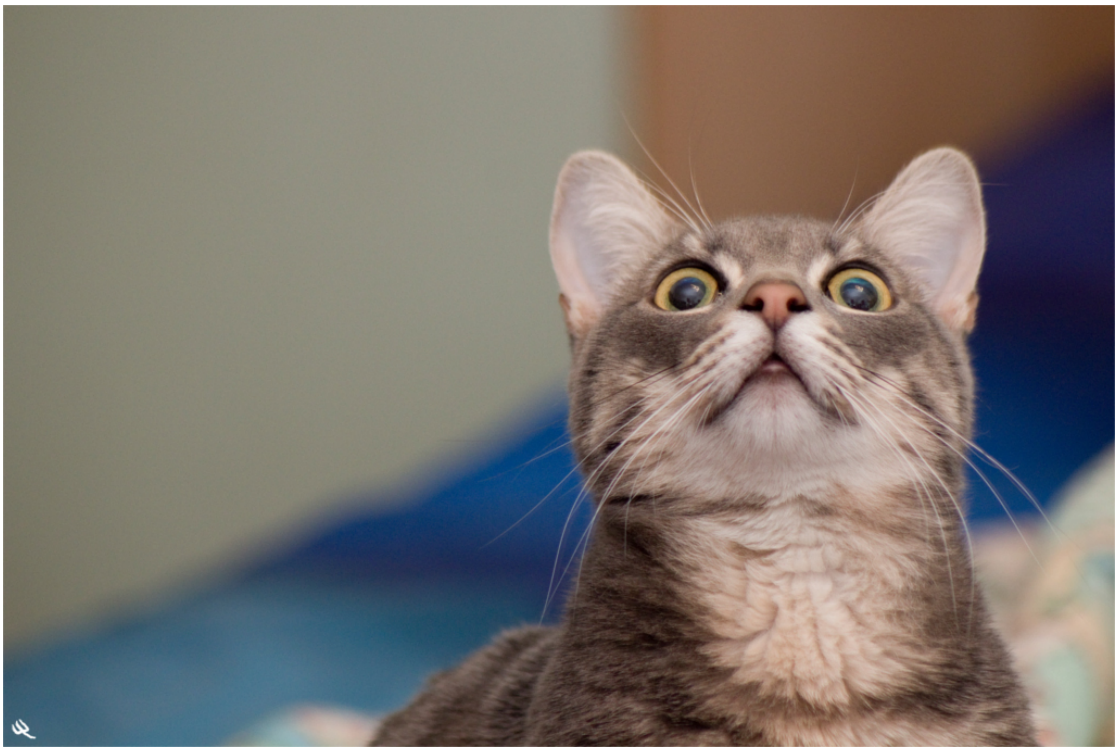


Figure 2.2: TODO Detected peaks in sensor measurements

therefore analyzed typical usages to find a good k, as outlined in Section 3.2.4.

2.1.3 Features of keystrokes

Historically, the features of key strokes were only extracted from the events keyboards reported to the operating system. In example the X.Org Server, the de-facto standard input handling system in UNIX-like operating systems, handles a single key press via two separate events: A `KeyPress` event, whenever a key is pressed down and a `KeyRelease` event, when the key is lifted up again.

These two events can now be measured in separate metrics, as shown in Figure 2.1. Furthermore, when considering more than two keystrokes, we can gather additional possible measurements:

- Overall typing speed, usually measured in Characters per minute (CPM)
- Overall typing rhythm and flow, measured in fundamental frequencies
- Intensity of taps, measured by the amplitude of acceleration

These keystroke dynamics can be measured by several different aspects. To recognize typing patterns in arbitrary text, the typing patterns are usually broken down to di-graph, tri-graph of general n−graph segments. This means, that the overall typing pattern is reduced to sequences of $2, 3, ..., n$ key presses and only the features of these n−graphs are analyzed.

In our approach, we are monitoring the user's input in a controlled environment, i.e. we only monitor input of the same sequence of characters, for example a password or a defined navigation sequence. This allows our approach to compare the whole sequence and we don't need to identify n−graphs in this user input. Extending our implementation to also extract these graphs might be a lever to improve the generality of the implementation in subsequent work.

2.1.4 Adaption to phones

For the implementation of keystroke recognition on smartphones, we first need to identify, which TODO. In this approach, we are considering text input directly on a smartphone with no additional devices. In Android, the inertial coordinate system for the acceleration sensors is as displayed in Figure 2.3. The coordinate system, according to which the

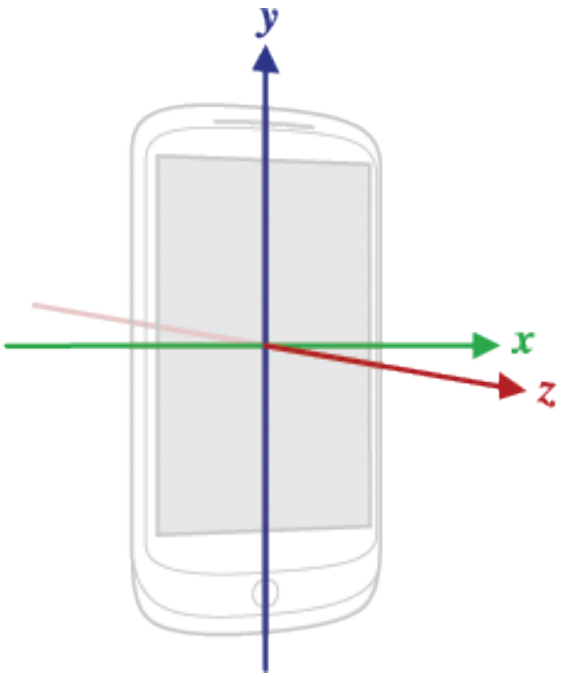


Figure 2.3: Inertial coordinate system of Android devices[2]

acceleration sensors report their measurements is defined relative to the default orientation of the device and are static, despite orientation changes of the devices display. The X-axis points horizontally to the right, the Y-axis vertically up and the Z-axis points towards the outside of the front face of the screen[2].

The main force of taps on a touchscreen is opposite to the direction of the Z-Axis as displayed in Figure 2.4. Thus, we can safely neglect the X- and Y-axis for our use-case of identifying individual taps on the screen.

The basic approach in keystroke classification on smartphones then would be to use a peak detection algorithm, as described in Section 2.1.2. We can apply this algorithm to the Z-axis acceleration sensor records of the phone, which yields sufficiently good data for individual taps.

Figure 2.5 compares actual pressed keys and detected taps on the screen.

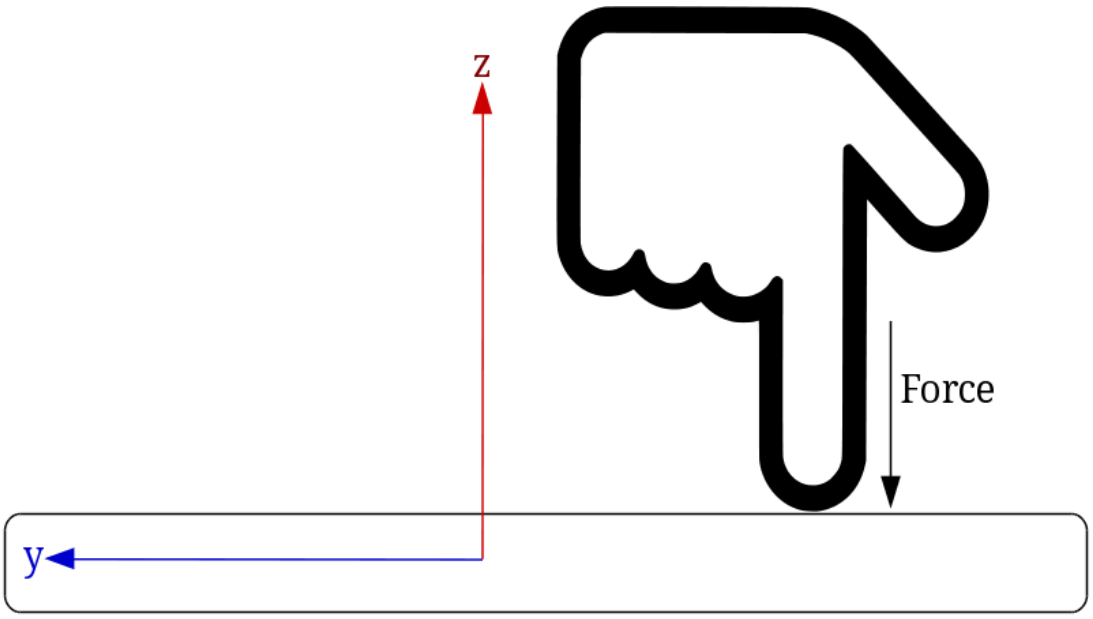


Figure 2.4: Model of touches on Android devices

2.1.5 Adaption to watches and keyboards

For recognition of key presses with smartwatches, we examined a scenario of a user wearing a smartwatch while typing on a physical keyboard. In this scenario, the X,Y-plane of the watch's inertial coordinate system is almost parallel to the keyboard (compare Figure 2.6).

Wang et al.[29] discussed in their paper of MotionLeaks for MobiCom'15, that the key press timings on keyboards can be extracted by the Z-axis movement of the watch. When the user presses a key on the keyboard, the user's finger dips and the wrist also undergoes a partial dipping motion. This motion can be detected by the Z-axis acceleration, in combination with a peak detection algorithm, similar to the one used with phones in Section 2.1.4.

Wang et al. also improoved their peak detection algorithm by chaining a peak detection tool with a bagged decision classifier. Since their goal was to guess individual typed keys instead of analysing the pattern as whole, the additional computational overhead might be worthwhile. This however, does not hold for our approach of matching the whole input pattern, thus we use a simple peak detection algorithm.

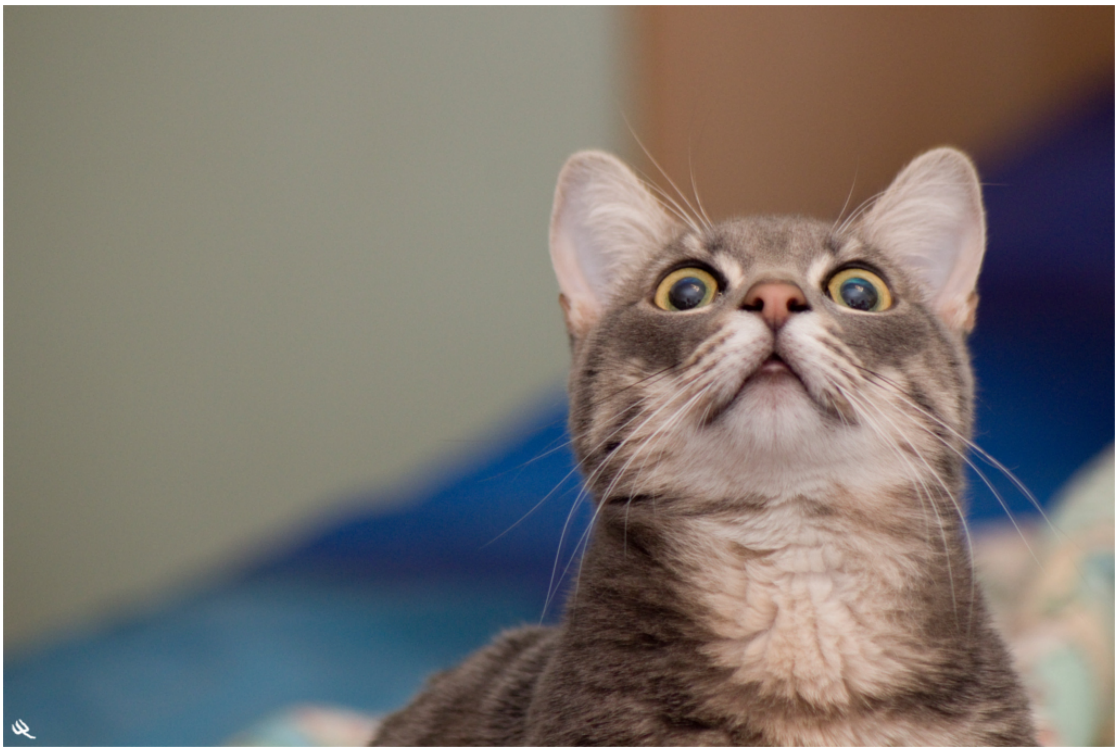


Figure 2.5: TODO Detected peaks and corresponding taps on the screen

Figure 2.6: The coordinate system of a smartwatch while typing on a keyboard

Additionally to keystroke pattern recognition the overall movement of the watch while typing can be used as an additional authentication vector. Different users might move their hands differently for text input. However we are not aware of scientific studies measuring the effectiveness of this approach.

2.2 Gait recognition

Gait is defined as “the way in which a person (...) walks” in the Cambridge Dictionary. Various scientific papers analyzed the specifics of human gait[16, 18, 17] and concluded, that individual gait can be used as a biometrical recognition mechanism. The steps a user walks throughout the day can for example be used to generate user profiles and extract an identifying pattern.

As early as 1975, Johansson[16] had shown, that observers could identify individuals just by watching videos of lights mounted to joints of otherwise invisible walking people. Additionally, the observers were able to not only identify previously known people, but also identify the gender of unknown persons. However human gait is influenced by many more personal aspects, as the individual weight, leg length, posture and speed of walking. Thus gait patterns are highly individual and are usually unique.

2.2.1 Related work

The early attempts to gait recognition used video footage and moving light displays to extract the gait information. This approach works quite well, but is largely unpractical since face and shape recognition algorithms work even more precise on videos. Starting in 2005 with Mäntyjärvi et al.[19], researchers used accelerometers to extract gait information of users. These sensors were attached to different body parts, such as hip, arms and feet to evaluate the recordable data. Optimal positioning of these accelerometers is still disputed, but recent studies showed, that portable devices such as commercial phones[10] or smartwatches[17] are sufficiently good sensors for gait recognition.

For gait recognition with mobile phones, Schmidtbartel[26] implemented a framework to recognise specific user-device combinations. His model accumulated sensor data of the user’s gait and aggregates a median step pattern, as shown in Figure 2.7.

Each individual recorded step circle may vary due to signal noise or different user behaviour. Shown in the bottom half of the figure as individual lines, individual step circles are recorded in light grey. In black, the calculated median is shown. To calculate

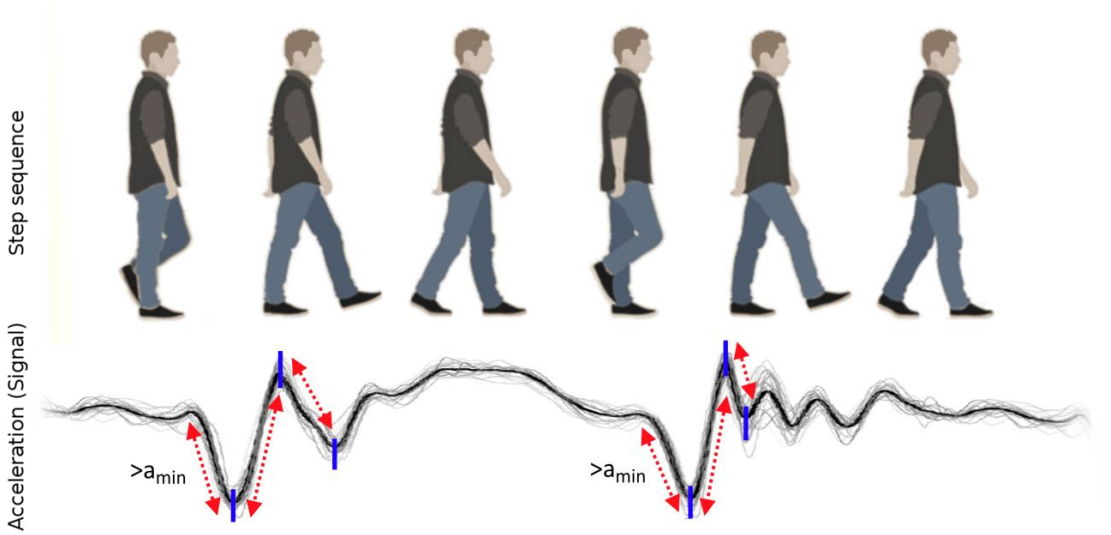


Figure 2.7: Motion circle and corresponding measured acceleration data[26]

this value, multiple steps are clustered, according to the similarity of those steps. As a result, there might be multiple clusters of gait signals, i.e. for walking, jogging or running.

Schmidtbartel is using an Manhattan-Distance metric in his implementation, however other researchers[10] suggest, that Dynamic time warping (DTW) and an extension of DTW, called Cross DTW Metric even result in better gait recognition performance.

2.2.2 General limitations

To monitor the gait patterns of users, constant monitoring of the devices sensors is necessary, even though the user is not actively using the device. This prevents the device to go in so called “deep sleep” state where less energy is consumed. Since battery is mostly a big concern on mobile devices, this is a major deal-breaker. Users tend to deinstall battery draining applications quickly.

For authentication purposes, an timely response to whether or not the authentication attempt was successfully is required. However, gait is not available all the time and certainly not on demand. Prompting the user to take a walk to get access to his data is not an option.

2.2.3 Conclusion

As consequence of these limitations, we decided against using an gait recognition approach as personal authentication factor. Nonetheless, gait recognition might be an additional approach for intrusion detection. For example the device itself can recognise it being stolen by detecting other gait patterns. This allows the device to take countermeasures, e.g. lock itself, alert the user and activate “Find My Device” functionality. In contrast, gait recognition is not suitable for concrete, immediate authentication needs, as it is the vision of this paper.

3 Implementation

For this paper, we are implementing app prototypes to demonstrate the capability of pattern recognition as personal authentication factor. We chose Android as the main smart device platform, since access to development tools and documentation is freely available. Also Android applications are developed using Java, which allows to use many existing libraries. Android code also remains portable across different form factors of devices, such as phones, tablets and smartwatches running Android-Wear.

The development and testing of the Android application was conducted on the author's personal devices, an OnePlus One and a Nexus 10. To be able to develop an Android Wear app, the Chair of Operating Systems kindly provided a Sony Smartwatch 3.

For the implementation we also used SQL to store data gathered in tests platform-independently. To visualize and plot the sensor measurements and their correlation to keystrokes, we used Python with the excellent *matplotlib*.

3.1 Platform identification: Device vs. Server

One of the first tasks when designing applications, is to analyze the target platform, the application should run on. For an authentication scenario, we can identify two different platforms: A client, in our case an Android device, and a server, which wants to authenticate a user. For our approach, we can use both platforms, to base the pattern recognition of the acceleration data on. Thus we can gather arguments pro and contra each platform.

We could process all the data locally on the device, since we already need an Android App to record the accelerometer data. When we do the data processing client-side, we can keep the server application as small as possible. Since typical Android devices have multi-core processors, they also should be computationally powerful enough to process the data. In our test scope, all sensor processing work was handled fast enough to not create any user noticeable wait times. Moreover, decentralized data processing on the individual devices also has significant advantages in reducing server load, which allows

the authentication approach to easily scale to a big number of users.

Local processing of the acceleration data also reduces the size of the data transmitted to the server. The raw data can easily reach several megabytes, which is especially troublesome for mobile devices. When the networking connectivity of mobile devices can be slow, e.g. in cellular networks, raw data transmission can take several seconds.

There are also approaches to privacy protecting biometric authentication by locally generating cryptographic bio-keys[5, 28, 25]. Since handling personal identification information requires special precautions to not loose the data, these bio-keys can be used to implement so called “zero-knowledge proof of knowledge” protocols. This is especially useful, since only the information needed to verify the proof of knowledge needs to be stored on the server. If an attacker acquires this information, he can only verify the identity of the user and is not able to extract personal data about the user, thus preserving the users privacy.

A client-side data processing can also be used to authenticate the user locally, e.g. when entering the phones unlock code. This can be used for intrusion detection or for locking stolen devices.

However, server side authentication is easier to deploy, since changes in the authentication algorithms only need to be made in a single place. Authentication spoofing also is harder, since an attacker can analyze locally installed apps, but does not have access to the server application.

For this paper, we implemented a client-side authentication mechanism. In our opinion, the privacy of a user is very important and personal data should not be transmitted over the network, when it can be avoided.

3.2 General purpose Android acceleration pattern detection library

Since we are creating two different apps, a normal Android and an Android Wear variant, we quickly decided to share code among those apps. To do this, we created a general purpose acceleration pattern detection library for Android. This also allows rapid prototyping of the different approaches outlined in Chapter 2.

The main goal of this library is easy integration into existing apps to enhance the authentication security without the need for specially crafted implementations. With our library, apps can compose their own pattern recognition implementations in a modular way, based on a stable and modular basis.

Listing 3.1: Obtaining the default acceleration sensor data in Android

```
SensorManager mgr = (SensorManager) context
    .getSystemService(Context.SENSOR_SERVICE);
Sensor sensor = (manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
mgr.registerListener(myListener, sensor, SensorManager.SENSOR_DELAY_FASTEST);
```

pipeline erklären TODO: hier pipeline schaubild einfügen.

3.2.1 Sensor recording in Android

In Android, the access to sensors of the device is managed by the `SensorManager` class. The process of obtaining the `SensorManager`, getting the default acceleration sensor and registering a custom `SensorEventListener myListener` is shown in Listing 3.1. The mechanic of getting data is then defined in the `SensorEventListener`, which is periodically called by the Android system with new data.

The rate at which the `SensorEventListener` gets callbacks is defined via the third parameter of the `SensorManager.registerListener()` function. In our case, we are using the fastest rate possible, since we don't want to miss even the slightest features of the movement pattern. Miluzzo et al.[20] also have shown in their paper about guessing letters from device movement, that their results drastically improve with higher sensor sampling rate. Hence, we chose to simply poll the sensor at the fastest rate possible, with `SensorManager.SENSOR_DELAY_FASTEST`. In our tests, this corresponds to a sapling rate of about 200 Hz.

3.2.2 Sensor measurement framework

Within the library, we provide a simple way to record sensor values into a predefined data structure, called `SensorData`. All classes defined to measure and record the sensors are organized in the `measurement` package.

The `SensorData`, as shown in Listing 3.2, consists of a two-dimensional array of floats, called `data`. The first dimension of this defines the direction of the sensor measurement, i.e. X- Y- and Z-acceleration, while the second dimension defines the series of individual measurements. We also record a timestamp of the individual measurements in the `timestamps` array. This is necessary, since the time between measurements can vary,

Listing 3.2: Class `SensorData` containing the raw sensor readings

```
public class SensorData {

    public final float[][] data;
    public final long [] timestamps;

    public SensorData(float[][] data, long[] timestamps) {
        this.data = data;
        this.timestamps = timestamps;
    }

    public int getDimension() {
        return data.length;
    }
}
```

depending on the current load of the processor. For example, if we access `data[0][41]`, we get the 42nd measurement of the X-acceleration in the series. The corresponding timestamp can be accessed via `timestamps[41]`.

Since arrays with static size are not suitable for building up data, we also defined a corresponding `SensorDataBuilder`, that uses lists of dynamic size to append new measurements. We can do this by calling the instance method `SensorDataBuilder.append()`, that dynamically grows the list as needed. When we completed recording of new measurements, we can create a `SensorData` object of these measurements with the `SensorDataBuilder.toSensorData()` method.

The reasoning behind converting the data from lists to arrays, is that arrays have significantly less overhead, in accessing random data as well as memory consumption. Also the preprocessing and feature extraction steps usually operate on simple arrays. So instead of converting the data for each step, we only use dynamic lists for buildup and use simple arrays afterwards.

3.2.3 Preprocessing of SensorData

To create meaningful and comparable sensormeasurements, we needed to implement a preprocessing step after recording the sensor data. Since the measured acceleration includes gravity, we need to factor it out, depending on how the user holds the device.

To negate the effect gravity has on the measured data, we assume, that the device is relatively static in overall acceleration. This holds true for most applications, such as the device is lying on a desk or a user is carrying it around. We neglect the fact, that we cannot simply factor out gravity when we are measuring in a changing acceleration environment, e.g. driving in a car.

To factor out the gravity, we normalize the measured data to get a mean value of 0. We can do this by simply calculating the mean value and shift all sensor values by the negative mean, which results in the mean afterwards being 0. This results in an overall formula of: $x_{new} = x - \mu$ with μ being the mean of the measurement.

To enhance the sensor recordings, we also implemented smoothing functions, that are able to dampen sensor noise. As for this paper, we implemented a simple moving average and exponential smoothing filters. The simple moving average algorithm works by averaging a certain number of measurements in a so called “window size”. For comparison, we also implemented a simple moving average filter, which factors in the last smoothed value with a factor α with $0 < \alpha < 1$.

TODO: normal signal vs. moving average vs. exponential figure.

Since the rate of sensor measurements can vary with the processor load, we also could interpolate the measurements to a continuous function or simply a fixed rate. A technique to implement this would for example be cubic spline interpolation. However our sensor measurements are fairly regular and sufficient for a proof-of-concept, with a standard variance in measurements $< 0.01ms$. In real world applications with varying loads and background activity during sensor measurements, this might be needed.

Since there might be multiple preprocessing steps needed, we also implemented a simple `ComposingPreprocessor`, which can combine multiple preprocessing steps to a single one. We then simply apply the other preprocessors sequentially as specified. Thus we can for example first noremalize the data to a mean of 0 and then smooth it with one of the implemented smoothing functions.

3.2.4 Feature extraction from SensorData

Transformation into FeatureVectors PeakDetector

PhoneKeystrokeFeatureExtractor -> Tap intensity and dwell time Parameters for peak detection

WearKeystrokeFeatureExtractor -> Tap intensity, dwell time and wear movement

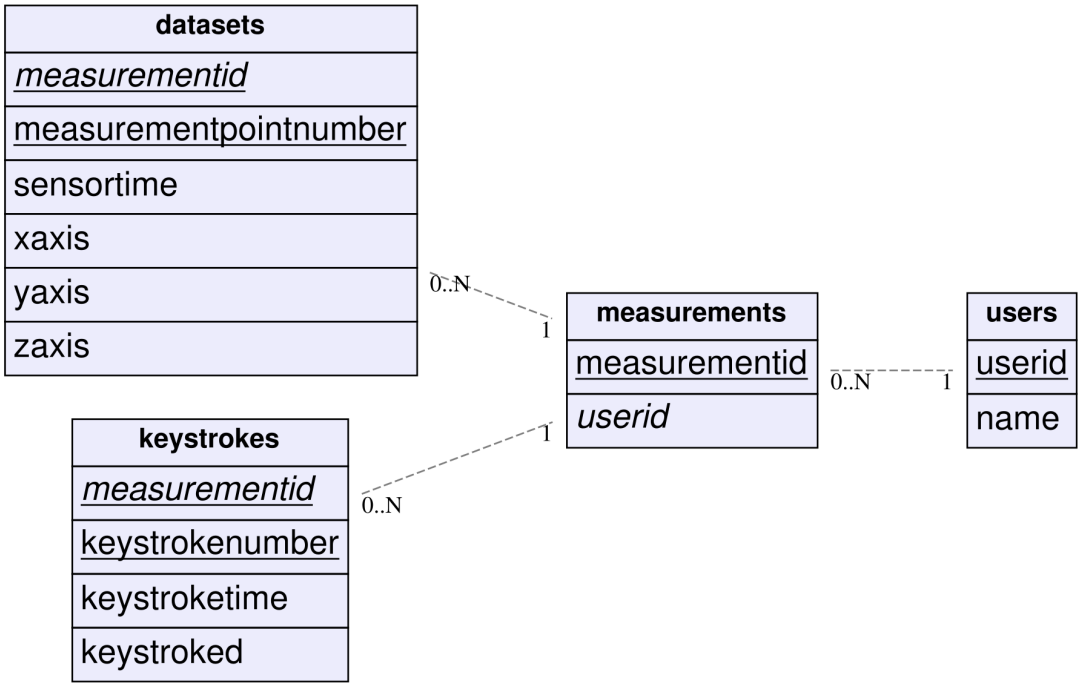


Figure 3.1: Schema of the SQLite database used by the prototypes

3.2.5 Classification and machine learning

distance between feature vectors via DTW classification clustering and classification with kNN Classifier, implement Encog?

3.3 Data storage and processing

Persistent storage is important -> machine learning Replay of previously recorded data to compare approches

3.3.1 SQLite Database

SQLite lightweight database. standard android. no need for extra features
.db file can be easily shared among devices -> data visualization via python

3.3.2 Background verification of patterns

Planned enhancement for unobstructed user experience. Not implemented yet

3.4 App prototypes

App prototypes on Github. Android Studio project, 3 different modules to share code between Phone and Wear project

Sensorprocessing: Implementation of the sensor recording, preprocessing, analyzing and classification

App: Phone / Standard Android module, Activities for Text input / user information what this app does

Right now: only proof of concept, due to limited time. No real authentication of users. demonstrating

TODO: figure of module structure

3.4.1 Android application

TODO: application screenshots

onFocusChangeListener to define start and end of text input

combined authenticator: accelerometer, normalizer, smoothing, PhoneKeystrokeFeatureExtractor, kNN Classifier with DTW distance

3.4.2 Android Wear application

TODO: wear screenshots

3.4.3 Limitations

4 Evaluation and Interpretation

4.1 Test setup

4.2 Rejection rate vs. accuracy tradeoff

4.3

5 Conclusion

5.1 Current state

5.2 Future prospects

Glossary

app An application program. A computer program designed for a specific type of application.

dynamic time warping An algorithm measureing the similarity between time series data, compensating variations in time or speed.

keystroke dynamics Individual characteristics in typing, e.g. key strokes, that can be used to identify users.

Manhattan-Distance A distance measuring the distance between data points, accoring to the sum of the absolute differences of their coordinates.

Acronyms

CPM Characters per minute.

DTW Dynamic time warping.

GSM Global System for Mobile Communications, originally Groupe Spécial Mobile.

MFA Multi–factor authentication.

OTP One Time Password.

SaaS Software as a service.

TUM Technische Universität München.

UMTS Universal Mobile Telecommunications System.

List of Figures

1.1 Examples for smart mobile devices that are worn in close proximity of the user (from left to right): A OnePlus One smartphone, a Sony SmartWatch 3, Smarty Ring concept design 4

2.1 The five temporal features of keystroke dynamics[11] 8
2.2 TODO Detected peaks in sensor measurements 10
2.3 Inertial coordinate system of Android devices[2] 12
2.4 Model of touches on Android devices 13
2.5 TODO Detected peaks and corresponding taps on the screen 14
2.6 The coordinate system of a smartwatch while typing on a keyboard 15
2.7 Motion circle and corresponding measured acceleration data[26] 17

3.1 Schema of the SQLite database used by the prototypes 24

Listings

3.1 Obtaining the default acceleration sensor data in Android 21
3.2 Class `SensorData` containing the raw sensor readings 22

Bibliography

[1] F. Aloul, S. Zahidi, and W. El-Hajj. “Two factor authentication using mobile phones.” In: *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*. IEEE. 2009, pp. 641–644. URL: http://staff.aub.edu.lb/~we07/Publications/Two%20Factor%20Authentication%20Using%20Mobile%20Phones.pdf.

[2] Android Open Source Project. *Android's SensorEvent API reference*. https://developer.android.com/reference/android/hardware/SensorEvent.html. Accessed: 2016-01-05.

[3] Android Open Source Project. *Android's SensorManager API reference*. https://developer.android.com/reference/android/hardware/SensorManager.html. Accessed: 2016-01-09.

[4] D. Bartmann. “PSYLOCK—Identifikation eines Tastaturbenutzers durch Analyse des Tippverhaltens.” In: *Informatik'97 Informatik als Innovationsmotor*. Springer, 1997, pp. 327–334.

[5] A. Bhargav-Spantzel, A. Squicciarini, and E. Bertino. “Privacy preserving multi-factor authentication with biometrics.” In: *Proceedings of the second ACM workshop on Digital identity management*. ACM. 2006, pp. 63–72.

[6] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[7] S. Block and A. Popescu. *DeviceOrientation Event Specification*. http://www.w3.org/TR/orientation-event/#devicemotion. Accessed: 2016-01-04.

[8] T. Braun. *User authentication from acceleration sensor data*. http://www.mobsec.rub.de/media/mobsec/arbeiten/2014/12/12/2014-ba-braun-knockknock.pdf. Accessed: 2015-12-10.

[9] N. L. Clarke and S. Furnell. “Authenticating mobile phone users using keystroke analysis.” In: *International Journal of Information Security* 6.1 (2007), pp. 1–14.

[10] M. Derawi and P. Bours. “Gait and activity recognition using commercial phones.” In: *computers & security* 39 (2013), pp. 137–144.

[11] P. R. Dholi and K. Chaudhari. “Typing pattern recognition using keystroke dynamics.” In: *Mobile Communication and Power Engineering*. Springer, 2013, pp. 275–280.

[12] evelyn and starbug. “Basteltips Biometrieversand.” In: *die datenschleuder* 92 (2008), 56f.

[13] T. Fiebig, J. Krissler, and R. Hänsch. “Security impact of high resolution smartphone cameras.” In: USENIX Association, 2014.

[14] R. S. Gaines, W. Lisowski, S. J. Press, and N. Shapiro. *Authentication by keystroke timing: Some preliminary results*. Tech. rep. DTIC Document, 1980.

[15] IDC. *Market share of smart wristwear shipments worldwide by operating system in 2015 and 2019*. `http://www.statista.com/statistics/466563/share-of-smart-wristwear-shipments-by-operating-system-worldwide/`. Statista, Accessed: 2016-01-09.

[16] G. Johansson. “Visual motion perception.” In: *Scientific American* (1975).

[17] A. H. Johnston and G. M. Weiss. *Smartwatch-Based Biometric Gait Recognition*. `http://storm.cis.fordham.edu/gweiss/papers/btas-2015.pdf`. Accessed: 2015-12-10.

[18] L. Lee and W. E. L. Grimson. “Gait analysis for recognition and classification.” In: *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*. IEEE. 2002, pp. 148–155.

[19] J. Mäntyjärvi, M. Lindholm, E. Vildjiounaite, S.-M. Mäkelä, and H. Ailisto. “Identifying users of portable devices from gait pattern with accelerometers.” In: *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*. Vol. 2. IEEE. 2005, pp. ii–973.

[20] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury. “Tapprints: your finger taps have fingerprints.” In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM. 2012, pp. 323–336.

[21] F. Monrose and A. D. Rubin. “Keystroke dynamics as a biometric for authentication.” In: *Future Generation computer systems* 16.4 (2000), pp. 351–359.

[22] R. Moskovitch, C. Feher, A. Messerman, N. Kirschnick, T. Mustafić, A. Camtepe, B. Löhlein, U. Heister, S. Möller, L. Rokach, et al. “Identity theft, computers and behavioral biometrics.” In: *Intelligence and Security Informatics, 2009. ISI'09. IEEE International Conference on*. IEEE. 2009, pp. 155–160.

[23] G. Palshikar et al. “Simple algorithms for peak detection in time-series.” In: *Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence*. 2009.

[24] Parks Associates. *Nearly 60% of U.S. broadband households use OTT video services, but “Account Sharing” is prevalent*. `http://www.parksassociates.com/blog/article/cus-2015-pr11`. Accessed: 2015-12-10.

[25] A. Ross and A. Othman. “Visual cryptography for biometric privacy.” In: *IEEE transactions on information forensics and security* 6.1 (2011), pp. 70–81.

[26] N. Schmidbartl. “Analyse zur Identifizierung von Benutzern/Geräten anhand verschiedener Faktoren und Integration in ein Framework.” Accessed: 2016-01-06. MA thesis.

[27] A. P. Shanker and A. Rajagopalan. “Off-line signature verification using DTW.” In: *Pattern recognition letters* 28.12 (2007), pp. 1407–1414.

[28] E. Verbitskiy, P. Tuyls, D. Denteneer, and J. Linnartz. “Reliable biometric authentication with privacy protection.” In: *Proceedings of the 24th Benelux Symposium on Information theory*. 2003.

[29] H. Wang, T. T.-T. Lai, and R. Roy Choudhury. “MoLe: Motion Leaks through Smartwatch Sensors.” In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM. 2015, pp. 155–166.