ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Fraud Prevention and Detection in Mobile Payments

Master Thesis

CONFIDENTIAL

Cédric Waldburger

`wcedric@ee.ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zurich

Supervisors:
Christian Decker
Conor Wogan (SumUp)
Prof. Dr. Roger Wattenhofer

April 18, 2013

Acknowledgements

I would like to thank my advisors Prof. Roger Wattenhofer and Christian Decker from the Institute for Distributed Computing at ETH Zurich for their help and support. I also wish to thank Stefan Jeschonnek, Conor Wogan, Matti Biskup and the whole software development team at SumUp for their continuous support.

Abstract

Mobile payments have seen a lot of traction recently and a set of companies has emerged around the opportunity of payments processed from mobile devices. We've conducted this work in collaboration with a company that allows merchants to process credit card transactions on their smartphone by using a mobile application and a hardware credit card reader which plugs into the phone's headphone connector.

The innovative technology has opened up new beneficial opportunities for both, merchants and clients but it has also opened new opportunities for fraud in the area of credit card transactions. Historically prone to fraud, the credit card industry is highly regulated. However, as existing anti-fraud regulations are specified for the traditional credit card business, the advances in technology call for new anti-fraud measures.

We identify two separate but complementary methods against credit card fraud: **preventive**, in the form of an automatic, in-depth check of all users during the sign-up process against a database of individuals with high risk status, and **reactive** by providing mechanisms to verify every transaction's signature in real time with the card holder's previous signatures to detect if a stolen or copied card is used by an impostor.

Our work has shown that the optimization of the check during sign-up is able to reduce the number of unnecessary matches by up to 90 percent which results in a reduction of 8 - 16 hours of manual work done by the operations team each month. At the same time, we were able to make the system more reliable by reducing the number of false positive matches significantly.

We found the characteristics of a signature captured by finger on a mobile device to be much less stable compared to signatures captured with a pen. To account for the resulting higher number of false positives forgery matches, we propose to use a feedback loop to ensure no lost transactions. We propose a real time signature verification algorithm that fuses the score of a Dynamic Time Warping implementation, an algorithm based on Hidden Markov Models and a score based on global features. With over 96 percent accuracy, our system was able to identify forgeries in real time.

Keywords: Fraud Detection, Signature Verification, Mobile Payments, Mobile Applications, Dynamic Time Warping, Hidden Markov Models

Contents

CONTENTS iv

CHAPTER 1

Introduction

As smartphones become ubiquitous, they are being used in more and more industries and fields. An application that has seen a lot of traction recently is mobile payments. The usage of the smartphone in this field includes its utilization as an electronic wallet, a virtual bank account and as a payment processing terminal to accept credit cards and other payment methods. This work focuses on an application of the latter. The density of smartphones per capita and the computational power of current mobile phones have reached a level where it becomes practical to use phones as electronic cash registers. iZettle[1], Square[2] and SumUp[3] are just some of the companies which make use of this situation and are enabling merchants to receive payments via their smartphone. This thesis was written in collaboration with SumUp, a mobile payment company located in Berlin.

Financial businesses have a higher risk of becoming targets of fraud attacks than businesses in other industries due to the direct financial gain for fraudsters. On mobile devices, it becomes even more important to build automatic processes to protect the system from fraud as mobile payments are characterized by low amounts per transaction but a very large quantity of transactions which makes it unfeasible to monitor all transactions manually. While some security standards exist and are imposed by financial authorities, those rules were made for traditional financial institutions. The new mobile, connected environment not only opens up a lot of beneficial opportunities but also more attack vectors for fraudsters and requires advanced countermeasures. Therefore, it is important to not only respect given guidelines but pro-actively develop new ways and adapt existing guidelines to the specifics of the mobile environment to prevent attacks.

We focus our work on two separate, complementary approaches to reduce the fraud risk for a mobile payment company like SumUp. We implement in-depth checks of new users during the sign-up process to reduce the risk of signing up a high risk user. After screening a new user, suspicious applications are flagged for

---

[1]http://izettle.com
[2]http://squareup.com
[3]http://sumup.com

manual inspection of the information that caused the flagging. This significantly reduces the amount of accounts to be checked manually and we thereby not only improve the security of the sign-up process but also allow SumUp to continue its exponential growth by removing a bottle neck of the process.

Our second approach focuses on reactive fraud detection and improves the security of the most popular method to authorize credit card transactions in mobile payments: by signature. Authorization of credit card transactions can be done in various ways. They differ in terms of technical complexity and thus cost, speed at which a transaction can be processed and acceptance by card schemes. Among the most popular card schemes in Europe are MasterCard, VISA, American Express and Diner's Club. Widely used are the three following authorization methods:

- Swipe And Signature (SAS): The card holder swipes his card through a card reader, the information on the magnetic stripe is read and the transaction is confirmed by signature on the mobile device. This approach has the lowest technical complexity of all three but is easier to attack as the data on the magnetic stripe is not encrypted. Its main problem is that card data can be copied while the card is swiped. This method is widely deployed in the USA due to its universal acceptance by American card schemes.

- Chip And Signature (CAS): The card holder inserts his card into a card reader and the encrypted information on the card's chip is used to validate the card against the card reader. The user authorizes the transaction by signing on the smartphone's screen. As of this writing, this is the most popular method to authorize payments on mobile devices in Europe. The biggest drawback of using CAS to authorize a transaction is that Visa Europe, even though the technical complexity for CAS readers is much higher, does not consider this method secure enough and therefore requires an extended flow that requires the client to confirm the transaction with a text message. Except for Visa Europe, all other card schemes consider CAS to be sufficiently secure.

- Chip And PIN (CAP): A card holder inserts the card into a card reader and enters his Personal Identification Number (PIN) into the reader's PIN pad to authorize the transaction. Only after the correct PIN has been entered, the phone is able to read the data required to process the transaction. The main drawback of this authorization method is its high technical complexity, mainly because the reader must be equipped with a PIN pad. This incurs higher costs on both, the hardware and software. Although this process allows to authorize transactions with any card scheme, the high cost makes a widespread use for mobile payments unlikely.

Although authorization with CAS is a lengthy process with VISA cards in

Europe as each transaction needs to be confirmed via text message, it is currently the best compromise between speed, cost and an almost universal acceptance by card schemes. SumUp provides merchants with SAS or CAS readers based on the merchant's profile. This means that as of April 2013, all transactions are authorized by signature. Also with other providers of similar mobile payment services, authorization by signature remains the predominant way to authorize transactions. It is therefore of paramount importance that the authorization by signature process is as secure as possible.

We propose an automatic signature verification system based on the concatenation of an algorithm based on a signature's global features, Dynamic Time Warping (DTW) and Hidden Markov Models (HMM) to make the authorization process more reliable. As the signatures written with a finger and captured on a smartphone tend to have a high variability and unstable characteristics, our system has to account for false positive matches. A disruptive new technology like the one offered by SumUp needs to create a good user experience to be successful and it is important to avoid frustrated customers who cannot process a transaction because their genuine signature was identified as a forgery by our system. To account for that, we propose a system involving a feedback loop which guarantees that an honest card holder can always process transactions. With a feedback loop, transactions are still possible even if a card holder's signature could not be matched to previous signatures. At the same time, the feedback is used to train the signature models for future authorizations.

Going forward, we refer to the preventive part of our work as *fraud prevention* and the reactive part as *fraud detection*. We use the term *fraud protection* for findings that apply to both, fraud prevention and detection.
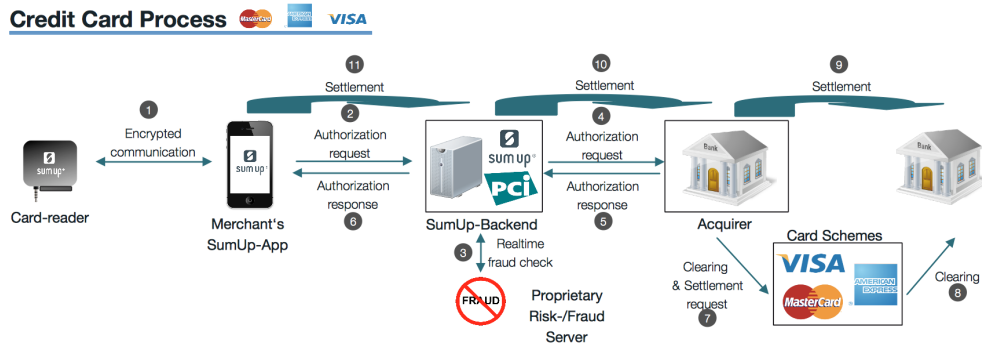
Figure 1.1: The simplified flow of a single credit card transaction starting with the SumUp CAS credit card reader

1.1 SumUp - Card Payments for Everyone

Credit cards enjoy a huge popularity among consumers which is illustrated by the number of credit card holders in the United States of America (USA): In 2008 over 176.8 million people owned a credit card with an average of 3.5 cards per card holder. About 60 percent of consumers own a rewards credit card and approximately 51 percent of the USA's population owns at least two credit cards [1].

While credit cards are also very popular with consumers in Europe, not as many businesses in Europe allow their customers to pay by credit card. This is due to the following reasons:

- A European business initially pays between 200 EUR and 500 EUR for a credit card terminal

- There is a monthly subscription fee of 20 EUR to 50 EUR

- A percentage of each transaction goes to the credit card company. Usually, merchants pay between 2.75 and 5 percent to the card payment terminal provider. Often, a minimal fee is charged for small transactions.

All these costs are much higher than in the USA and thus there is a higher entry barrier for businesses in Europe to provide the option to pay by card. It is likely that the costs are higher due to more extensive regulations in Europe which require more operational effort. Also, the CAS and CAP devices are more complex than SAS devices and the hardware costs therefore higher.

SumUp's goal is to lower the barrier for merchants to accept credit card payments by providing a professional yet inexpensive solution for anyone to accept

(1)   (2)   (3)   (4)   (5)

Figure 1.2: The five steps within SumUp's mobile application while processing a credit card transaction

card payments. The company was founded in the fall of 2011 and has enjoyed a rapid growth since.

As of April 2013, SumUp's services are used by thousands of merchants in more than ten European countries. The business has been successful because it removed two out of three of the previously mentioned monetary obstacles to a more widespread adoption: anyone who signs up with SumUp receives a free card reader for use with a smartphone and there is no monthly subscription fee. As the merchant already owns the expensive hardware in form of a smartphone and most people already pay for a data subscription, the cost on both ends is reduced substantially. This way, SumUp is able to finance itself through the 2.75 percent transaction fee. At its core, it competes with traditional credit card terminal providers who require their clients to pay a monthly fee for their terminal and the high initial charge for the device.

Instead of building and selling expensive hardware, the only required hardware — the card reader — is shipped at no charge and the mobile application is distributed for free via the Apple AppStore and Google Play Store. We refer to tablets and phones which run either iOS or Android as *smartphones* to avoid confusion with tablet input devices such as Wacom[4] tablets to which we refer as *digital tablets*.

Payment through SumUp gives taxi drivers, market traders and other small businesses, who could not afford one of the traditional payment terminals, an enormous economic advantage. Card payments can now be processed right on the spot and without an initial financial investment.

Processing a transaction with SumUp is illustrated in five simple steps in Fig 1.2.

---

[4]http://wacom.com/

0. Once the merchant has registered a SumUp Account and provided identification documents, he receives a free card reader and can immediately start to accept card payments

1. The purchase amount can be entered manually into the mobile application or via previously created products

2. Debit cards and credit cards like Visa and MasterCard are supported and are selected by clicking on their respective logo

3. After reading the card, the mobile application shows a confirmation of the card type and a masked version of the Primary Account Number (PAN)

4. The customer confirms the transaction with a signature written on the screen of the smartphone or tablet

5. After successful completion, the customer can have the receipt sent to their email account or to their phone via text message

Depending on the authorization method, a number of requests are made to the SumUp servers and from there to other entities of the transaction authorization chain, including acquirers, issuing banks and credit card institutions. The simplified mechanics behind a credit card transaction processed through SumUp are shown in Figure 1.1. Internally, a request is sent to SumUp's fraud server which performs a variety of checks and analysis to decide whether the transaction is accepted or declined. There is currently no established third party provider for fraud detection in this area and all rules are custom made. The signature is part of the data that is exchanged with the fraud server and the signature verification process, as described in Chapter 3, is an integral part of the authorization process.

Along with simplicity, quick setup time and low cost, SumUp's advantage over its competitors is that it deducts a relatively low fee per transaction of 2.75 percent. From that fee, it also pays the other parties in the value chain. To build a sustainable business model, SumUp will enable customers to also process other types of mobile payments, alongside credit card transactions. The Consumer-To-Business (C2B) credit card transactions will only be one part of all mobile payment transactions. A long term strategy is to implement all payment schemes as listed by the European Payments Council (EPC). Mobile payments may at one point replace all current payment methods for Business-to-Consumer (B2C) transactions as well as Business-to-Business (B2B), Consumer-to-Consumer (C2C) and C2B transactions. The EPC predicts this will happen due to the availability and convenience of mobile devices paired with the user's perception of having full control over it. This creates an environment of trust and convenience for conducting payments [2].

1.2 Fraud

Fraud, as defined by Phua et al. [3], refers to the abuse of a profit organization's system without necessarily leading to direct legal consequences. Fraud detection, as part of the overall fraud protection, has become one of the most established applications of data mining.

The large volume and real time nature of the transactions processed each day by SumUp makes a manual verification of each transaction impossible. As such, SumUp has to rely on automated systems to process and validate transactions. At time of this research, there is no established provider of anti-fraud software in this field. However, companies like Sift Science,[5] a company which specializes in providing a fraud control service as a third party, has recently raised more than four million USD from leading venture capital firms.

SumUp does its own internal research to implement, train and tweak its own fraud rules. An internal assessment has shown that the two most popular fraud scenarios are the use of merchant accounts for money laundry and copied or stolen cards.

Our fraud prevention mechanisms focus on preventing money laundry while our fraud detection measurements will help to identify a card holder by signature and thereby make it hard for fraudsters to use stolen cards. After a card has been used a certain amount of times, the goal is to build a reliable signature model to verify it is the same person signing the next time the card is used.

Not only the physical cards are at risk of theft, the digital copy of the credit card data also needs to be protected. This is one of the reasons SumUp only saves encrypted card information and is obliged to do so according to the Payment Card Industry Data Security Standard[6] (PCI DSS) and in an environment defined by the standard. The standard defines a set of rules to reach these control objectives:

- Build and maintain a secure network

- Protect card holder data

- Maintain a vulnerability management program

- Implement strong access control measures

- Regularily monitor and test networks

- Maintain an information security policy

---

[5]http://siftscience.com
[6]http://pcisecuritystandards.org/

This has an impact on our work on fraud protection as the regulations connected to protecting card holder data only allow storing card numbers and not names of card holders. The goal of this requirement is that even if someone would gain access to SumUp's database, it would not be possible to retrieve all information needed to process a transaction. This however, bears one disadvantage: we cannot collude information of multiple cards used by a single customer and can therefore only build a signature model based on the signatures we collect per card, not per card holder. As the majority of all US card holders holds at least two credit cards [1], being able to merge the signature databases from different cards of one card holder, could significantly increase the accuracy of the signature model.

It is impossible to be absolutely certain about the legitimacy of a transaction. In reality, the goal must be to filter out possible evidences of fraud from the available data using cost-effective algorithms in real time.

1.3 Signature Verification

Most previous work on signature detection has been done on signatures written with a pen on paper or on a digital tablet. When a signature is captured on paper, it was afterwards digitalized using a scanner or camera. The dynamic characteristics of a signature are lost in that process and the algorithms that can be applied to the signature information are very different. It is common to divide signature verification into two methods:

- **Offline signature verification** performs recognition algorithms based on static features of a signature, mainly shape and length.

- **Online signature verification** performs algorithms on the dynamic features of a signature. These analyze the speed, the acceleration, the angular acceleration, the pressure and many other local properties of a signature.

Since digital tablets, touchscreens interfaces and smartphones became affordable and widely deployed, it became feasible to capture dynamic features of a signature. In Chapter 3 we look at the common techniques of both types, offline and online, to verify signatures.

The high accuracy and resolution of current digital tablets and smartphone screens enables to capture signature data in high resolution and precision on relatively cheap devices. Traditionally, digital signatures were captured on a digital tablet with a pen. However, signatures captured by SumUp are collected on a variety of mobile devices and have fundamentally different characteristics than signatures captured on digital tablets. The main reasons for these differences are:

- The signatures are written by finger instead of a pen, which is not how people are accustomed to sign. This leads to higher variability and less stable signature models.

- Most smartphones use an event-triggered sampling in contrary to digital tables which often use a a fixed sampling rate. While the sampling rate on digital tablets is constant, the rate in signatures captured on smartphones can vary a lot within one signature.

- The signatures are captured on a lot of different devices with different screen sizes, resolutions, sensor densities, operating systems and other device specific characteristics.

As transactions need to be authorized or rejected in real time, we are limited on the number of algorithms we can use during the process and the complexity of our models and dataset. We discuss our strategies to overcome these difficulties in Chapter 3.

Fraud Prevention
PEP & HRA Detection

Fraud prevention is applied during the sign-up process of a new user with the goal to keep high risk users from gaining access to the system. Any financial institution has a high interest to implement as many anti-fraud mechanisms as possible. With the financial industry being a strictly regulated industry, many precautions are mandatory and described by the governing financial authority. Yet, more mechanisms are added on top of the mandated ones as they provide a competitive advantage.

The financial authority governing SumUp's operations is the United Kingdom (UK) Financial Services Authority[1] (FSA) and the regulatory environment is specified in the UK Money Laundry Regulations (MLR) from 2007 [4]. Chapter 2 of the MLR specifies the due diligence that has to be done for every client. As part of the due diligence process, the financial institutions are required to check that the customer does not fall into one of the two following categories:

- **Politically Exposed Persons** (PEP) are people who hold a prominent political function. In the United Kingdom this is defined as a national position. Also a PEP's spouse and children are included in this category.

- **High Risk Accounts** (HRA) are people who have previously committed a financial crime, been involved in a money-laundering related crime (e.g. dealing with narcotics) or are listed on a government watch list.

We will refer to the combination of PEP and HRA as *high risk clients*. Any new client needs to be checked against existing lists for both types at sign-up and regularly after the initial sign-up. If the algorithm returns a positive match with either a PEP or an HRA, a flag is raised in the SumUp Operations Admin Panel (SOAP) and the match has to be confirmed or falsified manually. If the

---

[1] http://fsa.gov.uk/

match can be falsified, nothing more needs to be done and the user will continue the sign-up flow. However, if there is a positive PEP or HRA match, different procedures have to be followed.

For an identified PEP match, additional due diligence needs to be done to sign up the client. Additional documents need to be collected that give detailed information about the wealth and assets of said client. After performing the additional due diligence, a client can be signed up but settlements need to be blocked. A positive match with the HRA list means that the subject cannot be signed up and a Suspicious Act Report (SAR) needs to be filed with the Serious Organized Crime Agency (SOCA). All HRAs are banned from conducting business with any financial institution.

Previous to our work, a test based on only first and last name was already in place but produced a numerous false positive matches. This required a lot of manual work to falsify matches. Our goal was to create a fast, reliable check, requiring as little manual input as possible. Chapter 4 lists our results.

In the next section we will present the data of the two databases and their origin before talking about the way we structured the lookup and implementation of our solution.

2.1   PEP & HRA Database

World-Check[2] is one provider of the PEP and HRA database. It consolidates the list by polling many different lists - including the FBI's Most Wanted List, Interpol and others.

The data is supplied in a text file as Comma Separated Values (CSV) and contains information in 26 fields per person listed. The most important fields are listed in Table 2.1. Unfortunately, the records are far from being complete. A lot of the records are missing some of the important fields like birthday or locations, which makes it hard to falsify a match automatically and sometimes a match has to be made on just first and last name correlation. World-Check normalizes the data so that all dates have the same format, city and country names are always spelled the same way and all data in a specific field is formatted the same way. The full database file is about three gigabytes in size and contains some 1.8 million records of PEPs and HRAs in total. As the database is constantly changing, there is also a daily, weekly and monthly incremental update and delete file, with which the database is kept updated without having to download the full database file every so often. World-Check is a paid service and the data is retrieved over an encrypted HTTPS connection.

---

[2]http://world-check.com/

Field	Value
uid	unique identifier within the World-Check database
last_name	subject's primarily used last name
first_name	subject's primarily used first name
aliases	other first/last name combinations the subject has used
alternative_spelling	alternative spellings of the subject's first/last name
category	used to define if subject is a PEP or HRA
age	the subject's age
dob	the subject's date of birth
deceased	information about the subject's death
locations	the cities the subject is associated with
countries	the countries the subject is associated with
further_information	description and references about the subject
external_sources	information about from which sources the information in the database is extracted

Table 2.1: Fields of the World-Check database CSV file

2.2   Algorithm

Clearly, parsing three gigabytes of raw data and rebuilding the database each time a new user processes through the sign-up flow is slow and not practical. We therefore chose to parse the needed data into a relational database, allowing us to perform queries within split seconds. The algorithm we used to make the lookup faster, consists of two parts - parsing and lookup.

Parsing the data

The raw data contains only one record per individual but often lists not only one pair of first and last name but many. The additional pairs are retrieved from the aliases and alternative_spellings fields. For faster lookup, each record from the original CSV file is parsed into multiple records in the database such that there exists one database record for each pair of first and last name combination. We create a record for each permutation of first and last name retrieved from those fields to cover all identities a high risk user might possibly use. Listing A.1 shows the algorithm used to create the cross product of all name pairs.

Lookup

To improve the speed of a lookup, the database has an index on first and last name on the PEP/HRA table to speed up the lookup. An index was also generated on the other fields used by the lookup algorithm: date of birth (dob), decease date ($deceases$), locations ($cities$) and countries.

To reduce the number of false positives, the lookup uses additional information besides the first and last name to confirm or falsify a match. For each lookup, at least the following information needs to be provided to the algorithm:

- First and last name

- Date of birth

- City

- Country

The algorithm is outlined in Listing 2.1. Its output is a set of Know Your Customer[3] (KYC) action and statuses which flag the customer in the system and may restrict her from certain actions, e.g. processing transactions with her account.

---

[3]http://fsa.gov.uk/smallfirms/resources/factsheets/pdfs/KYC_table.pdf

Listing 2.1: The Lookup algorithm

```
def lookup fname, lname, dob, city, country

    # collect all hits on first and last name
    all_hits = verified_hits = []
    all_hits << PEP.find_by_first_name_and_last_name(fname, lname)
    all_hits << HRA.find_by_first_name_and_last_name(fname, lname)

    for record in all_hits

        # skip if peson is already deceased
        next if record["deceased"]

        # parse dob. if the dob in the db is incomplete (eg
           1971/02/00)
        # only check the complete parts
        y = record["dob"].year  != 0 ? record["dob"]["year"]  :
           null
        m = record["dob"].month != 0 ? record["dob"]["month"] :
           null
        d = record["dob"].day   != 0 ? record["dob"]["day"]   :
           null

        # now skip if one of the values exists but does not match
        next if y && y != dob["year"]
        next if m && m != dob["month"]
        next if d && d != dob["day"]

        # at this point, dob matches or was not given
        # now, check city
        next unless record["locations"].contains? city

        # and country
        next unless record["countries"].contains? country

        # all checks passed, add this record to the result set

        verified_hits << record

    end

    # now set KYC status for the proven records
    for r in verified_hits
        set_kyc_status r
    end
end
```
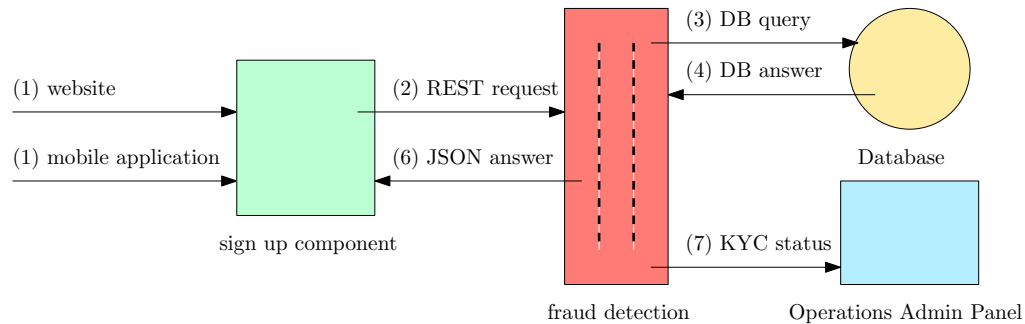
Figure 2.1: The simplified architecture and strucutre of a PEP/HRA check request

2.3   Implementation & Architecture

The PEP & HAR check was implemented as a Sinatra[4] Ruby Applicaton, accessible through a REST interface. It fulfills multiple purposes: It parses the initial or incremental database file, it continuously rechecks all users in the database versus the updated database and also answers requests from the sign-up component.

It uses Ruby ActiveRecord (AR)[5] to connect to a relational database. The Ruby Sinatra Application runs within a Linux environment and the Linux Crontab Tool[6] is used to run the first two tasks in a constant interval:

- Each day, the incremental update and delete file are downloaded from World-Check to keep the database up to date.

- On a weekly base, each user is screened against the updated database and new hits raise a flag in the operations team's panel for further inspection.

The third task, answering requests from the sign-up flow is done via a REST interface. Chapter 4 goes into greater detail of how the requests are structured. The system topology is shown in Figure 2.1.

A request is triggered and answered with the following steps:

1. A new user signs up from one of the mobile apps or the website

2. The sign-up component sends a request for the new user to the fraud detection component

3. The component retrieves the necessary data from the database

---

[4] http://sinatrarb.com/
[5] http://guides.rubyonrails.org/active_record_querying.html
[6] http://unixhelp.ed.ac.uk/CGI/man-cgi?crontab+5/

4. The database delivers all matches based on first and last name

5. The Lookup algorithm is performed within the fraud detection component

6. The result is returned to the sign-up component with information of whether or not the user can continue the sign-up flow

7. The respective KYC actions and statuses are set in the database.

CHAPTER 3

Fraud Detection Signature Verification

For centuries, being able to authenticate someone based on biometrics, has been important to identify each other, in order to conduct business and verify the authenticity or grant access to resources. In order to guarantee someone's identity, it has always been important to work towards more exact and more reliable means to measure someone's biometric properties and to match recorded properties against confirmed samples, while at the same time minimizing the chance of an erroneous accept or reject. The characteristic properties of a handwritten signature help to prove a signer's identity. A signature has four legal properties [5]:

- **Authentication**: Signature verification allows to confirm a signer's identification

- **Acceptance**: By signing a document, the writer conveys a willful intent and acceptance of the document's terms and contents

- **Integrity**: By signing a document, the signer establishes the integrity of the signed document and that it has not been altered

- **Non-repudiation**: The above three factors make it impossible for the signer to deny having signed the document

Signature verification is a particularly important biometric identification process since the signature is a widely accepted method for endorsing financial transactions [6]. As the signature is recorded during the SAS and CAS authorization process, it makes sense for SumUp to use this information to improve the security of both authorization methods.

We present existing methods before describing the signature verification methods used in our work and the peculiarities for signature detection in mobile payments. Traditionally, detection methods can be categorized as either global or

local methods. We describe both approaches in Section 3.1 and Section 3.2. As a combination of feature- and function-based approaches has been providing better results than the individual techniques [7], we combine both approaches in our method to verify signatures.

3.1   Global Systems

Global systems, also called feature-based systems, are characterized by the fact that the feature vector consists of measurements that are based on the signature as a whole. The features are either analyzed as a whole or separately for each dimension. Often used global features include:

- Signature length

- Total time to sign

- Maximum and average velocity

- Maximum acceleration

- Total points recorded

- Number of segments

- Signature Height (H), Width (W) and W to H-Ratio

- Number of points with positive x/y velocity

A lot of research has been done in this area and the features can be simple measurements as those listed or more complicated characteristics obtained through techniques like the discrete Wavelet Transform [8], the Hough Transform [9], horizontal and vertical projections [10] or smoothness features [11].

To compare two signatures with a global system, the features may be compared as

- absolute values

- relative values in respect to another feature

- averaged values over the whole signature or key parts of it

- maximum or minimum values

- values of the standard deviation

A benefit of comparing signatures based on global features is that they are generally faster to compute than local features and simple to store and compare because they can be represented by a number and don't require data structures.

3.2   Local Systems

Function-based systems, also called local systems, are characterized by the fact that the feature vector consists of measurements on individual points or clusters of points. The most popular methods are Dynamic Time Warping and Hidden Markov Models. Often used local features include:

- Pressure

- Horizontal (x) and vertical (y) position

- Path tangent angle

- Velocity and acceleration in a particular point

- Log radius of curvature

- Pen elevation and pen azimuth; In our case this data is not available as the signatures are written by finger

Typically, a function vector has fewer, or at most as many, dimensions as the signature vector. Matching two signatures requires finding an algorithm to match either

- two feature vectors

- the vector of the test signature to a reference vector or model

Researchers have tried various techniques to match two feature vectors [12]. Among these techniques are: Dynamic Time Warping [13, 14], Hidden Markov Models [15], directional PDF [16], stroke extraction [17], synthetic discriminant functions [18], granulometric size distributions [19], neural classifiers [20], wavelets[21], grid features[22] and elastic matching[23].

Before trying one of these advanced techniques, one might try to compare two feature vectors in a simpler manner. The simplest approach of comparing two feature vectors element by element is to use linear correlation [24] and calculate the distance between each pair.

This approach has two major drawbacks:

- It only works if the two vectors have equal length. As signatures always vary a bit, it is rarely the case that two signatures and thus the two feature vectors have equal length.

- Even if the overall path of the signature is very similar, there often are parts that are distorted or additional in one of two signatures. Linear correlation is not able to account for local distortions and would thus create a bad score even if the beginning and end of the signature's path were a perfect match.

As both phenomena are very typical for signature verification, more advanced techniques were developed. Our work concentrates on Dynamic Time Warping and Hidden Markov Models as these two techniques have proven to be most successful — especially when combined — in previous work [7, 25, 26, 12, 27].
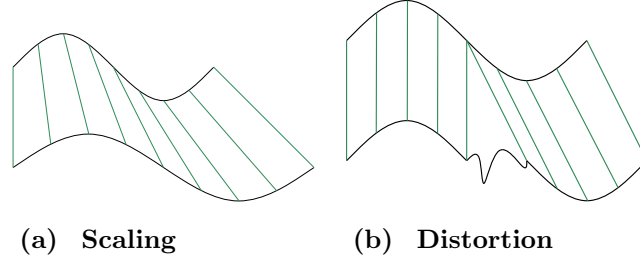
(a) **Scaling** (b) **Distortion**

Figure 3.1: Two conceptual drawings show how DTW deals with the two main problems in signature recognition. The time series on the left is stretched but still matched thanks to DTW's ability to account for the scaling of segments or the signature as a whole. The drawing on the right is a representation of how DTW is still able to match two sequences even if one is distorted.

3.2.1 Dynamic Time Warping

Dynamic Time Warping is a dynamic programming algorithm to measure the similarity between two time series which may vary in time or speed. This has been used for speech recognition and can also be used for signature detection to cope with non-linear time distortions which can be seen in the signals because a signer does not always sign with the same speed. It has shown to be a much more robust distance measure than the Euclidean distance in a simple linear correlation [28, 29, 6] due to its ability to match similar shapes even if they are out of phase in the time axis.

Koegh et al. [30] showed that the mean error rate average over 1000 runs for DTW was an order of magnitude lower than the error rate for the Euclidean distance. However, the DTW algorithm also took approximately 230 times longer to evaluate than the Euclidean distance. It was applied to signatures in 1977 by Yasuhara and Oka [31] who concluded that it is a very useful approach for online real-time signature verification. Yasuhara and Oka used an adaption of the algorithm that was originally proposed by Sakoe and Chiba [32] and improved the algorithm for the use on signature data.

As Figure 3.1 shows, DTW accounts for the two main difficulties when comparing two time series - stretching and distortion. This is of particular importance for our work, as we collect signatures on different devices with different sampling rates and even if a signer signs with the same speed but on two different devices, the two time series can significantly vary in length. Additionally, the different devices have different touchscreen sizes and surfaces and it is expected that people will not always complete the signature within the time due to different friction forces between finger and touchscreen and length of the signature path.

Classification is done by computing the DTW distance $dtw[s][t]$ between

the model signature s and a test signature t which leads to a DTW score. If the score is below a certain threshold, we will consider the two signatures to match.

Training is done by computing the distance measure $dtw[n][m]$ for all signatures n, m in the set of signatures for a certain user and selecting the signature s with the smallest distance to all other signatures. While there are other techniques, which allow to build and train a signature model based on all the reference signatures, DTW can only compare two time series at once. Our strategy is to compute the DTW score for a test signature with each signature in the card holder's database and apply majority voting to decide if it's a forged or genuine signature.

Algorithm: We have two signature vectors X, Y containing data for each point of the signatures:

$$X = x_1, x_2, ..., x_i, ..., x_I$$
$$Y = y_1, y_2, ..., y_j, ..., y_J$$

The distance between two vectors x_i, x_j is defined as the 2-norm:

$$d(i, j) = ||x_i - y_i||$$

We define a warping path C as

$$C = c_1, c_2, ..., c_k, ..., c_K$$

where each element c_k corresponds to a combination (x_i, y_j).

The algorithm spans an $I \times J$ matrix G between the two signature vectors. The matrix is initialized with

$$g_1 = g(1, 1) = d(1, 1) \cdot w(1)$$

where g_k is the accumulated distance after k steps and $w(k)$ is a weighting function that has to be defined. In each iteration, g_k is computed as

$$g_k = g(i, j) = \min_{c_{k-1}}[g_{k-1} + d(c_k) \cdot w(k)]$$

until both Signatures X, Y have been traversed.

The normalized distance of the two signatures is therefore:

$$D(X, Y) = \frac{g_K}{\sum_{k=1}^{K} w(k)}$$

where $\sum w(k)$ compensates the effect of the length of the sequences.

The definitions of weighting factors w_k defines the matching between the two signatures. The most common definition in literature is one where three types
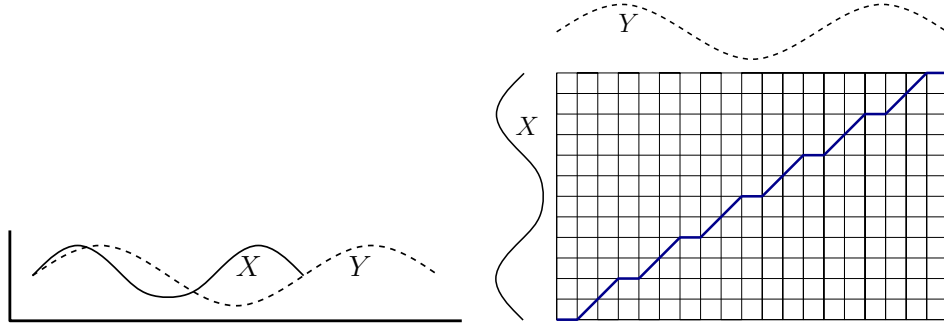
Figure 3.2: Two slightly distoreted, strechted time series X, Y can still be matched with the DTW algorithm as the almost perfectly diagonal warping path shows.

of transitions - deletion, match and insertion - are allowed. The resulting g_k becomes:

$$g_k = g(i,j) = \min \begin{bmatrix} g(i, j-1) + d(i,j) \\ g(i-1, j-1) + 2d(i,j) \\ g(i-1, j) + d(i,j) \end{bmatrix}$$

The first case corresponds to the case of insertion, the second to a match and the third to a deletion. If there is a match in each step, the path will go along the matrix' diagonal. Generally, if the path is close to the diagonal, the signatures are similar and the score is thus small.

Even though the DTW algorithm has been outperformed by more powerful algorithms like HMMs or SVMs in speech detection, it remains very effective in signature detection as it deals well with small amounts of training data. This makes it an especially good fit for signature detection where the database of new users is usually sparse.

In general, DTW is known to have two drawbacks in signature verification:

- heavy computational load

- warping of forgeries

DTW causes heavy computational load because it does not obey the triangular inequality and thus indexing a set of signatures takes a lot of time. It has complexity $\mathcal{O}(N^2)$ where N is the signature length. As soon as the pool of signatures for a signer gets bigger, the computation costs rise because the test signature has to be compared to each of the signatures in the pool of confirmed signatures. Eamonn Keogh et al. [30] presented a lower bounding method to index all samples without comparing each of them to each other. This is important when we analyze signatures in real time.

If a forgery is very close to the original, DTW might match it and not recognize it as different enough as it is one of the algorithm's strengths to account for scaling and minor distortions. This second drawback can be addressed by looking at how straight or bended the warping path is. A straight warping path indicates that a genuine signature is more likely whereas a curvy warping path indicates a forgery. Work on this has been done by Y. Sato et al. [33] but makes comparison between different signatures more difficult because it introduces another dimension and thus made computation harder and has hence not found widespread use.
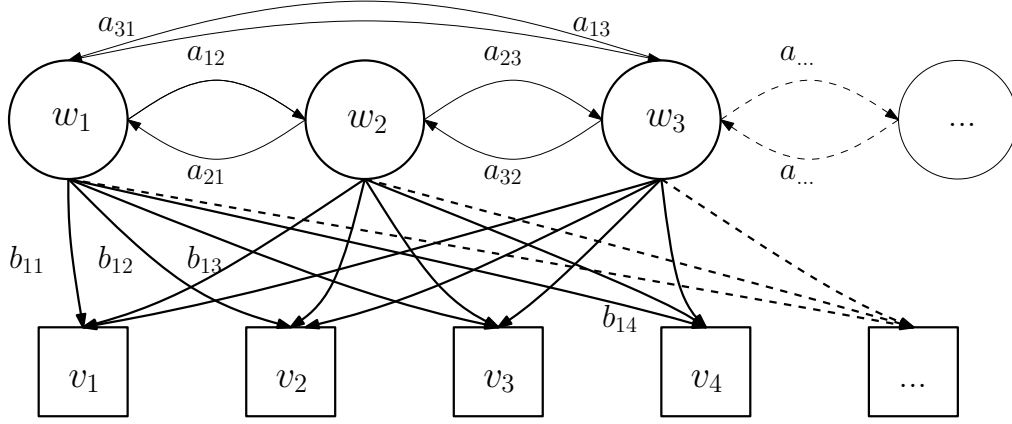
Figure 3.3: General representation of an HMM with states $w_1, w_2, w_3, ...$, observations $v_1, v_2, v_3, ...$, the state transition probabilities a_{ij} and the observation emission probabilities b_{ij}

3.2.2 Hidden Markov Models

A Hidden Markov Model is a stochastic process with an underlying Markov Model. Unlike with classical Markov Models (MM), the states of a Hidden Markov Model cannot be directly observed. Only the symbols emitted from model's states may be observed. If the signature is interpreted as a succession of observed symbols, then we can, given an HMM, calculate the probability of each path through the states in the model. By training the HMM to match the signatures that are known to be genuine we can thus reconstruct a probability of the user matching.

An HMM, as illustrated in Figure 3.3, is defined by:

- N hidden states $w_1, w_2, ..., w_N$

- the alphabet of M of symbols $v_1, v_2, ...$

- the $N \times N$ transition matrix A, which fulfills

$$\forall i, \qquad \sum_{j=1}^{N} a_{ij} = 1$$

  and whose elements are defined as

$$a_{ij} \geq 0, \qquad a_{ij} = P(w_j(t+1)|w_i(t)), \qquad 1 \leq i, j \leq N$$

- the $N \times M$ emission matrix B, which fulfills

$$\forall k, \qquad \sum_{k} b_j(k) = 1$$

and whose elements are defined as

$$b_j(v(t)) = P(v(t)|w_j(t))$$

- the initial distribution π

The underlying Markov Model of an HMM is modeled depending on the application. The most common types are shown in Figure 3.4:

- Left-To-Right HMMs (LTR HMM) are characterized by the fact that from each state only the same state or a state more to the right can be reached. Once a state is left, it is never reached again. Depending on the application, no skipping of states (top left), skipping a single step (top right) or skipping an arbitrary number of states may be allowed (mid left). Which states can be reached from which other states is defined by the initial transition matrix. We define our transition matrix in Equation 3.1 as an LTR model where no states can be skipped.

- Parallel HMMs have parallel paths, only the states belonging to the same path can be reached once a path is chosen. They have similar properties to LTR Markov Models with the exception that there are multiple paths.

- Ergodic HMMs are the most generic form of Markov Models where a transition is possible from any state to any other state or the same state. The transition matrix of an ergodic Markov Model has only non-zero elements.

In speech and signature recognition, a left-to-right Markov Model has proven to be a good choice. This can be interpreted as that a signer will never go back in his signature, which corresponds to the left-to-right analogy in the time series that corresponds to the signature.

While HMMs with a small set of states and observations perform bad because they are too simple, too many states and observations make the model computationally heavy and accuracy is reduced because of over-fitting.

The following three problems and their solutions is what makes HMMs so useful for real world applications:

1. **Evaluation**: Estimation of the optimal sequence of hidden states given the model parameters and observed data. This problem is solved by applying the Forward-Backward algorithm to the model.

2. **Decoding**: Calculation of the likelihood of the data given the model parameters and observed data. This is often solved with the Viterbi algorithm.
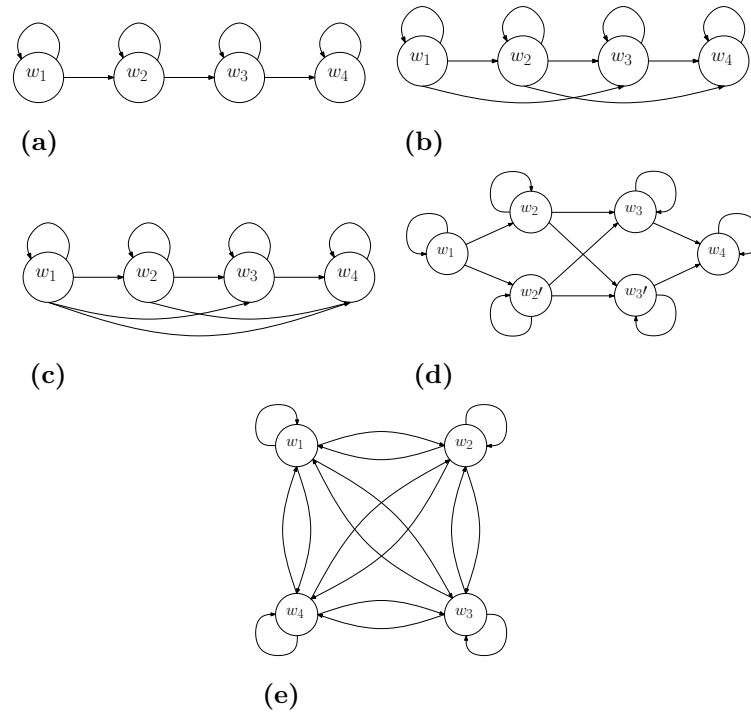
Figure 3.4: Different types of HMMs: (a) LTR HMM where skipping of states is not allowed, (b) LRM with skipping of one step, (c) LRM with allowed skipping, (d) Parallel HMM, (e) Ergodic HMM

3. **Training**: Estimation of the model parameters given just the observed
   data. The Baum-Welch algorithm is used to estimate the model parame-
   ters.

We are interested in the evaluation of an observed sequence to match a test
signature to one of the signature models and we are interested in the training
to train the signature models with the signatures we collect from a card holder.
We follow previous authors who used the discretized angle between two points
of a signature as the symbols. Based on those symbols, we train our model using
the Baum-Welch algorithm. After training signature models, we calculate the
likelihood of a signature belonging to a certain card holder by using the Forward-
Backward algorithm to compare the signature to this card holder's signature
model. A detailed description of all three problems and their solutions is given
by Rabiner et al. [34].

In our case, we chose to work with an LTR HMM that does not allow skipping
any states. This leads to a transition matrix A of the form:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \ldots & 0 \\ 0 & a_{22} & a_{23} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & a_{(N-1)(N-1)} & a_{(N-1)N} \\ 0 & 0 & 0 & 0 & a_{NN} \end{bmatrix} \tag{3.1}$$

We also chose to work with a LTR HMM that does not skip any states and
we chose π such that the system starts in the first state:

$$\pi = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$$

The symbols are calculated as the discretized angle between two points in a
signature as illustrated in Figure 3.5. We use an alphabet of 4, 8 and 16 symbols
in our experiment as described in Section 4.2.

Evaluation: The Forward-Backward algorithm is a recursive solution.
Before looking at the algorithm in detail, we discuss a direct approach to evaluate
the probability. A first approach to evaluate which model has produced a certain
set of observations might be the following:

1. For all possible observation sequences, calculate the probability of going
   from the first to the last state while emitting that sequence

2. These probabilities sum up to the total probability $P(V^T)$

$$P(V^T) = \sum_{r=1}^{r_{\max}} P(V^T|w_r^T)P(w_r^T)$$
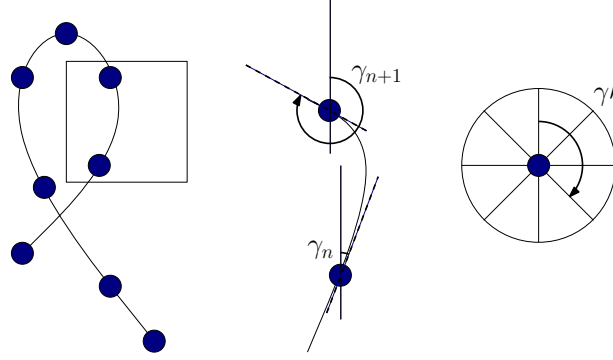
Figure 3.5: Our symbols are defined as the discretized tangent angle. The left image represents a close up look of a signature and the dots that define it. The drawing in the middle shows a zoomed version of the boxed segment on the right and the tangent angle for the two dots in this area. The right drawing shows the discretized $\gamma\prime$ for an alphabet of 8 symbols.

3. For all possible sequences in $w_r^T = w(1), w(2), ..., w(T)$ within T time intervals we get

$$P(V^T) = \sum_{r=1}^{r_{\max}} \prod_{t=1}^{T} P(v(t)|w_(t))P(w(t)|w(t-1))$$

The problem with this approach is its time complexity of $\mathcal{O}(N^T T)$ which makes it unfeasible even for small values of N and T. E.g. for $N = 5$ States and $T = 100$ observations, there are on the order of $100 \cdot 5^{100} \approx 10^{72}$ computations.

Clearly, a more efficient method is required. Fortunately, there is The Forward-Backward algorithm is a recursive approach to approximate the solution.

1. We define the forward-backward variable which is the probability that a model is currently in state i and has already produced the first t elements of V^T

$$\alpha_i(t) = \begin{cases} \pi_i b_i(v(0)) & \text{if } t = 0 \\ \sum_{j=1}^{N} \alpha_j(t-1)a_{ij}b_j(v(t)) & \text{if } t \neq 0 \end{cases}$$

2. For $t = 0$ we initialize a

$$\alpha_i(0) = \pi_i b_i(v(0))$$

3. We compute α for all states i at all times t

4. The total probability that a model M generated the sequence V^T results from the sum of all $\alpha_i(T)$

$$P(V^T|M) = \sum_{i=1}^{N} \alpha_i(T)$$

This algorithm is only of complexity $\mathcal{O}(N^2T)$ which results in our example to $5^2 \cdot 100 = 2500$ computations. Compared to the $10^7 2$, a saving of about 69 orders of magnitude — as long as the number of states is reasonabily low.

In a similar way, we can consider a backward variable $\beta_i(t)$ defined as

$$\beta_i(t) = \begin{cases} 1 & \text{if } t = T \\ \sum_{j=1}^{N} \beta_j(t+1)a_{ij}b_j(v(t+1)) & \text{if } t \neq T \end{cases}$$

The backward variable β defines the probability of the partial observation sequence from $t+1$ to the end. In analogy to α, we get the same computational complexity to compute β.

Training: The Baum-Welch algorithm is used to train signature model. The problem to adjust the model parameters (A, B, π) to maximize the probability of the observed sequences is by far the most difficult of the three. There is no analytical way to solve for the model which maximizes the probability. We can, however, find a local maximum with an iterative procedure such as the Baaum-Welch algorithm. The algorithm's output is the transition matrix A that is most likely to generate the given sequence of observations. The following needs to be known about the model M to start the algorithm:

- the number of hidden states N

- one or more observations sequences $V_1, V_2, ...$

- the initial transition Matrix A, that defines the HMM's structure

- the initial distribution π

In order to get to an iterative solution, we proceed as follows:

1. We first define the probability of being in state w_i at time $t-1$ and being in state w_j at time t as

$$\xi_{ij}(t) = P(w_i(t), w_j(t+1)|V^T, M) = \frac{\alpha_i(t)a_{ij}b_j(v(t+1))\beta_j(t+1)}{P(V^T|M)}$$

$$= \frac{\alpha_i(t)a_{ij}b_j(v(t+1))\beta_j(t+1)}{\sum_k = 1^N \sum_l = 1^N \alpha_k(t)a_{kl}b_l(v(t+1))\beta_l(t+1)}$$

2. By summing over j we get the probability to be in state w_i at time t as

$$\gamma_i(t) = \sum_{j=1}^{N} \xi_{ij}(t)$$

3. With that, we can calculate the number of transitions from w_i to w_j as

$$\sum_{t=1}^{T-1} \xi_{ij}(t) \qquad (3.2)$$

and the expected number of transitions to any other state as

$$\sum_{t=1}^{T-1} \gamma_i(t) \qquad (3.3)$$

4. And we get the expected number of times in one state from

$$\sum_{t=1}^{T} \gamma_i(t) \qquad (3.4)$$

5. We get the expected number of times in state w_i and observing symbol k as

$$\sum_{t=1}^{T} \gamma_i(t) b_i(k) \qquad (3.5)$$

6. With that we can calculate:

  - Expected Frequency (number of times) in state w_i at time $t = 0$
    $\overline{\pi_i} = \gamma_i(0)$
  - Approximation of a_{ij}
    $$\overline{a_{ij}} = \frac{(3.2)}{(3.3)} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$
  - Approximation of $b_j(k)$
    $$\overline{b_j(k)} = \frac{(3.5)}{(3.4)} = \frac{\sum_{t=1}^{T} \gamma_i(t) b_i(k)}{\sum_{t=1}^{T} \gamma_i(t)}$$

7. Through repeated execution of this algorithm, we find the local maximum.

3.3   Challenges in Signature Detection Specific to Mobile Payments

A lot of work has been done on signature detection on offline and online signatures. However, almost all work on dynamic signatures has been done on signatures that were written using a pen on a digital tablet or paper. In our case, signatures were captured on a wide range of different devices and with a user's finger instead of a pen.

3.3.1   Signatures are written with Finger

Our experience shows that people are not used to write their signature with their finger and the first few times they sign, their signature differs a lot. However, after just 10-15 times, the signature's shape stabilizes.

This means that it will be a lot harder to detect signatures based on the first few signatures than on later signatures, once a user got used to signing with her finger. We notice that a card holder's signature characteristics stabilize once used to signing with a finger. We therefore prefer most recent signatures to older ones when we apply the algorithms implemented in Chapter 4.2

3.3.2   Sparse Initial Dataset

Although SumUp is live in over ten countries as of April 2013, it is still only used in a relatively small set of locations and we are therefore unlikely to gather a lot of data about a single user until the concept becomes more often deployed and used.

This means that we have to try and find a solution that works reasonably well with a small amount of initial data per user. The problem persists even after this initial phase as new people will continue to join and always start with an empty set of signatures. We therefore test the algorithms developed in Chapter 4.2 not only on the whole database for a card holder but also on small subsets.

3.3.3   Device & Software Fragmentation

Unlike signatures captured on a digital tablet, our database of signatures is captured on a variety of devices with different properties. There are various factors that have an influence on the digital representation of the signature:

- Different Manufacturers: Both, iOS and Android devices use a variety of manufacturers for their handsets and the built-in touch screens.

- Screen Size: the screen diagonal of current smartphones typically ranges between 3 and 6 inches, those of tablets typically range between 7 and 11 inches. A consequence is that the user might not only sign slightly differently but also that the signature will consist of more data points and it will take users a longer to sign on a larger screen.

We normalize the signature data before applying the algorithms in Chapter 4.2 to account for these device-specific differences.

3.4   Feedback Loop

Due to the mentioned challenges, our systems are likely not to perform as well as the same systems on traditional signatures captured by pen on a digital tablet. We are likely to get results with a higher Equal Error Rate (EER) which would hurt SumUp's business since we declined transactions because of false positive matches.

To eliminate the risk of blocking transactions because of a false positive match, we propose to use a feedback loop along with our signature verification system.

A simple approach to use signature verification is shown in Figure 3.6a:

- The mobile device sends the signature data along with a transaction request to SumUp's backend

- The signature data is analyzed and compared to this user's signature model to generate a score

- The score is compared to a threshold for this user's signature model and the signature is classified as genuine or forged

- Based on the outcome the signature is either accepted or declined

This means that a user has no way of processing a card transaction if the signature does not pass the threshold that is generated based on his previous signatures. To overcome this problem, we suggest to use a feedback loop as shown in 3.6b. Instead of declining a transaction, a request is sent to the merchant to confirm the identity of the card holder. If the merchant is able to confirm the card holder's identity, the transaction is processed. If the merchant is unable to confirm the identity, the card holder is reported as he is most probably a stolen or copied card.

Signature Detection Component
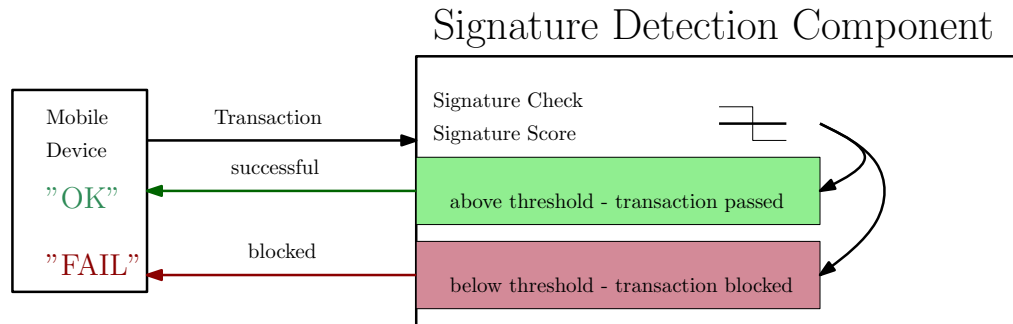
(a) **Simplified illustration of processing a transaction without the feedback loop. When the transaction object is received, the signature is extracted and analyzed. The score is compared with a threshold and if too low, the transaction is blocked. In case of a false positive match, this transaction is lost which we need to avoid.**

Signature Detection Component
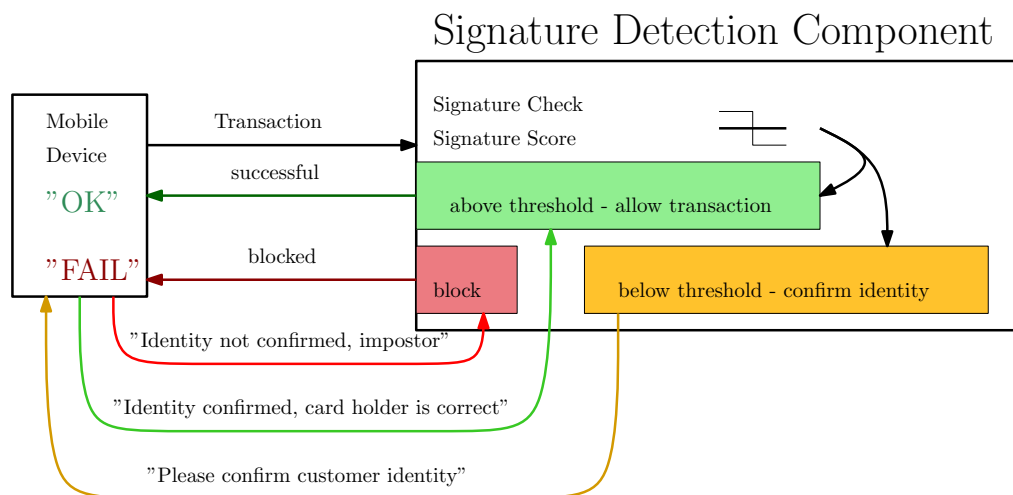
(b) **Simplified illustration of processing a transaction with the feedback loop. If the signature check does not reach the acceptance level, a request is sent to the merchant to confirm the card holder's signature. After a positive confirmation, the transaction is processed and otherwised blocked.**

Figure 3.6: A traditional signature detection system based on a threshold (a) and our proposed system that includes a feedback loop (b)

Implementation and Evaluation

Our work on fraud prevention was implemented and released to SumUp's production environment. Section 4.1 describes the implementation's setup and performance. We show how the new check's performance has affected the work of SumUp's operations team and to which extent we were able to reduce the amount of false positives.

We discuss our fraud detection system in Section 4.2. We introduced the database used for our experiments and give a detailed description of the performance of the separate algorithms used. We also discuss the technical feasibility of each approach and the fusion of all three scores.

Method	Path	Parameter
POST	/person/check	{"params": {"user_id": "1"}}
POST	/person/check	{"params": {"business_owner_id": "1"}}
POST	/person/check	{"params": {"business_officer_id": "1"}}

Table 4.1: The requests received by the PEP/HAR component

4.1 Preventive Fraud Detection

The algorithms described in Listing A.1 and Listing 2.1 were implemented in a Ruby Sinatra application with a REST interface and the application was deployed to production six weeks prior to the submission of this report.

4.1.1 Setup

The full World-Check PEP/HAR database was parsed in a testing environment using the algorithm in Listing A.1 and the parsed data was copied into the production database. The daily, incremental updates are downloaded from the World-Check servers into the production environment and are directly parsed into the production database.

The PEP/HRA check is performed automatically or manually in at least three occasions:

- **Automatically** for each new user during sign-up

- **Automatically** for each existing user at the end of each week to re-check each account after incorporating the incremental updates of that week

- **Manually** from the SumUp Operations Admin Panel (SOAP) via a button

Each request is triggered via a POST request. Depending on the user type to be checked, the body of the request changes. Table 4.1 lists the different request types.

4.1.2 Results

The new component was released to production at beginning of calendar week 10 on March 4th, 2013. At the same time, the old check was discontinued. Every new user since then was automatically checked during the sign-up process. In addition to that, Business Officers and Owners were checked manually through

SOAP by the Operations Team. The following data refers to checks during the sign-up process and excludes data from weekly re-checks or manual requests.

As SumUp is still growing rapidly, all numbers were normalized in respect to 1000 new sign-ups per week. The number of manual work by the operations team could be greatly reduced thanks to a much lower rate of false positives and therefore less matches which have to be manually confirmed or rejected. Figure 4.1 shows how the number of matches declined in calendar week 10 after the release of the new component. The high amount of false positive matches which had to be checked by the operations team caused a backlog which is clearly visible by the shifted red curve in Figure 4.1. Due to the high amount of manual work that was required to reject or approve a match, the false positives were often only rejected — and thereby marked as a false positive match — one or two week after the initial match.

Typically, about 50 to 100 matches were generated per 1000 new users with the old component. Almost all of those were false positives and therefore caused unnecessary manual work. With the new component, we were able to drastically reduce the amount of manual work required. After introduction of the new check to the system, the typical amount of matches per 1000 new sign-ups after the release of the new component declined to below ten and the amount of false positive matches was less than two. The higher number of false positive matches in week 13-10 was influenced by the backlog of previous weeks but it quickly stabilized from week 13-11 ongoing as shown in Table 4.2. The peak in calendar week 14 was caused by a system migration in the course of which a lot of accounts were checked again. However, as these accounts had been previously rejected, they did not cause any additional manual work.

The numbers in Table 4.2 show that the number of false positive matches could be reduced from 50-80 per 1000 sign-ups to less than two which corresponds to a reduction of over 96 percent.

SumUp's operations team reports the reduction had a significant impact on their work. The amount of work hours per week in this area has declined from more than seven to less than one hour on average as shown in Table 4.3. These are absolute numbers and do not account for the continued exponential growth of new sign-ups. The amount of work hours in this area has been almost doubling every month before the release of the new component in march.
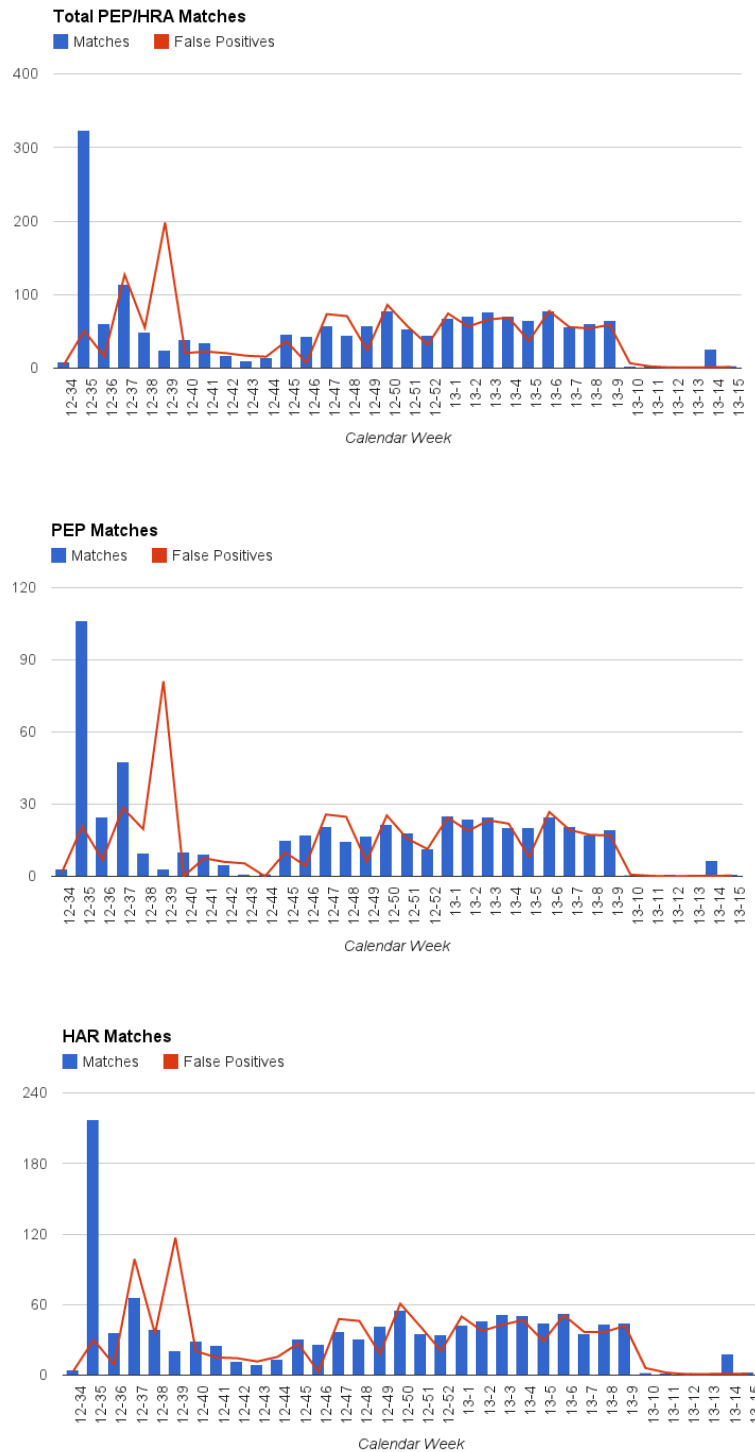
Figure 4.1: Evolution of the number of matches generated by the automatic PEP/HRA check. The top graph shows the evolution of the overall matches and false positives and the bottom and bottom graph the evolution for PEP and HRA separately. The new check was released to production in week 13-10 which is clearly visible as a drop in numbers.

	Week	Matches [per 1000 sign-ups]	False Positive Matches
after release	13-15	3,29	1,39
	13-14	25,36	1,40
	13-13	1,77	0,98
	13-12	2,84	0,95
	13-11	2,20	2,42
	13-10	2,64	6,71
before release	13-09	64,33	58,91
	13-08	60,71	53,99
	13-07	55,89	55,89
	13-06	77,83	77,83
	13-05	65,10	36,86
	13-04	71,00	68,73

Table 4.2: The requests received by the PEP/HAR component. Ratio: Erwaehnen von backlog! false positive beinhalten backlog

Month	Matches (absolute numbers)	Man Hours
December	283	10 hours
January	463	16 hours
February	853	29 hours
March	125	4 hours
April 1st - 14th	9	0.3 hours

Table 4.3: The requests received by the PEP/HAR component

Device	Software Version	Screen Size [cm]	Screen Resolution [px]	Pixel Density [ppi]
Apple iPhone 4s	iOS 6.1.2	8.9	640x960	326
Apple iPhone 5	iOS 6.1.2	10	640x1136	326
Samsung Galaxy Note II	Android 4.1.1	14.1	720x1280	267
Apple iPad mini	iOS 6.1.2	20	768x1024	163

Table 4.4: The four devices used to collect signatures, ordered by screen size

4.2 Reactive Fraud Detection

4.2.1 Database

As there is no publicly available database of signatures written by finger and captured by a smartphone's touch screen, we created a database for the purpose of this research. Between eight and 80 Signatures per person were collected from ten people on four different days on four different devices. This approximates the reality in terms of how many signatures SumUp has available per card holder and the devices they were collected with. In total, a set of 487 signatures was collected. The devices used to collect the signatures are listed in Table 4.4 and were all used equally often.

We also collected three forgeries for each person. The forgeries were created by two untrained persons and are of the following types:

- **Spontaneous Forgery** $\mathcal{F}_1$: A random signature from one card holder's signature set was shown to the impostor during five seconds. The impostor was allowed to start the forgery after the five seconds when he had no access to the signature anymore.

- **Sample Forgery** $\mathcal{F}_2$: A random signature from one card holder's signature set was shown to the impostor for an indefinite amount of time and the impostor was able to look at the signature while forging the signature.

- **Knowledgeable Forgery** $\mathcal{F}_3$: The impostor had unlimited access to a card holder's full signature set while creating the forgery.

For each technique described in Section 4.2.3 - 4.2.5, we gathered statistics about two use cases:

- **Classification of signatures**: This approach's goal is to measure how well the algorithm performs in connecting a signature with its card holder. In this part we run the ground truth against itself to train the algorithm and adjust its parameters.

- **Detection of forgeries**: Using the previously obtained parameters, we run the algorithm against the database of forgeries to see which of the forgeries are correctly detected. This corresponds to SumUp's use case of our fraud detection work.

All experiments were conducted on a current Mac OS/Unix system and run within a single thread. It is expected that the time needed to compute results could be reduced by using parallel execution of the algorithms. SumUp's server hardware exceeds this Mac OS/Unix system's computation power. We can assume that the execution times are even shorter on the production system.

4.2.2 Pre-Alignment and Normalization

In order to compare the signatures, we normalized some of the properties before comparing the signatures. To account for the fact that the signatures were collected on devices with specific characteristics, the following parameters were normalized:

- **Sampling rate**: Because we collected the signatures on touchscreen devices with event-based sampling, we recorded very different sampling rates between devices but also within the data for one signature. Typically, signatures collected on iOS devices were originally sampled at a rate of 40 Hz and signatures collected on the Android device at 60 Hz. In order to compare signatures, we normalize the sampling rate to 20 Hz for all signatures which equates about half of the typical sampling rate of a signature collected on an iPhone 4s.

- **Size and length**: The different screen sizes and pixel densities caused different height, width and length properties. In order to compare the signatures, we normalized the signatures height and width to fit within a 200 pixel by 300 pixel rectangle. The width-to-height ratio was recorded before the normalization as it is used as a global feature in Section 4.2.3.

- **Time**: As bigger screen sizes also means longer distances to draw a signature, we adjusted the timestamps when we normalized the size of a signature. We referred to the normalized value when we used time in Section 4.2.3.

4.2.3 Global Features

Our implementation of a signature verification system based on global features uses a subset of the proposed features in Section 3.1. Our system relies on the following features:

- Signature length l in pixels

- Total time to sign t in milliseconds

- Maximum and average velocity $v_{\max}, \bar{v}$

- Maximum acceleration $a_{\max}$

- Number of segments n

- Width to height ration r

Other proposed features, e.g. the total number of dots recorded, are irrelevant for our experiments as we normalize those features in order to compare signatures captured on different devices. Figure 4.2 shows our results for each card holder's signature set. We list the minimum (Table 4.2a), average (Table 4.2b) and maximum (Table 4.2c) value for each feature and card holder.

Similarity score S_{GF}
We use our knowledge about the minimum, average and maximum value for each of the signature sets to create a similarity score. For a test signature s_t, we compute the seven global features. For each feature of the test signature that lies within the card holder's maximum and minimum values, we assign a 1 in the score vector G. If the test value does not lie within this window, we assign a 0.

$$G = [g_l, g_t, g_{v_{\max}}, g_{\bar{v}}, g_{a_{\max}}, g_n, g_r], g \in \{0, 1\}$$

We assume that not all of the global features are of the same significance. We therefore define a weighting vector

$$W = [w_l, w_t, w_{v_{\max}}, w_{\bar{v}}, w_{a_{\max}}, w_n, w_r]$$

Our similarity score is calculated as the cross product of the score vector G and the weighting vector W:

$$S_{GF} = G \times W$$

As every signature added to the database possibly enlarges the window between a card holder's minimum and maximum value for each feature, a sliding window should be used for larger datasets. As we were are working with less than

P	l	t	$v_{\max}$	$\bar{v}$	$a_{\max}$	n	r
0	1316.81	1640.0	2.27	0.8	1.11	2.0	0.0
1	827.09	1373.0	1.5	0.6	0.58	2.0	1.0
2	839.37	1445.0	1.71	0.58	0.7	2.0	2.0
3	1434.08	2250.0	2.22	0.64	1.49	3.0	2.0
4	669.82	1122.0	1.79	0.6	0.49	1.0	2.0
5	924.78	1856.0	1.14	0.5	0.64	2.0	2.0
6	1281.41	1399.0	2.49	0.92	1.53	2.0	1.0
7	843.17	1424.0	1.13	0.59	0.48	2.0	1.0
8	639.14	1209.0	1.7	0.53	0.73	1.0	1.0
9	1299.5	5467.0	0.79	0.24	0.41	4.0	1.0

(a) **The minimum values for each signature set and all global features**

P	l	t	$v_{\max}$	$\bar{v}$	$a_{\max}$	n	r
0	1701.69	1942.75	15.5	0.88	16.93	2.13	0.5
1	1235.27	1773.95	11.24	0.7	13.36	2.0	1.93
2	1206.72	1762.06	10.22	0.68	18.96	3.11	2.33
3	2565.76	3217.96	17.62	0.8	27.72	5.58	2.42
4	1104.21	1588.11	8.1	0.7	9.58	3.43	2.81
5	1308.19	3047.47	5.23	0.43	8.25	8.0	2.3
6	2480.3	2102.87	13.71	1.18	16.89	3.87	2.04
7	1260.28	2194.41	6.79	0.57	9.41	2.64	2.07
8	1311.1	1574.98	11.18	0.83	15.12	2.6	1.47
9	2047.7	6953.81	3.95	0.29	5.04	12.19	2.4

(b) **The average values for each signature set and all global features**

P	l	t	$v_{\max}$	$\bar{v}$	$a_{\max}$	n	r
0	2571.75	2282.0	54.42	1.13	73.35	3.0	1.0
1	2048.9	2957.0	82.49	0.69	139.0	2.0	3.0
2	2069.97	2117.0	52.35	0.98	105.95	5.0	3.0
3	4673.42	9370.0	104.48	0.5	180.54	8.0	4.0
4	2041.9	2914.0	63.79	0.7	99.93	5.0	4.0
5	2425.82	4694.0	41.11	0.52	48.09	11.0	4.0
6	4388.68	3633.0	75.89	1.21	134.83	5.0	3.0
7	1958.88	4349.0	65.51	0.45	92.97	8.0	3.0
8	2146.82	2749.0	65.46	0.78	112.3	4.0	2.0
9	3428.25	8083.0	20.12	0.42	33.69	17.0	9.0

(c) **The maximum values for each signature set and all global features**

Figure 4.2: The minimum, average and maximum values we obtained for all global features for each subject's set of signatures.

Listing 4.1: The fitness function for global features tries to reduce the number of signatures with the hightes possible score for a given weighting vector W

```
until fitness == #signatures_in_groundtruth

    fitness = calculate_globabl_features w

    if fitness < previous_fitness
        save w
    end

    permutate w
end
```

50 signatures per card holder, we did not implement a sliding window mechanism as part of our research.

Fitness function $\mathbb{F}_{GF}$

In order to find the best performing weighting vector W, we run each signature against each of the ten signature models and minimize the number of false positive matches. A signature model is defined by the minimum/maximum range for each feature.

By definition, all signatures will score a perfect score at least once. The fitness function $\mathbb{F}_{GF}$ tries to get as close to that number as possible. Starting with the weighting vector $W = [0, 0, 0, 0, 0, 0, 0]$, the first result corresponds to the total number of signatures used for this experiment. By exploring all possible W for $w \in 0, 1, 2$, the algorithm finds the W for which the least elements the highest similarity score S_{GF} at the same time.

The fitness function for global features $\mathbb{F}_{GF}$ is defined in pseudo code in Listing 4.1.

Classification of Signatures

We defined the start vecor $W = [0, 0, 0, 0, 0, 0, 0]$ and let the fitness function permutate the vector until a local minimum was found. The local minimum was reached for

$$W = [w_l, w_t, w_{v_{\max}}, w_{\bar{v}}, w_{a_{\max}}, w_n, w_r] = [0, 1, 0, 1, 0, 1, 1]$$

We used a subset of 278 signatures for this experiment. The local minimum was reached at 755 signatures that scored the a similarity score $S_{GF} = 4$ for the same W. This includes 478 signatures more than the global minimum. From W we conclude that time t, average velocity $\bar{v}$, the number of segments n and the width-to-height ratio r are the important global features and should be used to compare signatures and detect forgeries.

When we extend the fitness function to include the two groups with the highest occurrences, we get 840 signatures in the top tier and the weighting vector W' which confirms the importance of the same features.

$$W' = [w_l, w_t, w_{v_{\max}}, w_{\bar{v}}, w_{a_{\max}}, w_n, w_r] = [0, 2, 0, 2, 0, 2, 1]$$

Detection of forgeries

We used the previously found weighting vector W to test the robustness of this approach against forgeries. We set

$$W = [0, 1, 0, 1, 0, 1, 1]$$

and run all forgeries against the previously trained signature models, which are defined by the minimum and maximum values.

For each forgery, we compute the value of each of the four important global features, $t, \bar{v}, n$ and r. We compare the generated value to the the bounds for that feature for each person. A match requires that the test value lies within the reference minimum and maximum for all four features. The results are shown in Figure 4.3. The tables first lists the minimum and maximum for each feature and card holder, the value found for the forgery and whether the signature was correctly rejected as a forgery or accepted (a false positive match).

All but one forgeries were correctly rejected as forgeries. We notice that most of the forgeries were rejected mostly because the impostor took too long to sign (t too high) or because he did not sign fast enough ($\bar{v}$ too little). We achieve an error rate of 3.33 percent with this system.

Real time feasibility

Computing the features for a signature and comparing them to the previously computed minimum and maximum values takes less than one hundredth of a second. By using a sliding window, the computational complexity of computing the minimum and maximum values is guaranteed to have an upper bound even if the total amount of signatures for a card holder may grow to very large. We conclude that this approach is feasible for real time application.

P	t_r	$\bar{v}_r$	n_r	r_r	t_f	$\bar{v}_f$	n_f	r_f	Hit
0	[1640.0, 2282.0]	[0.8, 1.13]	[2.0, 3.0]	[0.0,1.0]	13782.0	0.14	2.0	2.0	✗
1	[1373.0, 2957.0]	[0.6, 0.69]	[2.0, 2.0]	[1.0,3.0]	5229.0	0.29	1.0	2.0	✗
2	[1445.0, 2117.0]	[0.58, 0.98]	[2.0, 5.0]	[2.0,3.0]	4141.0	0.29	5.0	2.0	✗
3	[2250.0, 9370.0]	[0.5, 0.64]	[3.0, 8.0]	[2.0,4.0]	3252.0	0.34	2.0	1.0	✗
4	[1122.0, 2914.0]	[0.6, 0.7]	[1.0, 5.0]	[2.0,4.0]	8743.0	0.15	3.0	2.0	✗
5	[1856.0, 4694.0]	[0.5, 0.52]	[2.0, 11.0]	[2.0,4.0]	3658.0	0.39	4.0	2.0	✗
6	[1399.0, 3633.0]	[0.92, 1.21]	[2.0, 5.0]	[1.0,3.0]	2815.0	0.53	2.0	2.0	✗
7	[1424.0, 4349.0]	[0.59, 0.45]	[2.0, 8.0]	[1.0,3.0]	3978.0	0.48	2.0	2.0	✓
8	[1209.0, 2749.0]	[0.53, 0.78]	[1.0, 4.0]	[1.0,2.0]	10863.0	0.19	4.0	2.0	✗
9	[5467.0, 8083.0]	[0.24, 0.42]	[4.0, 17.0]	[1.0,9.0]	24910.0	0.08	18.0	1.0	✗

(a) **The results for spontaneous forgeries $\mathcal{F}_1$**

P	t_r	$\bar{v}_r$	n_r	r_r	t_f	$\bar{v}_f$	n_f	r_f	Hit
0	[1640.0, 2282.0]	[0.8, 1.13]	[2.0, 3.0]	[0.0,1.0]	4477.0	0.38	3.0	1.0	✗
1	[1373.0, 2957.0]	[0.6, 0.69]	[2.0, 2.0]	[1.0,3.0]	7835.0	0.22	2.0	1.0	✗
2	[1445.0, 2117.0]	[0.58, 0.98]	[2.0, 5.0]	[2.0,3.0]	10237.0	0.16	5.0	1.0	✗
3	[2250.0, 9370.0]	[0.5, 0.64]	[3.0, 8.0]	[2.0,4.0]	11261.0	0.17	3.0	1.0	✗
4	[1122.0, 2914.0]	[0.6, 0.7]	[1.0, 5.0]	[2.0,4.0]	5508.0	0.2	3.0	3.0	✗
5	[1856.0, 4694.0]	[0.5, 0.52]	[2.0, 11.0]	[2.0,4.0]	2465.0	0.54	3.0	2.0	✗
6	[1399.0, 3633.0]	[0.92, 1.21]	[2.0, 5.0]	[1.0,3.0]	7338.0	0.22	2.0	1.0	✗
7	[1424.0, 4349.0]	[0.59, 0.45]	[2.0, 8.0]	[1.0,3.0]	9386.0	0.19	2.0	2.0	✗
8	[1209.0, 2749.0]	[0.53, 0.78]	[1.0, 4.0]	[1.0,2.0]	15633.0	0.1	8.0	2.0	✗
9	[5467.0, 8083.0]	[0.24, 0.42]	[4.0, 17.0]	[1.0,9.0]	11826.0	0.12	16.0	2.0	✗

(b) **The results for sample forgeries $\mathcal{F}_2$**

P	t_r	$\bar{v}_r$	n_r	r_r	t_f	$\bar{v}_f$	n_f	r_f	Hit
0	[1640.0, 2282.0]	[0.8, 1.13]	[2.0, 3.0]	[0.0,1.0]	10799.0	0.16	2.0	2.0	✗
1	[1373.0, 2957.0]	[0.6, 0.69]	[2.0, 2.0]	[1.0,3.0]	3896.0	0.55	2.0	2.0	✗
2	[1445.0, 2117.0]	[0.58, 0.98]	[2.0, 5.0]	[2.0,3.0]	9016.0	0.27	4.0	2.0	✗
3	[2250.0, 9370.0]	[0.5, 0.64]	[3.0, 8.0]	[2.0,4.0]	6731.0	0.24	2.0	2.0	✗
4	[1122.0, 2914.0]	[0.6, 0.7]	[1.0, 5.0]	[2.0,4.0]	5506.0	0.29	2.0	2.0	✗
5	[1856.0, 4694.0]	[0.5, 0.52]	[2.0, 11.0]	[2.0,4.0]	3442.0	0.41	3.0	2.0	✗
6	[1399.0, 3633.0]	[0.92, 1.21]	[2.0, 5.0]	[1.0,3.0]	9328.0	0.17	2.0	1.0	✗
7	[1424.0, 4349.0]	[0.59, 0.45]	[2.0, 8.0]	[1.0,3.0]	6295.0	0.26	2.0	2.0	✗
8	[1209.0, 2749.0]	[0.53, 0.78]	[1.0, 4.0]	[1.0,2.0]	7133.0	0.24	6.0	2.0	✗
9	[5467.0, 8083.0]	[0.24, 0.42]	[4.0, 17.0]	[1.0,9.0]	9138.0	0.17	17.0	2.0	✗

(c) **The results for knowledgable forgeries $\mathcal{F}_3$**

Figure 4.3: The minimum, average and maximum values we obtained for all global features for each subject's set of signatures.

4.2.4 Dynamic Time Warping

Similarity score S_{DTW}
The similarity score in this case is directly given as the DTW algorithm's output. We vary the input to the DTW algorithm to see the effect on the similarity score. We test combinations of the following parameters in our experiments:

- **Age**: In order to reduce the computation time, we run the algorithm on a subset of all signatures for a card holder. We compute the similarity score for *recent*, i.e. the last 5, 10 or 20 signatures, or *old* signatures, i.e. the first 5, 10, 20 signatures ever recorded by a card holder.

- **Amount of signatures**: As the computation time depends on the number of signatures used for each card holder, we obtain results for 5, 10 and 20 signatures per card holder. We also conduct and experiment with the full dataset, i.e. up to 50 signatures per card holder, to measure the impact on accuracy.

- **Length**: By analyzing the visual representation of the signatures in our dataset, we assume that the variability of a signature is higher towards the end than at the beginning. To prove our visual impression, we perform experiments using the *full* signature vector, i.e. including all points of a signature, or only a *partial* amount of all points.

- **Vector**: The DTW algorithm uses a distance function to compute the distance between two points. For each point in our signature database, we obtained the geometric position x, y and timestamp t. We conduct experiments on both, the full vector xyt and only the geometric information xy. We use the 2-Norm to compute the distance between two vectors.

Fitness function $\mathbb{F}_{DTW}$

The fitness function $\mathbb{F}_{DTW}$ is defined as the Equal Error Rate (EER) during the classification phase. It is used to determine the optimal threshold to accept/reject a signature.

We are also interested in the computation time to compute the EER to know whether or not a combination is feasible for real time evaluation.

Classification of Signatures
The results of our experiments are listed in Table 4.5. We found that:

- The algorithm results in a lower EER for distance vectors that exclude time information.

- A higher of signatures per card holder and the addition of time information in vector, both negatively affect the computation time.

Experiment	Age	# Sig	Length	Vector	Time [s]	Threshold	EER
1	recent	5	full	xyt	249	0.02	0.29
2	recent	5	full	xy	222	0.07	0.32
3 4.4c	recent	5	partial	xyt	45	0.23	0.19
4 4.4a	recent	5	partial	xy	40	0.30	0.12
5	recent	10	full	xyt	782	0.02	0.33
6	recent	10	full	xy	762	0.06	0.29
7	recent	10	partial	xyt	189	0.22	0.29
8 4.4b	recent	10	partial	xy	171	0.30	0.14
9	recent	20	partial	xyt	728	0.16	0.37
10	recent	20	partial	xy	661	0.28	0.20
11	old	10	full	xyt	944	0.01	0.39
12	old	10	full	xy	866	0.05	0.30
13 4.4d	recent	50	partial	xy	2137	0.26	0.33

Table 4.5: Results of DTW experiments

- More recent signatures perform better than old signatures. This motivates the use of a sliding window.

- Partial information of the signature vector has a positive effect on the computation time and leads to a lower EER. This confirms the higher variability in the signatures towards the end.

- Signature classification is very time consuming as the cross product between all signatures in the same set needs to be computed.

Experiment 4 leads to the lowest EER. It uses the following input combination:

- Recent signatures

- Five signatures per card holder

- Partial information of the signature vector

- Time information not included in distance vector

Experiment 4 produces an EER of 0.12 with a threshold of 0.3. The threshold is relative to the highest score encountered during the experiment. With a computation time of 40 seconds to compute the similarity score S_{DTW} for the cross product of all signatures, it is also the fastest combination. The graphic

representation of the results is shown in Figure 4.4a. We use this combination to detect forgeries.

Detection of forgeries

In order to detect forgeries we conduct the following steps for the above found combination of inputs.

1. For each forgery we calculate the similarity score S_{DTW} to the five most recent signatures of the card holder. The results are shown in Figure 4.5a.

2. We apply the previously found relative threshold of 0.3 to decide if the forgery is classified as genuine or forged. During the classification experiment we found the highest score of $241,881$ and thus we get an absolute threshold of $241,881 \cdot 0.3 = 72,564.2$. Figure 4.5b shows the results after applying the threshold.

3. As we create five similarity scores between the forgery and the five most recent signatures of a card holder, we can then apply majority voting to decide whether the test signature is classified as genuine or forged. Table 4.5c shows the results of the majority voting.

From the 30 forgeries in our database, 11 were wrongly classified as genuine. This corresponds to an error rate of 36 percent.

Real time feasibility

Calculating the DTW score between two signatures under the described circumstances takes about one tenth of a second which is fast enough to fulfill our real time requirements. However, computing the threshold requires longer computation and should be implemented in a background process.
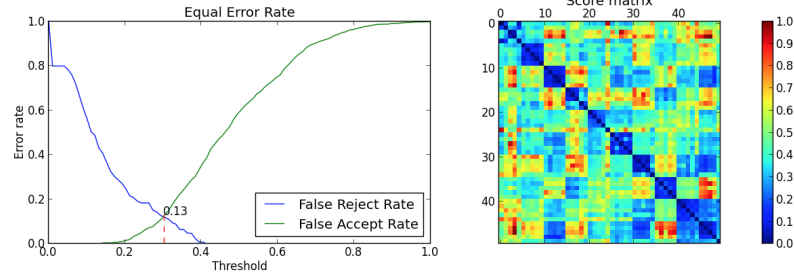
(a) **DTW with five recent signatures per card holder, partial information about each signature and excluding time information**
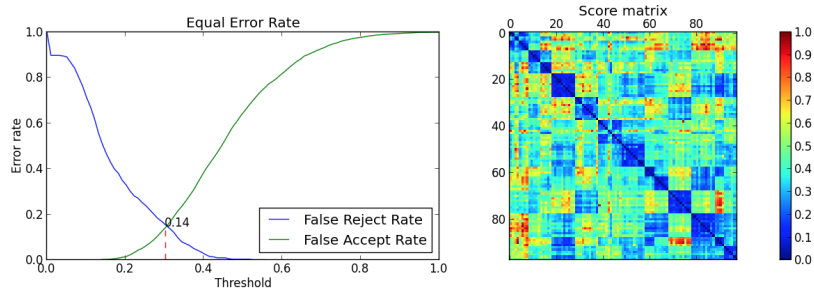
(b) **DTW with ten recent signatures per card holder, partial information about each signature and excluding time information**
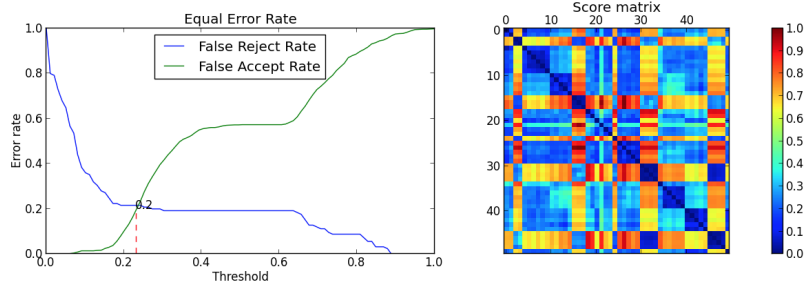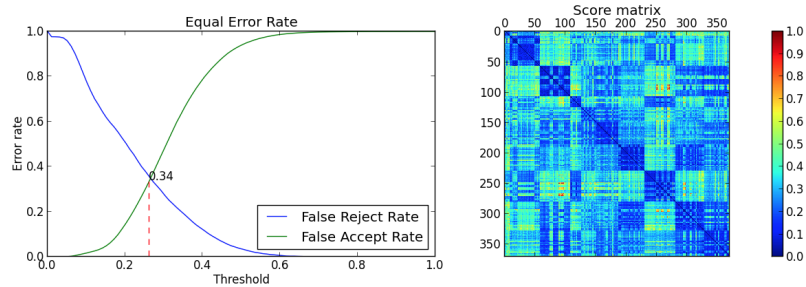

(c) **DTW with five recent signatures per card holder, partial information about each signature and including time information**

(d) **DTW with up to 50 recent signatures per card holder, partial information about each signature and excluding time information**

Figure 4.4: EER and Heatmap of the three best performing and the longtest DTW experiments

(a) (b)

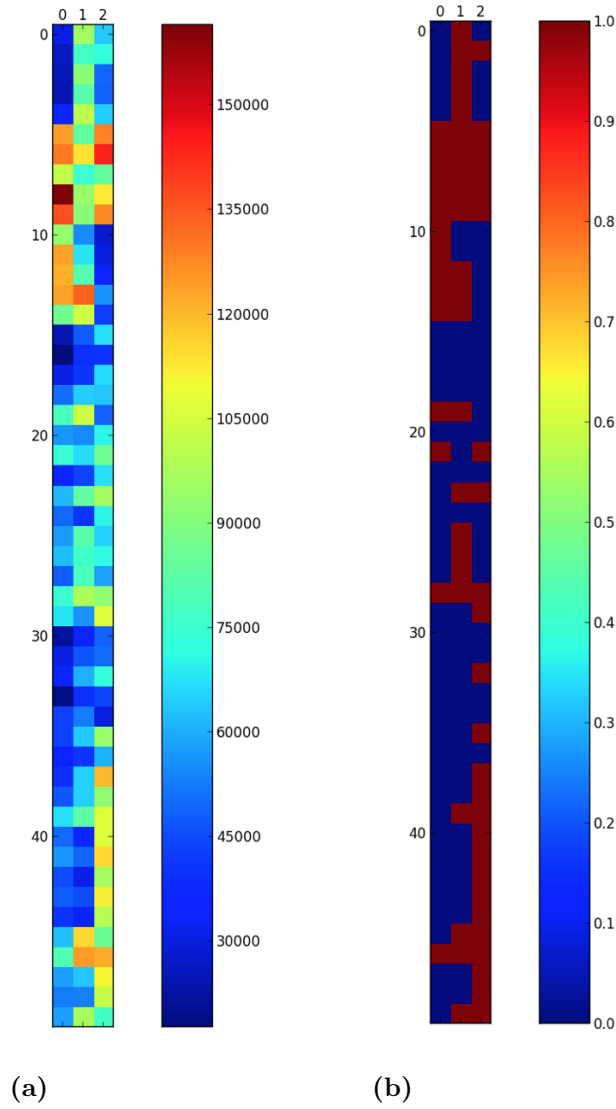| P	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
1	✗	✓	✗
2	✓	✓	✓
3	✓	✓	✗
4	✗	✗	✗
5	✗	✗	✗
6	✗	✓	✗
7	✗	✗	✗
8	✗	✗	✓
9	✗	✗	✓
10	✗	✓	✓

(c)

Figure 4.5: The heatmap (a), the discretized heatmap after applying the threshold (b) and the false positives after applying majority voting (c)

4.2.5 Hidden Markov Models

Several implementations of the HMM algorithm are available. We chose to work with the implementation included in the MATLAB[1] tool suite because it provided the most extensive documentation. However, when an algorithm based on HMM is at one point be deployed to SumUp's production environment, we suggest to create a custom implementation to optimize speed and avoid high license fees.

Similarity score S_{HMM}
After training the HMM with each card holder's signature set, we calculate the likelihood for each signature to belong to each of the trained HMMs. We define S_{HMM} as the likelihood of a signature being generated by a specific model.

We try different combinations of hidden states and amount of signatures per card holder to optimize the classification of signatures.

Fitness function $\mathbb{F}_{HMM}$
We define the fitness function $\mathbb{F}_{HMM}$ as the amount of false positive matches we get during the signature classification. We proceed as follows to compute the number of false positive matches for each experiment:

1. We train each card holder's signature set with his signatures.

2. By computing the likelihood for each signature of a card holder's set with this card holder's trained model, we obtain a maximum and minimum likelihood for each card holder's trained model.

3. We compute the likelihood for each signature in the database with each trained model. If the likelihood value is within the previously obtained maximum and minimum value, but the signature was not created by this model's card holder, we count it as a false positive.

Classification of Signatures
Table 4.6 shows the results of our experiments. Although the number of false positives decreases with more states, the computation time increases and we did not conduct any experiments for more than 9 states because the computation time would make an implementation unfeasible.

Experiments with more signatures cause larger windows and thereby a higher number of false positives per 100 Signatures. We use the configuration used in Experiment 5 to detect forgeries as it is a reasonable trade-off between computation time and accuracy. We use the following setup:

- 7 hidden states

---

[1]http://mathworks.com/products/matlab/

Exp	States	# Sig	Time [s]	Avg window size	False Positives / 100 Sig
1	5	10	48	142.0	16.63
2	5	50	214	209.0	33.8
3	6	10	61	144.0	12.45
4	6	50	560	593.0	44.18
5	7	10	105	144.0	9.8
6	7	50	740	370.0	34.72
7	8	10	150	145.0	7.14
8	9	10	225	148.0	5.1

Table 4.6: Results of HMM experiments

Person	Window	Width	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
1	[-144, -121]	23	-119 ✗	0 ✗	-435 ✗
2	[-97, -73]	24	0 ✗	0 ✗	0 ✗
3	[-331, -100]	231	0 ✗	-506 ✗	-231 ✓
4	[-223, -75]	148	0 ✗	0 ✗	-235 ✗
5	[-142, -110]	32	-396 ✗	-657 ✗	0 ✗
6	[-526, -190]	336	-482 ✓	0 ✗	-269 ✓
7	[-448, -192]	256	-625 ✗	-476 ✗	-344 ✓
8	[-235, -102]	133	0 ✗	0 ✗	0 ✗
9	[-106, -78]	28	-403 ✗	-478 ✗	0 ✗
10	[-330, -93]	237	-278 ✓	-599 ✗	-244 ✓

Table 4.7: Window Sizes for Experiment Configuration 5

- The model is trained with the 10 most recent signatures

- The symbol alphabet consists of 8 angles

Detection of forgeries

For the mentioned configuration, we obtain the windows between minimum and maximum likelihood shown in column two of Table 4.7. We run each card holder's forgeries against this card holder's trained HMM. If the likelihood is within the shown window, we consider it a false positive match.

Table 4.7 shows the results we obtain. Out of 30 forgeries, 6 were false positive matches. We conclude an error rate of 20 percent of our chosen HMM algorithm.

Real time feasibility

The computation time needed to train the HMM models are shown in Table 4.6. Given the parameter combination of experiment 5, it takes takes 10.5 seconds to train the model for each card holder with the last 10 signatures. As this task can be done in the background and in parallel, we consider it to be a small enough time to scale the system for thousands of users.

The computation of the likelihood in respect to a given model takes less than a tenth of a second and thus fulfills real time requirements.

Person	Global Features	DTW	HMM
1		$\mathcal{F}_2$	
2		$\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$	
3		$\mathcal{F}_1, \mathcal{F}_2$	$\mathcal{F}_3$
4			
5			
6		$\mathcal{F}_2$	$\mathcal{F}_1, \mathcal{F}_3$
7	$\mathcal{F}_1$		$\mathcal{F}_3$
8		$\mathcal{F}_3$	
9		$\mathcal{F}_3$	
10		$\mathcal{F}_2, \mathcal{F}_3$	$\mathcal{F}_1, \mathcal{F}_3$

Table 4.8: The false positives we encountered with the global features, DTW and HMM algorithm

4.2.6   Fusion of scores

Table 4.8 shows a summary of all algorithms and the false positive matches we encountered. Only for one out of ten test subjects (Person 10), we encountered the case that the same forgery was a false positive match in more than one system ($\mathcal{F}_3$). For all other forgeries we are able to apply majority voting to exclude the false positives. We propose the following system:

1. As the algorithm based on global features has the highest accuracy and to save computation power, we suggest to first run this algorithm and stop if the signature is identified as a forgery

2. If the signature was not identified as a forgery, perform the DTW and HMM algorithm.

3. Apply majority voting based on the three results. If a forgery is detected by at least two algorithms, flag it as a forger. Otherwise consider the signature to be genuine.

Conclusions

We designed and implemented two complementary fraud protection systems. Our fraud prevention system was already released to production and performs very well. The number of false positive matches on high risk clients dropped significantly after the new component was deployed. This helped SumUp's operations team to reduce the unnecessary work hours that were spent to manually clear false positive matches. SumUp's exponential growth caused an exponential growth of false positive matches with the old check and caused delays in the sign-up process. Since the new component was released, the backlog never had to be carried to the next week and thereby helps SumUp's to grow even faster.

The proposed fraud detection system is able to detect forgeries with a high accuracy while retaining the real time property. Signatures written by finger on a mobile device show less stable characteristics than signatures written with a pen but the combination of three algorithms allows us to minimize the number of false positive matches. For the remaining false positives, we apply a feedback system in addition to our signature verification system to ensure that clients are at all time able to process card payments with SumUp, even if their signature is not recognized by the automatic system.

Bibliography

[1] Woolsey, B., Schulz, M.: Credit card statistics, industry facts, debt statistics. Google Search Engine (2010)

[2] Council, E.P.: White paper mobile payments. EPC492-09 (October 2012)

[3] Wang, S.: A comprehensive survey of data mining-based accounting-fraud detection research. In: Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on. Volume 1. (2010) 50–53

[4] Authority, F.S.: The money laundring regulations 2007 (2013)

[5] Hanmandlu, M., Yusof, M., Madasu, V.K.: Off-line signature verification and forgery detection using fuzzy modeling. In: Pattern Recognition. (March 2005)

[6] Shafiei, M., Rabiee, H.: A new online signature verification algorithm using variable length segmentation and hidden markov models. In: Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on. (2003) 443–446 vol.1

[7] Fierrez-Aguilar, J., Nanni, L., Lopez-Peñalba, J., Ortega-Garcia, J., Maltoni, D.: An on-line signature verification system based on fusion of local and global information. In: Audio-and video-based biometric person authentication, Springer (2005) 523–532

[8] Ji, H.W., Quan, Z.H.: Signature verification using wavelet transform and support vector machine. In: Advances in Intelligent Computing. Springer (2005) 671–678

[9] Kaewkongka, T., Chamnongthai, K., Thipakorn, B.: Off-line signature recognition using parameterized hough transform. In: Signal Processing and Its Applications, 1999. ISSPA'99. Proceedings of the Fifth International Symposium on. Volume 1., IEEE (1999) 451–454

[10] Fang, B., Leung, C., Tang, Y., Tse, K., Kwok, P., Wong, Y.: Off-line signature verification by the tracking of feature and stroke positions. Pattern recognition **36**(1) (2003) 91–101

[11] Fang, B., Wang, Y., Leung, C., Tse, K., Tang, Y.Y., Kwok, P.C.K., Wong, Y.: Offline signature verification by the analysis of cursive strokes. International Journal of Pattern Recognition and Artificial Intelligence **15**(04) (2001) 659–673

[12] Piyush Shanker, A., Rajagopalan, A.N.: Off-line signature verification using dtw. Pattern Recogn. Lett. **28**(12) (September 2007) 1407–1414

[13] Herbst, B., Coetzer, H.: On an offline signature verification system. In: Proceedings of the 9th annual South African Workshop on Pattern Recognition. (1998) 600–604

[14] Yoshimural, M., Yoshimura, I. In: An application of the sequential dynamic programming matching method to off-line signature verification. Volume 1339 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1997) 299–310

[15] Justino, E.J.R., Yacoubi, A.E., Bortolozzi, F., Sabourin, R.: An off-line signature verification system using hmm and graphometric features (2000)

[16] Drouhard, J., Sabourin, R., Godbout, M.: A neural network approach to off-line signature verification using directional PDF. Pattern Recognition **29**(3) (March 1996) 415–424

[17] Lau, K.K., Yuen, P., Tang, Y.Y.: Stroke extraction and stroke sequence estimation on signatures. In: Pattern Recognition, 2002. Proceedings. 16th International Conference on. Volume 3. (2002) 119–122 vol.3

[18] Wilkinson, T.S., Pender, D.A., Goodman, J.W.: Use of synthetic discriminant functions for handwritten-signature verification. Appl. Opt. **30**(23) (Aug 1991) 3345–3353

[19] Sabourin, R., Genest, G., Preteux, F.: Off-line signature verification by local granulometric size distributions. Pattern Analysis and Machine Intelligence, IEEE Transactions on **19**(9) (1997) 976–988

[20] Bajaj, R., Chaudhury, S.: Signature verification using multiple neural classifiers. Pattern Recognition **30**(1) (1997) 1 – 7

[21] Ramesh, V., Murty, M.N.: Off-line signature verification using genetically optimized weighted features. Pattern Recognition **32**(2) (1999) 217 – 233

[22] Qi, Y., Hunt, B.R.: Signature verification using global and grid features. Pattern Recognition **27**(12) (1994) 1621 – 1629

[23] de Bruyne, P., Forre, R.: Signature verification with elastic image matching. In: Proceedings - 1986 International Carnahan Conference on Security Technology: Electronic Crime Countermeasures., Chalmers Univ of Technology, Gothenburg, Swed; Univ of Kentucky, Coll of Engineering, Office of Engineering Continuing Education, Lexington, KY, USA; IEEE, New York, NY, USA; GTE Service Corp, Evaluation & Support Dep (August 1986) 113–118

[24] Plamondon, R., Lorette, G.: Automatic signature verification and writer identification - the state of the art. Pattern Recognition **22**(2) (1989) 107 – 131

[25] Camino, J.L., Travieso, C.M., Morales, C.R., Ferrer, M.A.: Signature classification by hidden markov model. (1999) 481–484

[26] Kashi, R., Hu, J., Nelson, W., Turin, W.: A hidden markov model approach to online handwritten signature verification. International Journal on Document Analysis and Recognition **1**(2) (1998) 102–109

[27] Martens, R., Claesen, L.: On-line signature verification by dynamic time-warping. In: Pattern Recognition, 1996., Proceedings of the 13th International Conference on. Volume 3., IEEE (1996) 38–42

[28] Keogh, E.J., Pazzani, M.J.: Scaling up dynamic time warping for datamining applications. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '00, New York, NY, USA, ACM (2000) 285–289

[29] Fuentes, M., Garcia-Salicetti, S., Dorizzi, B.: On line signature verification: Fusion of a hidden markov model and a neural network via a support vector machine. In: Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on. (2002) 253–258

[30] Keogh, E.: Exact indexing of dynamic time warping. In: Proceedings of the 28th international conference on Very Large Data Bases. VLDB '02, VLDB Endowment (2002) 406–417

[31] Yasuhara, M., Oka, M.: Signature verification experiment based on nonlinear time alignment: a feasibility study. IEEE Trans. on Systems, Man and Cybernetics, part C **12**(3) (1977) 212–216

[32] Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. Acoustics, Speech and Signal Processing, IEEE Transactions on **26**(1) (1978) 43–49

[33] Sato, Y., Kogure, K.: Online signature verification based on shape motion and writing pressure. Proceedings of the 6th ICPR (1982) 823–826

[34] Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE **77**(2) (1989) 257–286

Appendix Chapter

Listing A.1: The Algorithm that parses the csv file creates a separate DB record for each first/last name pair

```
for record in csv-file

    # Create Arrays to store all first & last names
    # and initialize with the most common first/last name
    first_names = []
    first_names << record["first_name"]

    last_names = []
    last_names << record["last_name"]

    # Get all other pairs from alternative_spellings ...
    for pair in record["alternative_spellings"]
        first_names << pair[0]
        last_names << pair[1]
    end

    # ... and from aliases
    for pair in record["aliases"]
        first_names << pair[0]
        last_names << pair[1]
    end

    # Now create the DB records
    for fname in first_names
        for lname in last_names
            if record["category"] == "PEP"
                PEP.create_in_db fname, lname, record
            else
                HRA.create_in_db fname, lname, record
            end
        end
    end
end
```