Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

User Management of a Privacy-Based Carpooling Platform

Marvin Fischer

Course of Study:	B. Sc., Softwaretechnik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Robin Pesl, M.Sc.
Commenced:	April 3, 2023
Completed:	October 3, 2023

Abstract

The principle of carpooling is on a path of increasing popularity. In combination with self-driving cars, we are going to encounter new ways to travel collectively. This, however, brings us to new problems. Self Driving cars would depend on a supervised and coordinated system, which would allow it to track all journeys of the passenger and therefore create an extended knowledge graph about a real person. The solution to this is a user management system, which would mask the actual identity of a user and make them untrackable. The goal of this work is to give an overview of the latest technology, which can be used to identify and authenticate the user without revealing their identity neither to the owners of the self-driving cars nor to the carpooling platform itself. Moreover, it provides a sample implementation, running on mobile and Blockchain technology, and identifies potential weak points regarding the currently available technologies. In addition to the identification and authentication mechanism, we provide a reputation system, which enables all parties to rate each other, without revealing their identity of themself or knowing the identity of the party, that is rated. Likewise, we also indicate possible ways, in which the authentication system could be extended to run in a real-world environment.

Kurzfassung

Das Prinzip der Fahrgemeinschaften erfreut sich immer größerer Beliebtheit. In Kombination mit selbstfahrenden Autos werden wir neue Wege des gemeinsamen Fortbewegung ermöglicht. Dies bringt uns jedoch zu neuen Problemen. Selbstfahrende Autos wären auf ein überwachtes und koordiniertes System angewiesen, das es ermöglichen würde, alle Fahrten des Passagiers zu verfolgen und so einen erweiterten Wissensgraphen über eine reale Person zu erstellen. Die Lösung hierfür ist ein Benutzerverwaltungssystem, das die tatsächliche Identität eines Benutzers verschleiert und ihn unauffindbar macht. Das Ziel dieser Arbeit ist es, einen Überblick über die neueste Technologie zu geben, die zur Identifizierung und Authentifizierung des Benutzers verwendet werden kann, ohne dass seine Identität weder den Besitzern der selbstfahrenden Autos noch der Fahrgemeinschaftensplattform selbst preisgegeben wird. Darüber hinaus stellt es eine Beispielimplementierung bereit, die auf Mobil- und Blockchain-Technologie basiert, und identifiziert potenzielle Schwachstellen in Bezug auf die derzeit verfügbaren Technologien. Zusätzlich zum Identifizierungs- und Authentifizierungsmechanismus stellen wir ein Reputationssystem bereit, welches es allen Parteien ermöglicht, sich gegenseitig zu bewerten, ohne ihre Identität preiszugeben oder die Identität der bewerteten Partei zu kennen. Ebenso werden wir Möglichkeiten aufzeigen, wie das Authentifizierungssystem auf den Einsatz in einer realen Umgebung erweitert werden könnte.

Contents

List of Figures

List of Tables

List of Code Samples

Acronyms

AES Advanced Encryption Standard. 22

API Application Programming Interface. 26

BSI Bundesamt für Sicherheit in der Informationstechnik. 33

CA Certificate Authority. 25

ECDSA Elliptic Curve Digital Signature Algorithm. 24

eID Electronic Identification. 31

eIDAS electronic IDentification, Authentication and trust Services. 33

FHE Fully Homomorphic Encryption. 24

GCM Galois/Counter Mode. 23

GDPR General Data Protection Regulation. 32

HTTP Hypertext Transfer Protocol. 26

HTTPS Hypertext Transfer Protocol Secure. 26

JSON JavaScript Object Notation. 26

JWTs Json Web Tokens. 35

NFC Near Field Communication. 27

NIST National Institute of Standards and Technology. 69

PCA Pseudonym Certification Authority. 39

PD Personal Data. 32

PII Personal Identifiable Information. 32

PKI Public Key Infrastruktur. 25

REST Representational state transfer. 26

RSA Rivest–Shamir–Adleman. 22

SHA Secure Hash Algorithm. 24

SSH Secure Shell. 35

SSL Secure Sockets Layer. 25

TLS Transport Layer Security. 25

1 Introduction

Automation and artificial intelligence are well discussed nowadays and cars handle a lot of the driving on their own and will become fully autonomous over the next couple of years [GAGK21]. This opens up new possibilities when it comes to public transportation. Such as using carpooling instead of owning a car or taking the bus or train. But is it possible to build a carpooling platform, which provides the user with full anonymity, so that they can primarily use such a system without revealing any information about themself? This work shows that at the current stage, we are not capable of providing full anonymity, however, further progress in technologies such as Self Sovereign Identities, might change this over the next years.

1.1 Context

The level of autonomous driving is reaching a point where the driver needs to do less work and is a passenger in their own car. So, why should a person buy and own a car, if they could simply rent a car and only pay for the journey they actually take? And if the person is just a passenger in that car, why not share the ride with other people, who need to take a similar journey? This would reduce the costs of all people taking that particular journey.

Car sharing is in increasing demand. With the rise of new wireless technologies and the interaction with our surrounding by mobile apps becoming more mainstream, people are more willing to share vehicles with others. Those range from sharing a car for reaching a destination or sharing vans for transporting things from one location to another [SC19].

This is supported by the rising popularity of companies such as BlaBlaCar or the taxi competition Uber. Those take a large advantage of the use of mobile applications and the way people interact with others by using those apps. Additionally, both applications target different groups [QPM21].

Moreover, for the passengers, additional stress such as searching for a parking slot in bigger and crowded cities is dismissing and instead, only the journey itself impacts the people. This makes it easier to plan in advance and use time more efficiently. Further, cities would need to provide fewer parking slots in general [SCB+18].

Such a carpooling platform would give a good alternative to public transportation, which is strictly timed, not as reliant and in some areas not even present. A carpooling platform with autonomous cars would enable people in rural areas, who do not have a driver's license or are too old to drive safely, i.e. for medical reasons, to reach every location they want to. Moreover, older people could socialise more easily and even receive help for their daily activities, while at the same time being independent from inflexible timetables [ADA23; HDPB19; HNS+20; Inf23b; SCB+18; SPP22].

But when people are using a carpooling platform, their privacy of movement is quite restricted, compared to a private car or public transportation where people just can hop on and take their journey. This makes an anonymous identity pretty important so that people can take their everyday journies, while nobody can track back the movement to a single person and gain personal information such as their private address or workplace address. Further, such a platform should provide the same privacy as using a private car. For increasing the truth in the user, a user management system with rateable users is essential. Anonymous reputation systems have been widely discussed in recent days and are analysed and compared with each other. Similar to Self Sovereign Identities, which enable users to manage their digital identity for themself without the need to verifier [HBB22; MGGM18].

In addition to the benefits for the user, carpooling has also a positive environmental impact. By sharing a car with others, the traffic density is reduced and commuting becomes more efficient, while at the same time, the amount of CO2 produced per person decreases [LZW21; MGV20].

1.2 Objectives

With this work, we want to show that it is possible to create an authentication system, which enables a carpooling service, to operate without knowing the actual identity of the owner. This is shown by giving a concept, supported by a minimal prototype.

1.2.1 Identification

A correct and trustworthy[1] identification is important to all users of a carpooling system, especially if you share a car with strangers, it is important that all customers are trustworthy. This identification method should work in most European countries and be completely digital. However, it should be easily expandable to other countries.

1.2.2 User Management

After identifying a person successfully, an account needs to be created and managed. Besides the management of the user information, the prototype shall be able to provide an authentication method. The goal is that this account is usable by carpooling platforms, which want to set their focus on privacy.

1.2.3 Reputation

Because reputation and ratings are critical information about a user, if no other information is known, the authentication system must provide reputation capacities. Further, that reputation must not give any private information about a person and must not be linkable to a real identity.

[1] Carpooling Platform Users must trust the system

1.3 Conditions

Before we can start to focus on the actual topic, we need to define the precondition for this project research as well as its desired postcondition.

1.3.1 Precondition

Because we target to create a user management and authentication system, we do not have a direct influence on the carpooling platform itself. However, therefore we assume, that the carpooling platform provider, actually has an interest in proving anonymity and cooperating in adapting their system. For now, we assume that there exists a platform that operates on smart contracts, running on the Ethereum Blockchain. More about Ethereum is provided in Section 2.3.3. The predefined contracts, which we assume can be found in the attachments.

1.3.2 Postconditions

Before we can take a deeper look into the current state of the art and our concept, we need to define some requirements, which are required for providing a carpooling platform and much more what requirements the authentication mechanism must fulfil.

The main goal of the platform is that users can use it in an anonymous way. This, however, can lead to some security problems. If no user can be tracked back, it leads to misuse of the service or even results in a dangerous situation for the travellers [KT21]. This includes damaging the provider's vehicles, stealing someone else's belongings or other serious crimes, which leads us to the first requirements. Every user, who wants to register with the service for the first time needs to provide their actual identity in order to allow future punishments if needed and to ban people from the platforms if required.

Because the platform is focused on privacy and it shall not be possible to track someone's action even with external information, no identifiers can be used, which could lead to cross-platform tracking and even enable criminals to gather other private information via email tracking [HBFL18]. In particular, this means, that neither an email address nor telephone number can be used since those identifiers are typically used on other websites or platforms and therefore the use of those platform information in combination with driving data could lead to gaining additional information about a person. This means we need an authentication protocol that enables users to log into their accounts without the usage of an email or telephone number and at the same time is secure.

Moreover, it shall be possible to rate or report users, who misbehave on the platform and to engage providers and travellers to behave in an optimal way. Those ratings must provide complete privacy, which means that the rater as well as the ratee must not reveal their real identity to each other. Further, if a user creates a new account, all the old ratings must get transferred to the new account and the old account must get deactivated [HBB22].

In this work, we focus on Europe and the European market and solutions. Nevertheless, the created concept and prototype shall be expandable to other countries outside of Europe.

Lastly, the system should run in a distributed way, so that no components gain any power on the system based on their infrastructural role. Table 1.1 gives us a brief overview of all requirements and assists us in keeping track of and referencing the requirements later on.

#	Requirement	Description
1	Anonym while taking rides	The user must be able to use the service and book car rides, without exposing any of their related bookings
2	The person's identity must be checked	Anonym car sharing opens the door to crime, whether it is between passengers or towards the provider's property. Therefore the identification must be verified on register time
3	Access to Personal Data	It must be possible to notify and share the identity with location police in case of a serious crime
4	Users must be rateable by others	To engage, all users to behave properly, it must be possible to rate users and punish or reward them user's with a rating
5	Anonym Authentication	For signing up and login no email must be used because it would allow to track users across sites and break the anonymous feature
6	Distributed Deployment	The system should not depend on the infrastructure, which is provided by a single party
7	Anonym rater and ratee	The ratee should not know, who rated them and the rater should not know the real identity of the ratee
8	Reputation transfer	If a user creates a new account, the old ratings are transferred to the new account
9	Geographically expandable	The concept must be expandable to other markets if needed

Table 1.1: Requirements

1.4 Structure

To achieve the named objectives, we start in chapter 2 and 3 by looking at the required background information and the state of the art for introducing and establishing the final concept in chapter 4. Afterwards, in chapter 5 we show the technology that is used by the prototype for archiving our objectives. Lastly, we evaluate the prototype based on the predefined requirements, and the security aspects. Furthermore, we show how the prototype can be extended in future.

2 Background information

The foundation of this paper are well known and established algorithms and technologies. Therefore, we start by giving the background information, which is required to make it easier to understand the state of the art and the concept later on.

2.1 Cryptography

Because a lot of the used protocols and technologies rely strongly on cryptographical concepts and algorithms, we start by introducing the cryptographical concepts, which are required for a better understanding of the following sections.

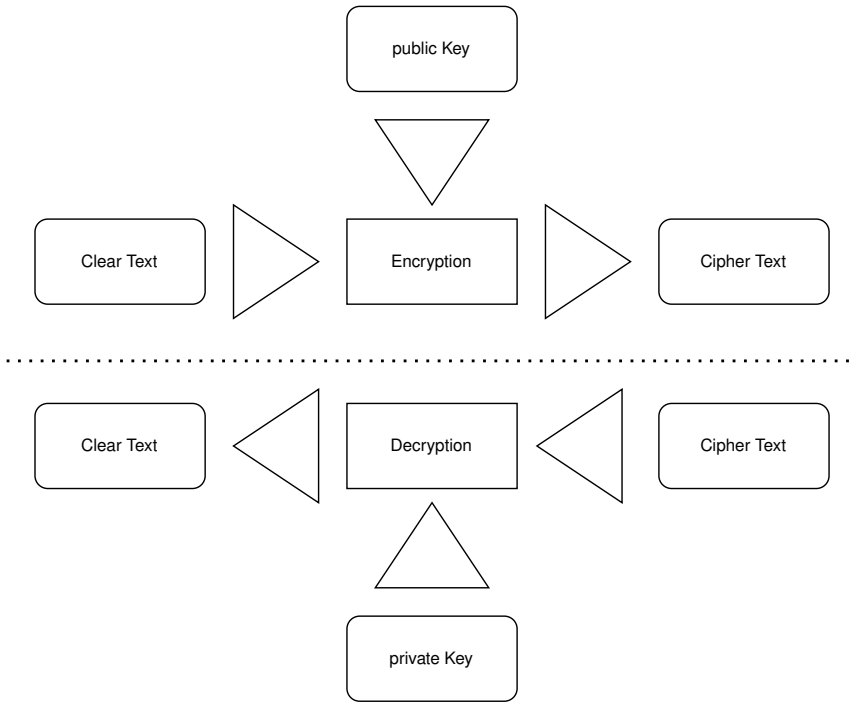

Figure 2.1: Asymmetric Encryption

2.1.1 Asymmetric Cryptography

Asymmetric cryptography is based on the principle that we have two different and distinct keys, as shown in Figure 2.1. One is publically available, and the other is only held by one particular party and must not be known by another party. The most common asymmetric cryptographical algorithms are RIVEST–SHAMIR–ADLEMAN (RSA) and Elliptic curve-based algorithms. RSA is based on factorisation problems, which state that it is not possible to factorize a large number efficiently by a classical computer. The algorithm is shown in Figure 2.2. Elliptic curves on the other hand are based on the discrete curve logarithm problem, which is mathematically harder to solve than the factorasation problems. Further, an elliptic curve is of the form $y^2 = x^3 + ax + b$ over a finite field [Bon+99; Sic18; Sul13].

Both Algorithms are widely in use nowadays, i.e. for encrypting the web traffic or creating digital signatures. Moreover, they have been analysed for quite some time. In practice, however, those algorithms are not that easily implementable, because they require specified padding on the input in order to be secure, which is defined in their specification.

2.1.2 Symmetric Cryptography

While asymmetric encryption uses two different keys for encryption and decryption, symmetric encryption uses just one key for both operations, which is illustrated in Figure 2.3.

The most widely adopted asymmetric encryption algorithm is ADVANCED ENCRYPTION STANDARD (AES). Which was standardised by NIST in 2001. AES uses matrix operations for transforming a clear text into a cipher text. The algorithm is divided into four phases, which are repeated multiple times and uses a S-Box. The S-Box is a predefined matrix constant. In those phases bytes get substituted ('SubBytes' phase), matrix rows shifted ('ShiftRows' phase), columns mixed ('MixColumns' phase) and lastly, the key is added to the matrix state by an xor operation ('AddRoundKey' phase) [NIS23].

$$N = pq$$

Find e with:

$$gcd(e, (p-1)(q-1)) = 1$$

Encryption:

$$M = m^e \ mod \ N$$

Decryption

$$d = e^{-1} \ mod \ (p-1)(q-1)$$

Figure 2.2: RSA Algorithm

Furthermore, AES is a block cipher, which means only a fixed length of bytes can be encrypted. For AES it is 16 bytes. This means shorter clear text must be extended by adding padding and longer clear texts split into blocks of 16 bytes and extended by padding if it is not dividable by 16 bytes [Dwo07; NIS23].

The way multiple blocks are linked together is called the cipher mode. For AES multiple cipher modes are defined, which all have different properties and advantages and disadvantages. The most widely used mode is GALOIS/COUNTER MODE (GCM). GCM has the big advantage, that it not only encrypts the clear text but also guarantees the integrity of the cipher text, also known as authenticated encryption. This prevents attackers from manipulating the cipher text. Furthermore, GCM uses an Initial Vector, which prevents two identical clear texts from producing the same cipher text. If taking a closer look at AES GCM it actually behaves like a stream cipher and less than an actual block cipher [Dwo07].

A stream cipher is basically an algorithm that generates a unpredictable, but infinitive bit stream. This bit stream is the encryption and decryption key. Each bit at position n is xored with the n bit of the clear text. Because the bitstream is unpredictable, also the ciphertext is unpredictable. Another Stream Cipher is ChaCha20, which is an alternative to AES but is much simpler.

Instead of using complex matrix operations, ChaCha20 is just using addition, shift and xor operations for generating a key stream. This leads to much faster encryption and decryption speed compared to AES and a much simpler implementation [DSS17; NL15; Sul15].

However the algorithm is much younger than AES and therefore not fully researched yet, but likewise to AES there exist no efficient attacks of ChaCha20. With the Poly1305 extensions, ChaCha20 can provide authenticated encryption, such as AES can [DS17; Sul15].

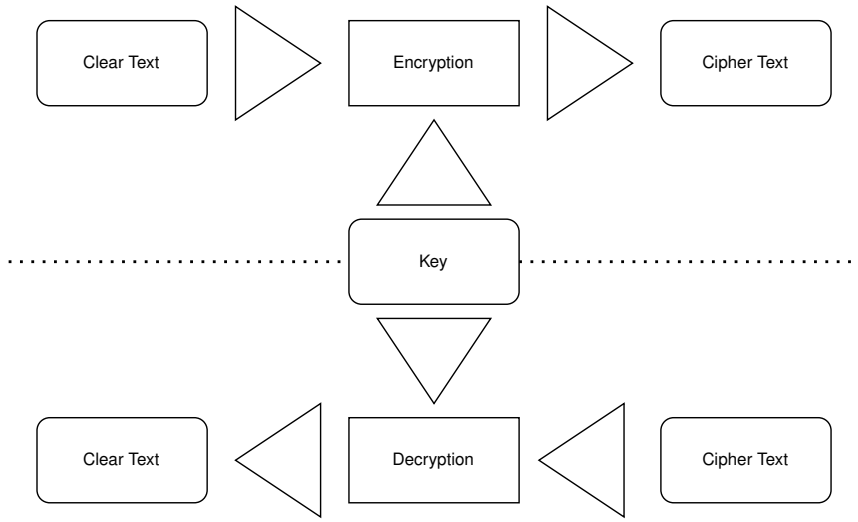

Figure 2.3: Symmetric Encryption

2.1.3 Hash Algorithm

In comparison to encryption algorithms, which provide a two-way function, hash algorithms, however, are a one-way function. That algorithm can take an input of any input length and convert it to an unpredictable and fixed-length output. For the same input, the output always stays the same. Because of the property that the length is reduced, there exist multiple inputs, which produce the same output - also called hash collisions. A good hash function needs to prevent that coalition as well as possible and at the same time guarantee that it is impossible to convert the output back to the input [Dwo15].

Hash Algorithms, which are recommended and widely in use nowadays for secure hashes are SHA2 and SHA3, from the SECURE HASH ALGORITHM (SHA) family. Both algorithms can produce an output of length 256bit, 384bit and 512bits and are standardized by Nist [Dwo15]. While SHA2 uses Merkle-Damård constructions, similar to older hash algorithms, which are already proven as insecure, SHA3 uses a different approach, the Keccak Sponge construction [Dev22a].

2.1.4 Cryptographic Signatures

Cryptographic Signatures are based on asymmetric cryptography. The signer holds the private key, while all other parties can validate the signature by using the public key of the signer. Only the private key can be used to sign a message. Common for cryptographic signatures are a reverse RSA scheme and Elliptic curve-based algorithms. Because asymmetric encryption schemes have an upper limit of a message length they can sign, the message is hashed before signing [KK12].

The most widely used signature algorithms are SHA256-RSA and SHA256-ECDSA. Both algorithms hash the message to a 256-bit output and afterwards perform either RSA or ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) on the hash. The final output is the signature. The validator requires the message and the signature. Then it hashes the message and checks if the signature matches the hash when the reverse signature function is applied to the signature [KK12].

2.1.5 Fully Homomorphic Encryption

FULLY HOMOMORPHIC ENCRYPTION (FHE) are asymmetric cryptographic algorithms, which have the ability to operate on encrypted data, without knowing the private key to encrypt the data. Basic RSA behaves homomorph under multiplication but loses this capability if the mandatory padding is added to the cipher. FHE algorithms, however, are homomorphic under addition as well as multiplication, which allows performing complex operations on the encrypted data [BBB+22].

Such algorithms are under active research and are interesting for deep learning and analysing sensitive data, without revealing the actual data. Those algorithms, however, use approximation and therefore lose accuracy with every performed operation. Additionally, they are slow compared to other cryptographic algorithms [BBB+22; Gen09].

```
GET /path HTTP/1.1
Host: www.uni-stuttgart.de
Accept-Language: en
```

Figure 2.4: HTTP request

```
HTTP/1.1 200 OK
Date: Sat, 10 Sep 2023 12:32:05 GMT
Content-Length: 2342
Content-Type: text/html

<body of 2342 bytes here >
```

Figure 2.5: HTTP response

2.1.6 Zero Knowledge Proof

Zero Knowledge Proofs are a concept to prove a statement without revealing any information, except the response to the statement. There exist two main types of Zero Knowledge Proofs. Firstly, there is the interactive Zero Knowledge Proof (IZKP). It requires that both parties, actively participate in the proof of data possession, which requires multiple rounds of messages. This is highly inefficient [eth23].

The second type, Non-Interactive Zero Knowledge Proofs (NIZKP), solves it by only requiring a single round of messages. Additionally, IZKP requires each party pair to prove a statement in an additional proofing session. NIZKP, on the other hand, only requires that proof is made once for a single party. Afterwards, the proof is publicly available and can be used by other parties, which have not been part of the creation process at all [eth23].

2.2 Protocols

The upcoming sections make use of some widely used protocols, for this reason, we shortly introduce the four most important ones.

2.2.1 TLS

TRANSPORT LAYER SECURITY (TLS) is a protocol, that targets to encrypt the traffic between two parties on the Transportation layer. It is widely used to encrypt the traffic on the web and is the successor of the SECURE SOCKETS LAYER (SSL) protocol. This protocol utilises asymmetric encryption to exchange a symmetric key, which is then used to perform authenticated encryption on the messages. Only the sender and receiver can decrypt the message. And user's certificates for proving the ownership of a public key [Clo23].

In practice, the server owns the private key and obtains a certificate from a CERTIFICATE AUTHORITY (CA). To ascertain that a certificate is valid, cryptographic signatures are applied. Each certificate can be signed by another certificate, which constructs a hierarchy of certificates, which is described as a PUBLIC KEY INFRASTRUKTUR (PKI) [Clo23].

2.2.2 HTTP

The HYPERTEXT TRANSFER PROTOCOL (HTTP) is operating on the application layer and has the task of transferring hypermedia documents. It is a stateless and text-based protocol, to enable structured communication between a server and a client [con23a].

This protocol uses a request-response pattern, which means that each request receives exactly one response, such as shown in Figure 2.4 and 2.5. Each request starts with an HTTP Method, such as GET, POST, PUT, POST and the resource path with which a client wants to interact. The response must start with a status code ranging from 1xx-5xx. After the leading line, headers follow, which describe the metadata of the actual message. After the header block a line breaks and an empty line follows. After that empty line, the actual message is printed [con23a; con23b; con23c].

The GET method is used for retrieving a resource from a server, the POST for creating a resource and the PUT method updates a resource at a given path. The DELETE method is used when a resource needs to be deleted. The GET method is the only method, that usually does not contain a request body, which allows it to cache GET and improve efficiency if a resource is retrieved multiple times within a short time window [con23b; con23c].

If HTTP traffic is encrypted via TLS, it is called HYPERTEXT TRANSFER PROTOCOL SECURE (HTTPS) [Clo23].

2.2.3 REST

A REPRESENTATIONAL STATE TRANSFER (REST) APPLICATION PROGRAMMING INTERFACE (API), is a loose standard and type of application interface, that provides access to a server's business logic via HTTP calls with predefined resource paths. Those paths are also called endpoints and use the typical HTTP methods, such as POST, GET, PUT and DELETE for interacting with the resources of the server. This enables another application to remotely interact with the given application via its REST API [Lim20].

It is usually the middleware between a client and a server-based application. Its task is to authenticate and restrict the operations, which a client can perform on actual data. Moreover, it serialises and deserialises the requests and responses, usually from and to JAVASCRIPT OBJECT NOTATION (JSON), illustrated in Figure 2.6. Because Rest APIs work on top of the HTTP protocol it is possible to use HTTP-based mechanisms and security measurements [Lim20].

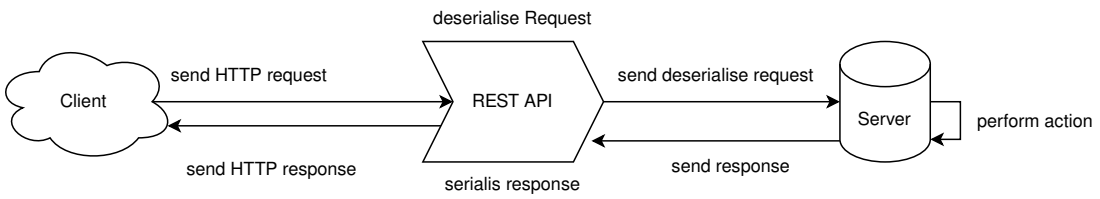

Figure 2.6: Rest API

2.2.4 NFC

NEAR FIELD COMMUNICATION (NFC) is a wireless data transfer technology, that enables sending information from one device to another at a maximum distance of about 10cm. One of the devices can also be passive, which means it does not need a battery source to operate. Instead, the active party provides the required energy to the passive device when interacting with it. This is used by the card reader and the corresponding card, where the card is the passive component [MLKS08].

2.3 Data Store

Now we take a look at the latest technologies for persisting data and establish the foundation regarding them.

2.3.1 Relation Database

Relational Databases are storage system, which manages data in tables. Each table has columns, which define the structure of the datasets in the table. Each row of the table is an individual dataset. By using SQL queries it is possible to interact with the data. Because of this such databases are also called SQL Database [IBM23a].

Moreover, it is possible to link rows of different tables with each other. This linking is a relation between two tables. This is archived by defining primary and foreign keys. Each table needs to have at least one primary key column. The value of the primary key columns must be unique. Other tables use foreign keys to reference the primary key of another table, as illustrated in Figure 2.7 [IBM23a].

Relational Databases are transaction-based. Those transactions fulfil the ACID properties, namely atomicity, consistency, isolation and durability. The atomicity property guarantees that all data-changing operations are executed as if they were a single operation. Meaning, either all changes or no changes are performed on the stored datasets. By satisfying the consistency property all data stays consistent and replicas have the same state as the original. This is supported by the durability property, which allows data to stay persistent even on the occasion, that the system fails. Lastly, all operations are performed isolated, guaranteed by the isolation property [IBM23a].

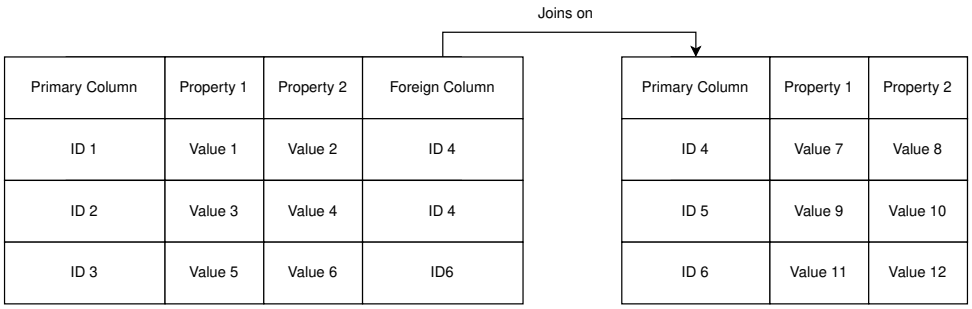

Figure 2.7: Relational Database

Examples of SQL Databases are PostgreSQL[1], MySQL[2], MariaDB [3], and SQLite [4].

2.3.2 NoSQL Databases

SQL-based Databases are working well on smaller datasets with a fixed structure, where a high level of consistency is needed. They do not scale in a distributed way. Moreover, they are also quite inflexible. For every small change in the structure, the whole tables would need to be updated. Those problems are addressed by NoSQL databases. [IBM23a; MK19].

Those databases drop the ACID properties and instead focus on high availability and scalability. This means that NoSQL database states are not consistent and their transactions are not atomic. Further, they target the CAP problem. This problem describes, that a distributed database can only fulfill two of the three properties, Consistency, Availability, or Partition Tolerance [IBM23a; MK19].

Currently, NoSQL databases are split into four categories. Key-Value Stores, Wide-Column Stores, Document Stores, and Graph Stores. The key value store is the most simple of the four. It stores a value, which can be only addressed by its key. Datasets are only callable by their key and no search on the actual data can be performed to obtain a set of values, matching the search criteria. Wide-Columns Stores are similar. But they allow the value to be semi-structured. Instead of a large value, the value can be split into several columns. Unlike SQL Databases, those columns are not searchable and therefore can not be used to create relations between different datasets. The advantage of the Wide Column Database is that not the whole value needs to be returned. Instead, the client can specify which columns they want to retrieve [IBM23a; MK19].

Examples of key-value stores are Riak[5] and Redis[6]. Cassandra[7] and HBase[8] are examples of column-based stores.

Document Stores on the other hand store JSON, , documents. Databases such as MongoDB[9] allow you to search on those JSON similar to SQL-based Databases. However, all documents can have different structures and different documents can not be linked to each other [IBM23a].

Lastly, Graph Stores does not use the classical table or key-value structure like all the three other types. Instead, it stores the data as a large graph and applies Graph theories to search for the data. Each dataset is stored as a node. In Neo4J [10], a Graph Store, both the edges and the nodes can have properties [IBM23a].

[1] https://www.postgresql.org/
[2] https://www.mysql.com/
[3] https://mariadb.org/
[4] https://www.sqlite.org/index.html
[5] https://www.tiot.jp/riak-docs-beta/riak/cs/3.1.0
[6] https://redis.io/
[7] https://cassandra.apache.org/$_{}$*index.html*
[8] https://hbase.apache.org/
[9] https://www.mongodb.com/
[10] https://neo4j.com/

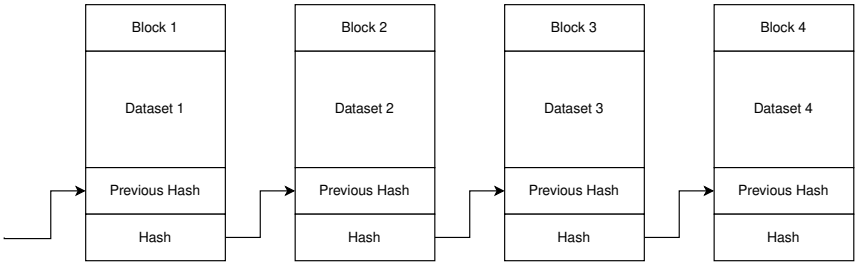

Figure 2.8: Blockchain - Hash Chain

Because of their replication properties, NoSQL Databases can be used to manage data across different data centres, and therefore guarantee a disaster recovery and an improved latency performance [IBM23a; MK19].

2.3.3 Blockchain

Blockchains are a modern way of storing data. They intend to store data in a distributed way and at the same time guarantee that the data is immutable. Meaning once the data is written to it, it stays there forever. Each node in this Blockchain network is called a ledger node [IBM23b].

This is archived by applying cryptographic signatures along with hash values. Data is stored in blocks, and each block computes its hash value based on the stored data. To archive immutability each block contains the hash value of the previous block. The linking of blocks would require that all previous blocks be changed in order to change the current block, as shown in Figure 2.8. The longer this chain becomes, the harder it gets to manipulate any values [IBM23b].

For submitting a new block a transaction is performed. Each transaction usually contains the data itself, the hash and the corresponding signature of the party, which wants to add some value to the chain. This signature is validated by the ledger node and only if the signature is valid, the new block is accepted and chained to the previous block. The private key is stored by the party itself. For making storing those keys simple, there exist wallets, such as Metamask[11]. Those wallets manage the private keys of a party and make it more straightforward for the user to interact with various Blockchains [IBM23b; Sic23b].

Because hash functions are usually quite fast, Blockchains add an extra mechanic to slow down transactions. This mechanic is either 'Proof of Work' or 'Proof of Stake'. Which mechanic is applied depends on the individual Blockchain itself [cor23; IBM23b].

Proof of Work demands that the parties solve increasingly challenging problems, to guarantee that the block they own is valid. This process is called mining. Each miner is rewarded, usually with some cryptocurrency [12] as a transaction fee, if they mined a block successfully. This creates competition between different miners to mine the next block, which ensures that they do not try to manipulate the blocks. By obtaining more than 51% of the blocks, a miner would be able to manipulate the block easily [cor23; IBM23b].

[11] https://metamask.io/
[12] virtual currency with is specific to one blockchain, i.e. Eth for Ethereum

Because the Proof of Work concept is quite computationally and energy-intensive, some Blockchains, such as Ethereum[13], have started to deploy the Proof of Stake concept. Proof of Stake-based Blockchains allows the party with the highest credibility to add a block to the chain, instead of the party with the highest computation power. This credibility is received by owning the highest stake in the ledger network. The idea is that the party, which has the highest stake does not want to lose their credibility and therefore does not tamper with the Blockchain [cor23].

Most Blockchains allow interaction with the dataset and transactions via a defined set of instructions or small programming languages. Those are called smart contracts. Smart contracts are small programs, that are deployed to the Blockchain and then manage and interact with assets, which are blocks, on the blockchain. [IBM23b].

Blockchain can be either private or public. Everyone can join the network and read or write data to it if the Blockchain is public. A private Blockchain on the other hand is managed by a single organisation, which can decide who can join the network. More advanced Blockchains use permissions, to give different parties different reading and writing permission. Usually, those permissions are managed by a central organisation, but technically it would be possible that permission-based Blockchains are public too. Lastly, there are Consortium Blockchains. Those are managed by multiple organisations, which decide, who can add and read data from the chain. [IBM23b].

[13] https://ethereum.org/en/

3 State of the Art

After introducing the required background information, we inspect the current state of the art. Therefore we will focus on the identification, authentication, and user management along with the reputation aspects.

3.1 Identification

The core of a secure and anonymised platform is the identification of new people, who join the platform. Hence, we take a look at some of the most common ID Systems in Europe. For simplicity, we focus on Germany and Sweden as an international reference [Grö10; PGH21]. More ELECTRONIC IDENTIFICATION (EID) providers in other countries are listed in table 3.2. But firstly, we need to take a brief look at identification properties used in Germany respectively in Sweden.

Property	Description
Date of Birth	The date when the person was born and given its name
Place of birth	The town or city in which a person was born
Current place	The town or city in which a person is living in at the moment
Name at birth	The name which was given to the person when born
Current Name	The name which is given to the person at the moment
Sex at birth	The assigned sex at birth
National identification number	A unique number, given to a person at birth, which is valid for their entire life, i.e. Tax-ID[1] in Germany or Personnummer[2] in Sweden
Eye Colour	The colour of the person's eye
Height	The height of a person

Table 3.1: ID properties used for identification

[1] https://www.bzst.de/DE/Privatpersonen/SteuerlicheIdentifikationsnummer/steuerlicheidentifikationsnummer_node.html
[2] https://skatteverket.se/privat/folkbokforing/personnummer.4.3810a01c150939e893f18c29.html

Table 3.1 lists international strong and weak id properties printed on the id cards or stored on the id card chips [FC23] or exist and are in use otherwise. The strong properties are the national ID number as well as the joined property of given name, date of birth, sex and place of birth. Those properties can be considered unique. Some properties such as the name can change over time, therefore the given properties at birth are taken into consideration if it is important to have static ID properties. The eye colour and height are weak properties because they can slightly vary and can be hard to verify online. The strong attributes are used by authorities and therefore printed onto the physical card, comparable Figure 3.1 and 3.2, and stored on the EID chip [FC23; Sic20].

Moreover, the GENERAL DATA PROTECTION REGULATION (GDPR) define PERSONAL DATA (PD) for the EU and how those data need to be protected. Similar to the United States, which specifies PERSONAL IDENTIFIABLE INFORMATION (PII) [Thu19]. The GDPR differentiates those properties into seven categories. Namely, those are 'physical, physiological, genetic, mental, commercial, cultural and social identity' as stated by the GDPR [AG23b]. Those properties range from abstract numbers such as telephone numbers and personal identification numbers to appearance information such as eye colour and hair colour. Some more information about the data, which belongs to such regulation is illustrated in Figure 3.3 [AG23b; Thu19].

First of all, there is not a unique system, which can be used across all EU countries. Instead, each country uses its own ID System. So, in Germany, VideoIdent is the most common Identification process, while the electronic ID card has started to establish itself. In countries like Sweden and Norway BankID is the most common ID system. More than 99% of the Swedish population use BankID [Ban23]. In Germany, while 70% of the population have access to the electronic ID card, only seven per cent are using this EID system [Pri21]. The video ident procedure is quite common, but also highly insecure, especially with the accelerated progress of AI-based software [Mar22].

In detail, the German EID cards use an electronic chip, which stores the data encrypted. When a party wants to read the data, it requires the user to read the chip with NFC-capable devices and enter their passcode. Afterwards, the encrypted data is sent to an EID Server, which decrypts the data and sends the data to the requesting party. Each transaction has a unique session ID [KG23b].

BankID on the other hand uses a traditional PKI. This means the person owns a private key, which is either stored on a bank card (BankID på Kort), on file (BankID på fil) or in the secure app storage of a mobile phone (BankID på mobil). The actual verification data and certificate is stored on the

Figure 3.1: Swedish ID card [swe21]

Figure 3.2: German ID card [Inn23]

BankID server. When a party wants to read that information, it requests an identification process, and the user signs this request with its private key. Afterwards, the requesting party can fetch the information of the particular user from the BankID server [Grö10].

All the provided solutions are centralized solutions and therefore they are quite powerful. Also, parties that are performing the identification have access to highly sensitive information. Therefore, a high level of trust is required, which means that there should exist requirements, that the party needs to fulfil in order to be listed as a trusted service provider. Within the EU there is a standard as Qualified Trusted Service. In Germany, the Bundesamt für Sicherheit in der Informationstechnik (BSI) is responsible for the legal requirements, the TÜV-IT for granting permission, if own infrastructure is used, and D-Trust[3] is issuing the certificate. Addionally, the 'Vergabestelle für Berechtigungszertifikate (VfB)'[4] needs to grant permission for the certificate application [D-T23b; Gro16; Sic23c]. The process for obtaining such a certificate is expensive and requires a lot of time. Alternatively, another Qualified Trust Services can perform the identification. Moreover, the strict requirements only refer to trusted services such as handling ID card verification itself. If using an external ID card verifier, such as D-Trust, such a certificate is not required [D-T23a]. However, there usually exists a trust agreement between the ID card verifier and the party, which wants to verify a person's ID card.

In addition to that widely used national solution, international solutions for the EU are planned by using the EU Wallets. International EIDs fall under the ELECTRONIC IDENTIFICATION, AUTHENTICATION AND TRUST SERVICES (EIDAS) regulations. There have already been German prototypes in the past based on Blockchain. Those however failed because of a lack of

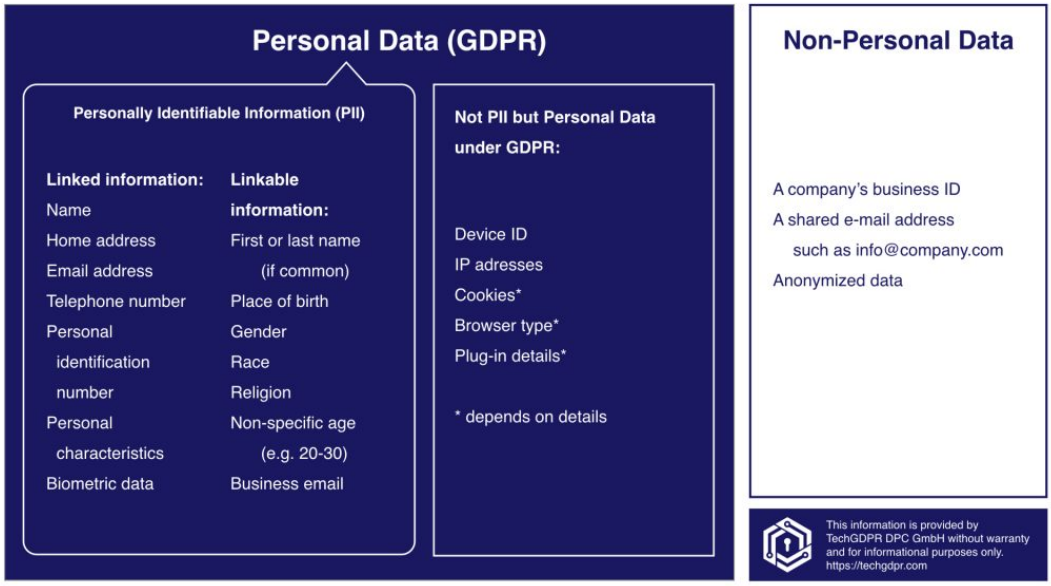

Figure 3.3: PII and GDPR data [Thu19]

[3] https://www.d-trust.net/de/loesungen/identifizierungsdienste
[4] https://www.bva.bund.de/DE/Services/Behoerden/Produkt/PA-VfB/pa-vfb_node.html

security. Meanwhile, more projects have been started for creating a collaborated solution across multiple countries. Those projects target a framework, not an implementation by itself though. Such wallets are often described as Self Sovereign Identities [Com23b; Rau22; Reu23; Voß21].

The Self Sovereign Identity is an approach in which the owner of the data holds the data, with proof of correctness. If another party needs the data, the owner can decide for themself whether to give the other party the data or not. Also, zero-knowledge proofs are typically used when Self Sovereign Identities are applied. Zero-knowledge proofs allow us to check if some data is valid, without actually getting to know the data. An example would be the proof of age. If someone wants to check if someone is above 18 years old, a zero-knowledge proof would allow them to check it without revealing the actual date of birth. The other party only received the information it requested, without getting more data than required [MGGM18].

This means, that even though the regulations by EIDAS 2 would require all countries to implement a digital wallet for across-country use, it is not required to use the same wallet across all countries. Moreover, it must be only possible that documents can get validated internationally. Even though the use of a unified framework may help, to target multiple wallets at once when implementing software, which uses those wallets [Tha23]. All in all, this requires us to handle different countries differently later on, when we construct a concept for the carpooling platform because there does not exist a unified ID system.

eID Provider / eID Product	**Country**	**Domain**
Swedish BankID	Sweden	bankid.com
Norwegian BankID	Norway	bankid.no
NemID	Denmark	nemid.nu
MitID	Denmark	mitid.dk
Freja	Sweden	frejaeid.com
Eid Belge	Belgium	eid.belgium.be
Personalausweis	Germany	personalausweisportal.de
Cartão de Cidadão	Portugal	autenticacao.gov.pt
Documento Nacional de Identidad electrónico	Spain	sede.policia.gob.es
Audkenni	Island	audkenni.is

Table 3.2: eID Providers

3.2 Authentication

After creating an account, users must be able to sign into their account without revealing personal information. This in particular means, that no identifiers such as e-mail addresses or mobile numbers can be used as identifiers. For archiving this there are two common ways. Firstly a pseudonym can be used along with a strong password. Secondly, asymmetric key pairs can be used for signing. When comparing both approaches, the pseudonym approach is simpler, however, it would make it hard to use 2-factor authentication, because this would require a mobile number or an e-mail address.

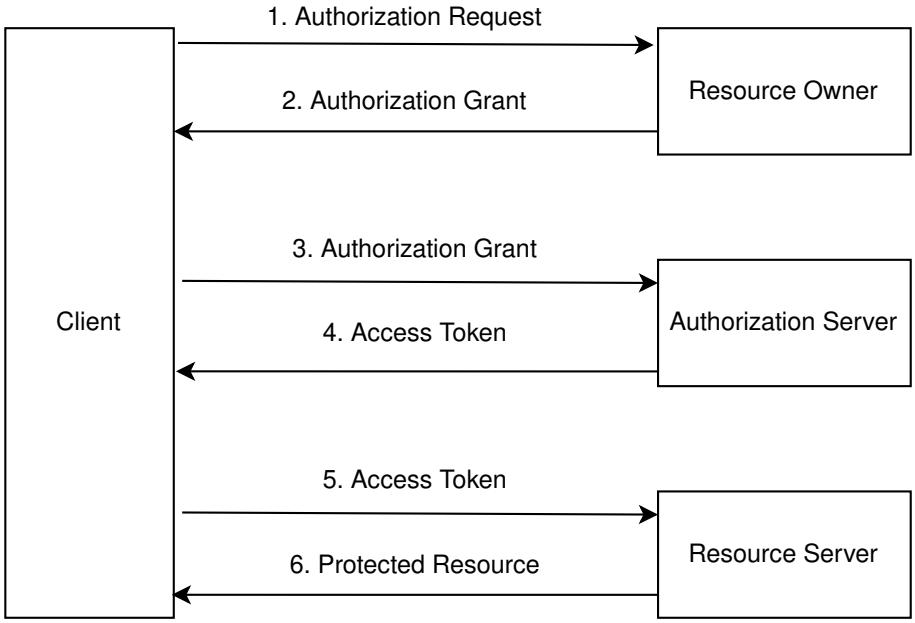

Figure 3.4: OAuth 2.0 flow [HPL23]

The same would take place when resetting a password. By using a private/public key pair we can increase security and anonymity, however, the user needs to handle the key management, similar to SECURE SHELL (SSH), which is used to log into a server passwordless [SSH23a; SSH23b].

Because we do not want to perform an authentication process with every server request, we need to use either session tokens or login tokens. Session tokens are quite common for simple website authentications but based on their server-binding properties, they are attached to the server, which stores the token. On the other hand, JSON WEB TOKENS (JWTS) are a common way to handle stateless authentication. While Session tokens do not store information by themselves, JWTS contain concrete information about the logged-in user. This again could decrease the anonymity, when tokens are collected and analysed [Bal17; BDW17].

Another approach and hybrid way is to use the OAuth 2.0 protocol, which essentially generates access tokens and stores the tokens in a database with attached information, which is required for validating the permissions. The permissions, which an access token is granting, is called the token's scope. In any case, no further information than the access token is transmitted between the client and server. Moreover, OAuth 2.0 is a quite common protocol, and because of scientific research, widely analysed against potential attacks [Har12; HPL23; Küs16].

The OAuth 2.0 protocol consists of multiple different components, as shown in Figure 3.4. Firstly, the client is authenticated with the resource owner by requesting the required data to perform one required authentication granting flow. OAuth 2.0 supports multiple different grant flows. The most common ones are the password grant flow, authorization code and refresh token flow.

When the password flow is used, the resource owner needs to provide their username and password. The authentication code flow uses a one-time password. This might be a 6-digit number or any other passcode, which is only valid for a very short period of time [Har12; HPL23; Küs16].

The refresh token grant flow uses a long-living token, the refresh token, which is returned by the authorization service on a successful authentication by another grant flow, i.e. the password grant flow. This refresh token is optional, which means the authorization servers, which protect sensible information, may not provide such a powerful token as the refresh token. However, the authorization server always returns an Access Token if the authentication is successful.

This Access Token is sent back to the client and has usually a short lifetime, in the range of a few minutes to a few hours. Afterwards, the Access Token expires and a new token must be retrieved by the client from the authorization server, with one of the given flows. But, as long as the token is valid, the client can access protected resources from a resource server. The owner of the resource server is usually another party as the authorization server, but can also be owned by the same party [Har12; HPL23; Küs16].

Besides the few given authorization flow, the protocol can be expended by further authorization grant flow. An authorization server is also capable of disabling authorization flows, which are not wanted by the owner of the authorization server. The user flow is based on an agreement between the client, the resource owner and the authorization server. We can use this when we use OAuth 2.0 for our authentication method [Har12; HPL23; Küs16].

3.3 User Management

When the users have identified themselves, we need to store the ID-related information. This information must be stored securely and must not be modified. The first idea that comes to mind is the use of Blockchain. Applications that rely on the use of a Blockchain easily solve the problems of data manipulation. However, this is only the case when used in large clusters. In case the application is stored on a too-small cluster, there is the risk of a 51% attack [KM23]. This attack allows it to manipulate about 38% of the Blockchain's contents, by winning the mathematical proofs based on the majority in computation power.

Additionally, Blockchains, especially public ones, provide us with an extra problem. When we insert a value to the Blockchain, the value is visible to all, who have access to the network. This would mean that sensitive information cannot be stored on the Blockchain, without encryption. And even by using encryption, there is the risk that a key can be leaked and all data can be stolen, because an update of that data is not possible. Instead of encrypting the data, it would also be possible to hash ID information. This would make it possible to store the data without handling any sort of key management but also opens up the problem that it would be possible to check whether a person is stored on the Blockchain, which violates the privacy of the users.

But do we even need Blockchain? In contrast to a Blockchain approach exist a centralised database. This ensures that no external party has access to those sensitive data. This, however, requires that everyone trust the user management provider and the integrity of the data because the data is not publicly verifiable nor transparently available. Privacy, on the other hand, is easier to provide with a centralized database. All that is required, is to encrypt the data for internal use. Furthermore no external party can easily check against public data. Another problem is the lack of data redundancy. When using a private database, especially a relational one, data redundancy is mostly not an inherited

feature of the database, which means the provider has more to do this additional work and everyone needs to trust that party, that it can recover data, i.e. in case of a server failure. To simplify the redundancy of data, NoSQL databases could be used [WG18].

When deciding between a blockchain-based solution and a non-blockchain-based one, there is often a tradeoff between privacy and transparency. While a private database makes it easier to keep private information safe, it does not provide any transparency.

Another approach is the use of permissioned and private Blockchains. Those limit other parties from accessing the Blockchain by restricting the write and read access to that blockchain. As such no sensitive data can be read by unauthorized parties. When adding an authorization component to the Blockchain, the transparency decreases. By adding an independent party to the private Blockchain, those concerns could be eliminated, depending on the required transparency degree. Moreover, private permissioned Blockchains can have a central component for managing the authentication process or even rollback transactions. In comparison to public Blockchains, where all data is replicated and stored on all nodes in the chain, authorized Blockchains can restrict the data that is replicated to particular nodes [LWX19].

Whether we use a Blockchain or not, we have to handle personal data. This requires encryption and integrity of data. Furthermore, the cryptography keys for encryption and integrity checks must be stored somewhere and exchanged somehow. When using encryption the transparency of the data fades. Only one party, which has access to the decryption key, can decrypt the data and have full transparency. This means that external parties need to trust the party with the private key. Because the requirements for performing the identification process based on sensitive ID information are quite high, it can be assumed that the party is indeed trustworthy for managing the data. If they would do any action, that harms privacy regulations, those parties would lose their permission to read ID information and are not able to operate anymore.

Another approach is to use the Self Sovereign Identities, which have been used to identify a user. By using Self Sovereign Identities, the personal data is stored decentral and no centralized organisation can manipulate the data or access it without the permission of the data owner [MGGM18].

Self Sovereign Identities combine three different technologies to archive this. Firstly, they rely on the user for a decentralized and immutable data store. The decentralized data store usable is a Blockchain, which can be either public or private, depending on the issuer of such Self Sovereign Identity documents. Such documents can be for example ID cards, driver's licenses, university degrees or other certificates. By utilising Blockchain for storing such documents, it is not possible to change the content of the documents, after they have been issued by the responsible organisation. Further, not even the issuing organisation can delete such a document [AG23a; cyb22; Okt22].

Secondly, Self Sovereign Identities apply the concept of decentralized identifiers. Those identifiers are used to establish a permitted connection between two or more parties. Such an identifier is a long pseudo-random generated string and does not contain any information about a particular user. For sharing the required documents in an encrypted way, each decentralized identifier is connected to one or multiple asymmetric key pairs. This way the parties can establish an end-to-end encryption. Further, only the sender and receiver know the actual shared data. Each user can produce as many decentralized identifiers as they want. They can even generate multiple for the same destination party and link different documents to one. Those identifiers are generated by the user and stored in the Blockchain as well [AG23a; cyb22; Okt22].

The third technique is the use of verifiable credentials. Verifiable credentials allow it to verify whether the data was issued and generated by a legitimate party, without requiring to ask that party for verification. This is archivable by using digital signatures and corresponding key management, such as a central or decentralized PKI [AG23a; cyb22; Okt22].

At the moment Self Sovereign Identities are under active discussion and development, but none of the prototypes and pilot projects is yet widely adopted or used. This, however, may change with regulation changes in Europe. Furthermore, in Europe, the necessary infrastructure can be provided by European Blockchain Services Infrastructure[5] Project [Com23a; Inf23a].

3.4 Reputation

When building a reputation system based on privacy and transparency, there is a wide range of consideration points, that need to be validated before applied and deployed. It is possible to group those into six categories, architecture, feedback, reputation score, aggregation model, attacks and costs [HBB22].

The architecture describes if a centralised or decentralised system is used, or even a hybrid of both. The Feedback and its subratings can be either stored as a set of values or as a range of results. Moreover, feedback can be split into chunks by increasing the granularity of the ratings. This is quite similar to the reputation score. A score can either be stored as a result, without storing the ratings themself, or as a set of ratings. Additionally, attributes such as liveliness, durability, visibility as well and monotonicity are important attributes of the reputation score [HBB22].

The aggregation model describes how the actual reputation score is calculated based on the given feedback. This could be tallied and averaged, using means or more complex formulas. Some may even consider aging reputations. Based on the different aggregation models and chosen reputation score models, a large range of different attacks are possible to manipulate own or others' ratings. Some are free riding, which allows them to receive ratings without actually providing any service. Another attack would be ballot stuffing, which allows one to improve the own rating by stuffing false positive feedback into the system. A related attack is the whitewashing attack. The whitewashing attack enables a user with a bad rating, to delete their account and reopen a new account with a fresh rating [HBB22].

The possible aggregation models also depend largely on how the data is anonymized and whether the rating is encrypted or not. In case the rating is encrypted the model depends largely on the used encryption scheme and its properties. For the purpose of running more detailed aggregation, a fully homomorphic encryption would be required and zero-knowledge proofs must prevent the user from submitting illegal ratings. Additionally, homomorphic algorithms such as RSA require padding to be fully secure, which would make RSA lose their homomorphic attributes. Besides that fully homomorphic algorithms usually contain errors with every computation, which would make rating inaccuracy when a large set of operations are performed. [HBB22; WNK20]

[5] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home

Lastly, costs are an important factor in reputation systems. Those depend on the needed amount of storage space, the length of messages and the complexity of the aggregation model. If the message length or quantity is large, the required bandwidth needs to be large too, which increases the cost for the operator of such a reputation system [HBB22]. Similar to the aggregation model, if a complex formula is used to recompute a user's rating based on a small change, the costs are increased for the operator.

In the survey [HBB22], a large number of reputations have been summarized and compared with each other. An overview and comparison regarding privacy and integrity is given in Figure 3.5. Those reputation systems use different approaches, which include Blockchain as well as token and pseudonym-based reputation.

Blockchain-based reputation systems use the properties of the underlying Blockchain, to achieve their privacy and security goals. Those reputation system does not rely on a trusted third party, which manages and validate the given ratings and computes them. Instead, everything is transparent and based on smart contracts [HBB22].

Token-based reputation systems on the other hand use cryptographic procedures to derive a pseudonym from a user instance and hide the actual identity of the user. Blind signatures are one way to generate such pseudonyms. Those are digital signatures, which mask the actual message, which is signed afterwards. This is either done by a central entity or by the ratee itself [Cha83; HBB22].

Transitory Pseudonym-Based Systems are another approach. It assigns multiple short-lived pseudonyms to a user. The link is only known by a central unit, which, however, has no more additional information than the link itself. That unit is often described as PSEUDONYM CERTIFICATION AUTHORITY (PCA). PCA can apply blind signatures to compute such pseudonyms, which allows it to transfer a pseudonym from one PCA to another [HBB22].

All in all, we can manifest that there exists a wide range of different reputation methods in a privacy-preserving environment. However, each has other requirements and preconditions regarding the system, which wants to use it. This means we need to take the reputation system into consideration when drafting our concept.

	Privacy							Integrity				
System	**Multiple Pseudonyms**	**User-Pseudo Unlinkability**	**Pseudo-Pseudo Unlinkability**	**Rater Anonymity**	**Ratee Anonymity**	**Inquirer Anonymity**	**Reputation Transfer**	**Unforgeability**	**Distinctness**	**Accountability**	**Authorizability**	**Verifiability**
Blockchain-based Systems												
Schaub et al. 2016	●	●	●	●	○	●	○	●	○	○	●	●
Bazin et al. 2017	●	●	●	●	○	●	○	●	○	○	●	●
Dou et al. 2018	○	●		●	○	○	○	●	○	○	●	○
Kang et al. 2018	●	●		●	●	○	○	●	○	●	○	○
Lu et al. 2018	●	●	●	●	●		○	●	○	●	○	○
Owiyo et al. 2018	●	●	●	●	○	○	○	●	○		●	●
Jo and Choi 2019	○	●	○	●	○	○	○	●	○	●	○	●
Liu et al. 2019	○	●		●	○			●		●	●	●
Dimitriou 2021	●	●	●	●	●	○	●	●	○	○	●	●
Token-based Systems												
Androulaki et al. 2008	●	●	●	●	●	●	●	●	○	●	○	○
Schiffner et al. 2009	●	●	●	●	●	●	●	●	○	●	●	○
Schiffner et al. 2011	●	●	●	●	●	●	●	●	○		●	○
Zhang et al. 2014	●	●	●	●	○	○		●		●	●	
Busom et al. 2017	●	●	●	●	○		○	●			●	
Proxy-based Systems												
Petrlic et al. 2014	●	●	●	●	○	●	○	●			●	●
Mousa et al. 2017	●	●	●		●	○	●	●		○	●	○
Signature-based Systems												
Bethencourt et al. 2010	●	●	●	●	●		●	●	●			
Guo et al. 2013	●	●	●	●	●			●				
Lajoie-Mazenc et al. 2015	●	●	●	●	●	●	●	●	●		●	●
Chen et al. 2016	●	●	●	●	○		○	●		●	○	○
Transitory Pseudonym-based Systems												
Miranda and Rodrigues 2006	●	●	●	○	●	○	●	●	○	○	○	○
Steinbrecher 2006	●	●		○	●	○		●	○		●	
Anceaume et al. 2013	●	●	●	●	○	○		●	○	○	●	●
Christin et al. 2013	●	●	●	○	●	○	●	●			●	
Other Systems												
Kinateder and Pearson 2003	●	●	●	●	●	○		●	○	●	○	○
Bo et al. 2007	●	●	●	●	○	○	○		○		○	○

Legend

●	Property satisfied
◒	Property partially satisfied
○	Property not satisfied
	Property not specified or not applicable

Figure 3.5: Overview of reputation system [HBB22]

4 Concept

After introducing the latest technologies and approaches, we take a look at the final concept. This concept is split into different subsections, which focus on identification, authentication, as well as user and rating management. The constructed component is called a 'Auth Service'. By design, it is possible that more than just one of those Auth Services exists.

4.1 Identification and Signup

The goal of this concept is to enable user management for a distributed carpooling platform. In Chapter 3.1, we have seen that different countries use different identification methods. This has the result that we cannot provide a uniform implementation for the identification process. Instead, we need to allow each country to implement its own optimised solution for the identification. By allowing external parties to implement their own solutions, we need to make sure that no harmful programs are used. This can be archived by a certification process introduced more in detail in the next chapter. Now let us take a look at how this would work for the German market.

When a new user wants to create an account, they need to have access to their physical ID card. This card needs to have the eID feature enabled. If those requirements are fully filled the user can request to open a new account. Afterwards, they are forwarded to the ID validator. In Germany, it

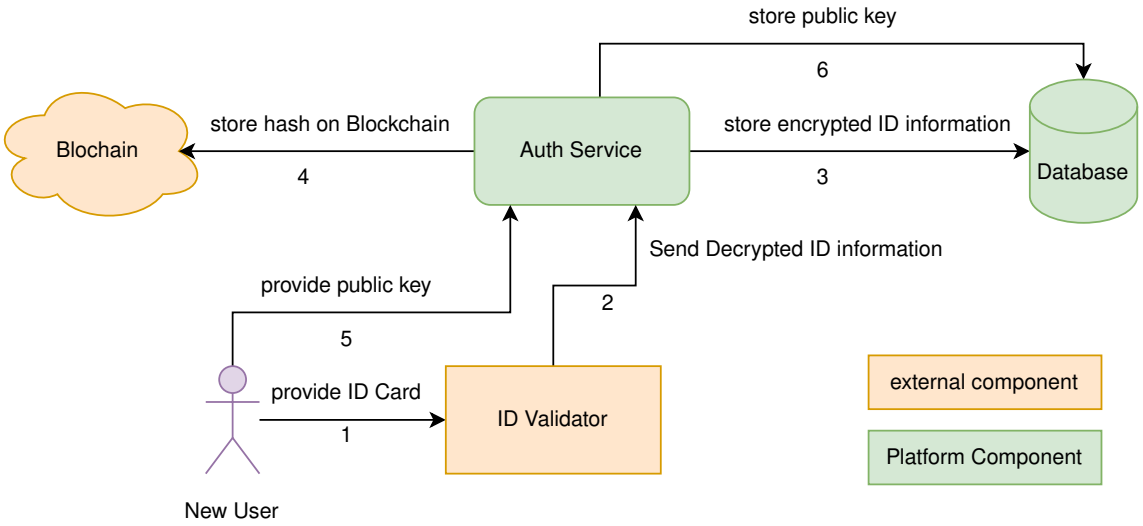

Figure 4.1: Registration Flow

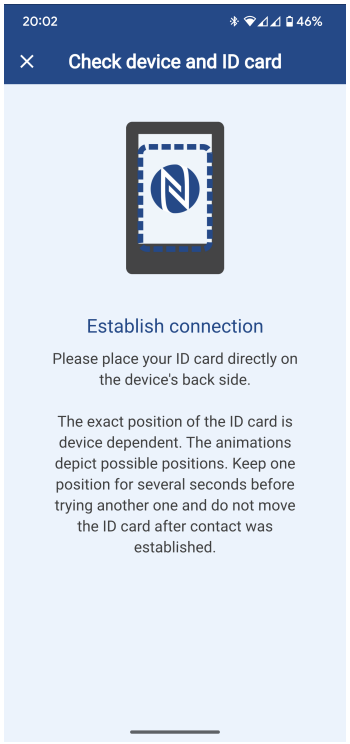

Figure 4.2: Ausweisapp

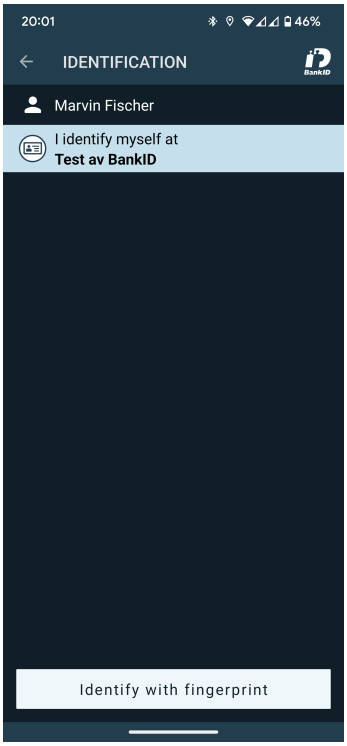

Figure 4.3: BankID Request

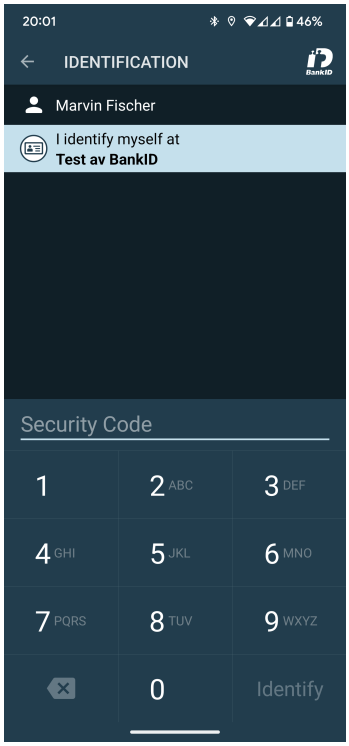

Figure 4.4: BankID Identification

is provided by the AusweisApp2[1], which looks to the user such as presented in Figure 4.2. This App reads the encrypted data from the ID card and sends it to a certified eID provider, comparable Figure 4.1. Afterwards, this eID provider sends the decrypted data to the Auth Service. The data is usually sent via a signed JSON Web token. However, the details and format can be different across different eID providers and are usually defined in an agreement between the eID provider and the identification requester.

For other countries, the ID validator can be changed. i.e., to BankID in Sweden. The flow is identical. When signing up, the user identifies themself by using BankID and their server is sending the response to the corresponding Auth Server. When using a BankID, the person provides their personnummer to the app and afterwards, the app sends an identification request to BankID. After the request is sent to the user, they open the BankID App and initialize the identification process. The identification is then done either by providing the registered fingerprint, as shown in Figure 4.3, or the person's security code, shown in Figure 4.4.

Because of the use of eID Services, all Auth Services must be trusted by the eID Validator. This means for legal and security reasons, it is only possible to use a semi-central solution. This is usually archived by issuing digital certificates.

[1] https://www.ausweisapp.bund.de/home

For test purposes, we use a dummy eID Service, which is simply implementable, comparable to Code Sample 4.1. The 'Authserver.Auth.Keys' module helps with dealing with cryptographic data. It signs the provided data, which then can be validated at the main backend logic. This means it would just behave like an additional different eID provider.

In addition to the identification process, each user generates a private-public key pair, which is later on used for authentication. Therefore, the user requests a signing challenge from the server and afterwards, the user signs the challenge. When the challenge is signed, the user sends it along with the public key to the server, where the server validates the challenges with the provided public key and stores the public key in the database if the validation is successful. This allows the user to sign into their account without any pseudonym or password.

```
1   defmodule Authserver.IdServer do
2     @private_key "-----BEGIN RSA PRIVATE KEY-----\n....\n-----END RSA PRIVATE KEY-----\n"
3     @hash_method :sha3_512
4     def parsed_key() do
5       Authserver.Auth.Keys.decode_pem(@private_key)
6     end
7     def sign_data(data) when is_map(data) do
8       json_data = :crypto.hash(@hash_method, Jason.encode!(data) )
9       Authserver.Auth.Keys.sign(parsed_key(), json_data)
10      |> Base.encode16()
11    end
12  end
```

Code Sample 4.1: Sample Dummy eID Server written in Elixir[2]

4.2 User Management

As soon as the identification is finished successfully, we need to store the information somewhere. For storing that information, we encrypt them and store them in a traditional database on the server side. At the same time, we compute a hash of the data and store it on a private and permissioned Blockchain. By storing a hash of that ID information, it is possible for other Auth Servers to validate if a user is already a user in the platform as a whole. This information is important to check whether a user has been banned because of improper behaviour.

The use of a permissioned and private Blockchain eliminates the risk of unauthorized access from third parties. Only Auth Services receive permission to read all ID hashes, stored on the Blockchain. At the same time only the Auth Service, which wrote an entry to the Blockchain can update it. The actual data is not stored on the Blockchain. This is due to security and privacy concerns. It should not be possible to retrieve all data from other Auth Services. Furthermore, it should only be possible to check whether another user was banned already. All under information is unimportant for a second Auth Service.

[2] https://elixir-lang.org/

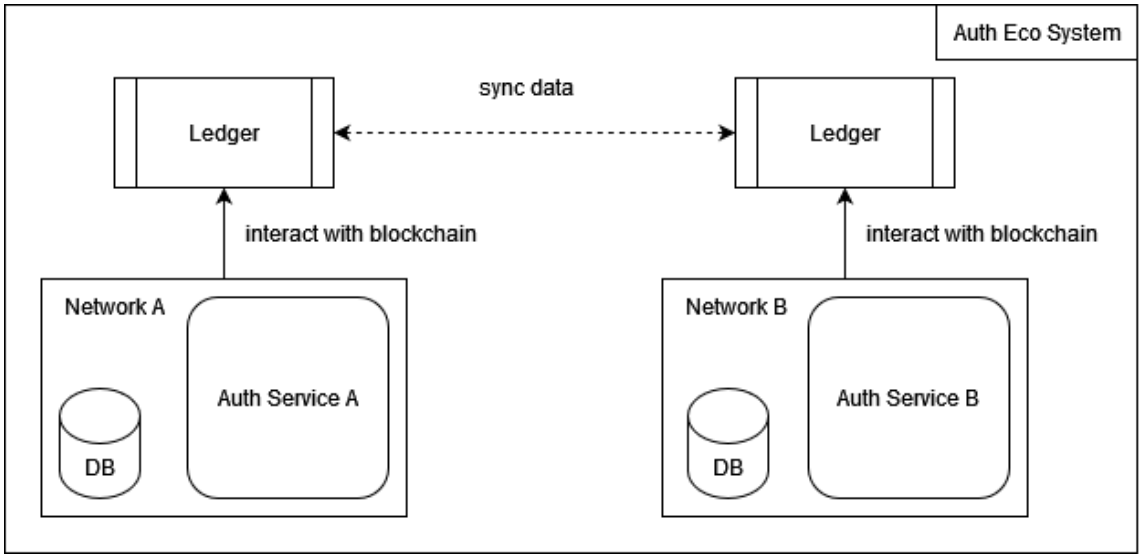

Figure 4.5: Auth Eco System

It is also worth thinking about if it is required to store the hash of all IDs on the Blockchain or if it is sufficient to store only banned IDs. The advantage is that each Auth Service provider can decide whether they want to allow a user to be registered with multiple Auth Services at the same time or not. In case all hashes are stored on the Blockchain the user's privacy would be decreased. This is because of the ability of a provider to check whether a user is already registered with some other provider. Or in case a user wants to change their provider, the old provider could Figure out easily to which provider the user has changed. When we stored only hashes of banned users the privacy of honest users is not decreased. Only if someone is banned on a platform, a hash is stored on the Blockchain and allows other Auth Services to decide whether they want to allow a banned user to register with them. For now, a ban would take global effect to avoid a user having multiple accounts.

This architecture leads to multiple isolated networks, which are owned by the individual Auth Services, comparable Figure 4.5.

Moreover, each Auth Service owns its ledger node, for sharing data on the Blockchain with other Auth Services. When a new Auth Service verifies itself as a trusted service it receives a certificate and private key for joining the ledger network. Only if a majority of all Auth Service providers trust the new service it can join and read from the Blockchain. In case a majority mistrusts an Auth Service provider, it gets removed from the Blockchain and cannot synchronize its ledger with the others. More about the used Blockchain can be found in the section 5.2.

4.3 Auth Service Requirements

For protecting the user's data and to regulate the relatively powerful Auth Services, there are a couple of requirements, which need to be fulfilled. Firstly, an Auth Service must support one of the eID services. This is connected with required verification and technical requirements. For reading data from the German eID card, it is required that the service has a permission certificate or a corresponding privacy agreement. Otherwise, it is not possible to request any data from the user.

Secondly, the potential Auth Service provider needs to get an SSL certificate with full organization validation. This is required for other Auth Services to identify the new identity and allow them to join the network or reject it. Moreover, the process of receiving such a certificate takes some time and reduces the risk of faulty providers, who just want to join the network for fetching data. The given expiration date on the certificate guarantees that each provider needs to redo the process in order to stay on the network.

Lastly, there is the requirement of a data protection officer, which is a legal requirement when handling a lot of personal information, and in some cases, it is required to receive permission to read the eID cards.

4.4 Authentication

At the same time a user is registering with an Auth Service, an asymmetric key pair is generated. The user stores its private key on the device and registers the public key with the Auth Server. This is done by a straightforward challenge-response protocol (CRP) [NIS17]. In particular, the server generates a random string, the challenge, which is sent to the user. Afterwards, the user is signing the given string and sends the signature and the challenge back to the server. The server now verifies the signature. Further, the server checks whether it has generated that particular challenge. If it is successful, the public key is added to the user's account and can be used for future logins.

Another approach would be to use a traditional public key infrastructure with key certificates. In that instance, the user would request a Certificate Signing Request (CSR), which the server validates and signs if the request is correct. The CSR itself contains a timestamp and a signature. However, this would be an unnecessary overhead for this use case.

When the user wants to log in in future, they request a login challenge and sign this challenge. This again is signed and used to retrieve an OAuth 2.0 token from the server. Only if the signature is valid and the corresponding public key is assigned to the user's account, a valid OAuth token is granted. This can be realized by implementing a custom OAuth authorization grant method. Which is performing the sign challenge.

By design, OAuth 2.0 is made to handle the user management and the authorization of those data. Therefore a third party cannot edit or view any data, for which it has no access rights. Moreover, the user management provider learns no additional information about the data the third parties store. Those properties are ideal for our use case, where the carpooling platform is the third party and the Auth Server is the user management provider. Moreover, a login token is only valid for a certain time, which means, that the access token cannot be used for tracking as long as the token itself does not grant permission to retrieve information, which would identify the user [Har12].

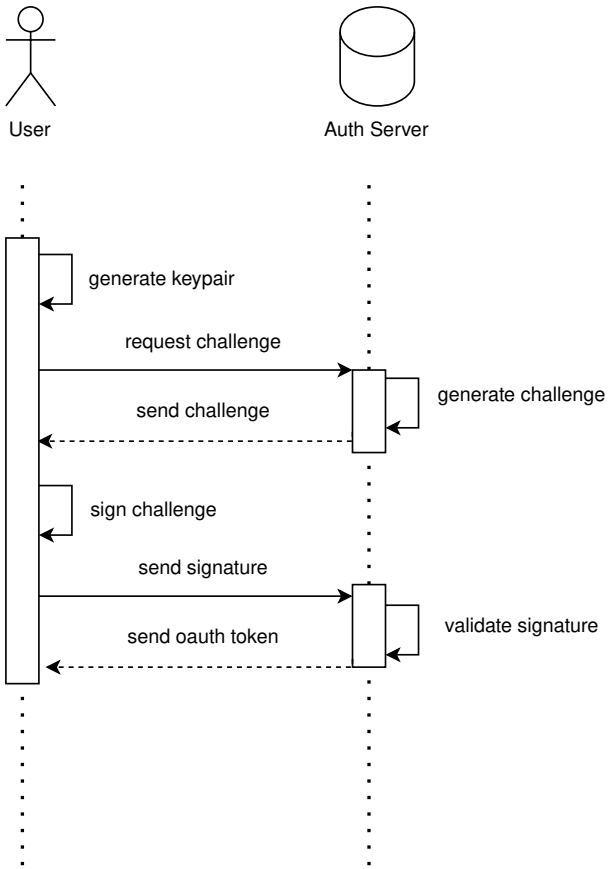

Figure 4.6: Authentication Setup as UML sequence diagram

This token then can be used by the user to generate subtokens. Those subtokens are called pseudonyms and can be used by the matching service of the carpooling platform. The user can generate as many pseudonyms as they want as long as they have an OAuth access token. The OAuth token, however, should not be given to the carpooling platform directly, unless the scope is restricted to generate pseudonyms only. Otherwise, the carpooling platform could take over the user's account as long as the token is valid.

Before a user can generate a pseudonym at least one Blockchain wallet needs to be registered and linked with the account. This linking is performed by a signing challenge, while the private wallet key is used to sign a nonce. The server is validating the challenge with the corresponding wallet address respectively the public key and if it is valid, the public key is stored by the Auth Service and can later on be used to generate a pseudonym for the linked account. This linking process is required to ensure that a pseudonym is linked to an existing user and no fake pseudonyms can be added to the public Blockchain. More details about the format and generation process are presented in section 4.5.

For generating a pseudonym the user's client is sending a pseudonym generation request for a particular public key, meaning it can only be used for transactions signed by a particular private key. If the public key is assigned to the same user as the OAuth token, a subtoken is generated. The private key of the account and of the wallet are not needed for this process. The OAuth token has a

validity of 2 hours by default. However, Auth Services can decide by themself for how long an OAuth token shall remain valid. It is recommended to keep the lifetime of the token short in order to avoid someone doing any harm with a stolen OAuth token.

4.5 Reputation

Based on the fact that all users can use the services without revealing any personal information, it is not possible to check the quality of a car or the reliability of a person on any criteria except an anonymised rating system.

This system needs to be able to provide customers with the required ride quality and the vehicle provider with the certainty that a customer is not doing any damage to their property or harm to any other customers. Which altogether makes the rating system a critical component of the whole service.

We define a rating as a number between 1 and 5, while 1 is the worst rating a user can have, 5 is the highest rating possible. A car provider can reject a user if they have a rating below a specified limit. This limit can be chosen by the provider. The user on the other hand can also specify a rating, which allows to filter vehicles with a particular rating. Because there exists a minimum rating, all users and cars must start with a rating, which makes it possible to match the other party's requirements.

One possible way would be to start with a rating of 3. This, however, would make it impossible for new cars to get any new customers. Therefore, we give every new user respectively car a rating of 5. This rating is overridden after the first actual rating is given by another customer or the car provider. Which means that the initial rating does not have a lasting impact.

Another problem is the aging of ratings. In case that user has gotten many good ratings in the past, they could behave badly after a certain time, which would not decrease their rating anymore. The same would be the case if a user got a few bad ratings in the past, they would never be able to increase their rating over time. All in all, this means, that older ratings must have a lower value than newer ratings.

Moreover, it must be guaranteed that a user cannot manipulate their rating, by whitewashing their account rating. To avoid such attacks, it must not be possible to delete and reopen an account with the same ID card. This is archived by requiring a one-time identification at registration time.

After ensuring that the ratings are plausible, we need to store them in an anonym way, so that it is not possible to gain additional information about the user. Moreover, it should not be possible to detect the user's account after taking a ride and giving a rating for this ride. But the technical requirements highly depend on the used reputation formula and the mathematical operations, which are required for the computation.

Because we do not want, past actions to have a too large impact on the current rating, we need to add an aging factor to the ratings. This has also consequences on the data, we need to store regarding the rating. Further, for archiving an accurate rating, we need to store the date when the rating was given. A sequential rating, where each rating has an index, would work, but it would reduce the signification of the total rating. This is because a person who takes more rides could change their ratings and their chronological impact faster, which would create an imbalance between users. Storing the actual time and making the time have an impact on the total rating, would require a

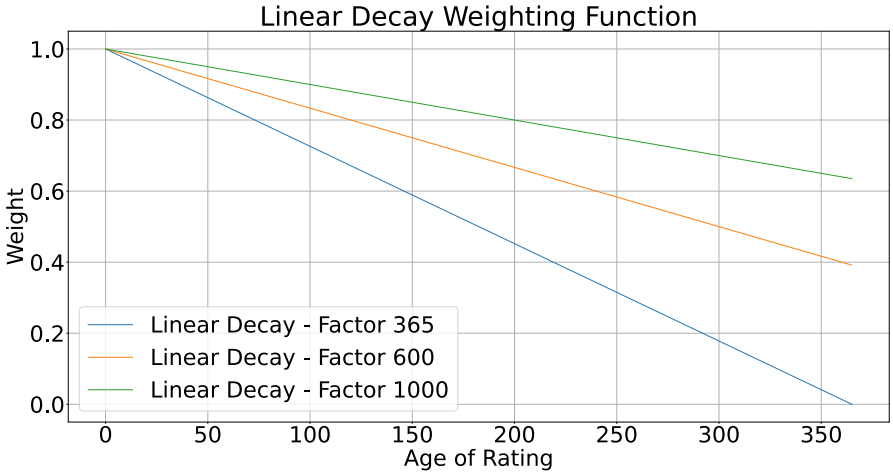

Figure 4.7: Linear Function for 365 days time frame

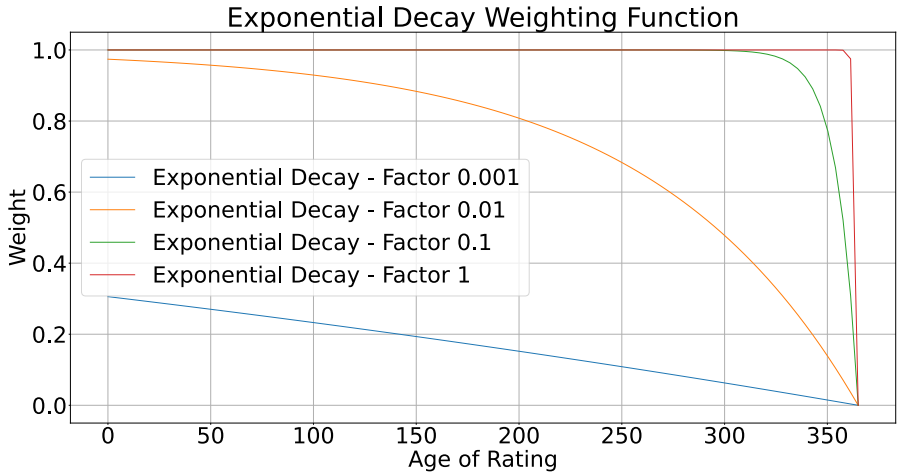

Figure 4.8: Exponential Function for 365 days time frame

recomputation of the rating after each new rating, otherwise, the aging factor would have no impact. Sequential ratings would not have that problem, because the aging is linear and depends largely on the current index and previous indexed are therefore known if the first index is 0 or 1.

Using a time-based rating system would allow us to apply either a linear or an exponential decay of the rating. A linear decay would have the problem, that fresh ratings age too fast or old ratings too slow. Meaning, that the distinction between one year or two years has less of an impact than a rating difference of one or two weeks. Plus, it would take a long time, until a past rating loses its impact on the current rating. By using an exponential decay, it is possible to make ratings age slowly within the first few months and ratings older than a few months losing their value rapidly. The Graphs in Figure 4.7 and 4.8 show how weighting functions would be distributed over a duration of 365 days. Those are using different parameters.

This leads us to the following formula:

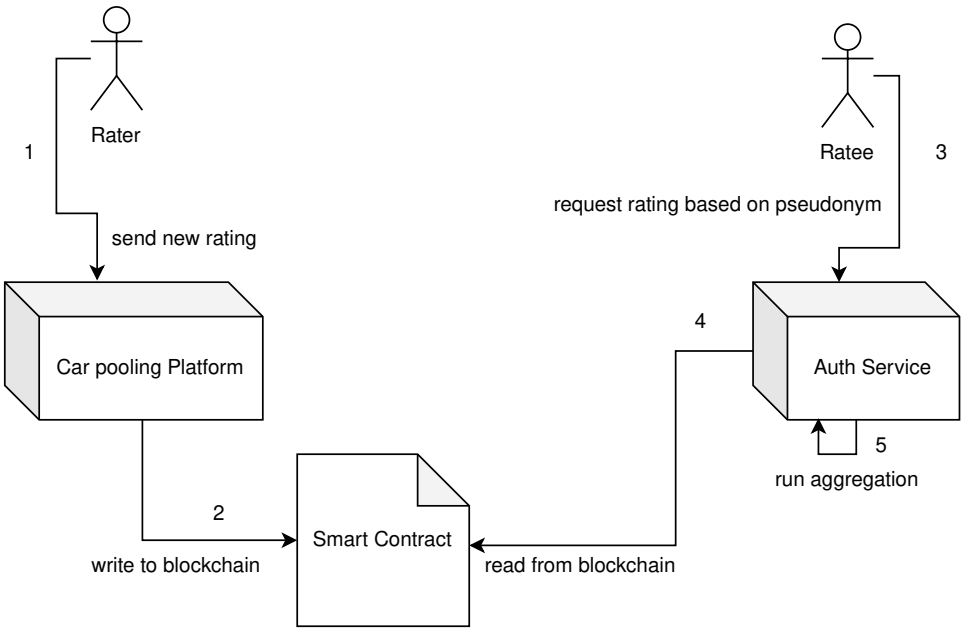

Figure 4.9: Rating - Blockchain Interaction

$$R = \frac{\Sigma_{i=1}^{n} w_i(t) \cdot r_i}{\Sigma_{i=1}^{n} w_i(t)}$$

With:

$$\Delta t_i = T - t_i$$
$$w_i(t) = e^{-\lambda \Delta t_i}$$

The given formula uses UNIX UTC timestamps, which are then converted into days instead of microseconds and rounded to the next integer to hide the exact time. The λ parameter is set to 0.01. This creates the best distribution, comparable to Figure 4.8.

Now, after we defined the formula for the calculation of the ratings and the range in which a rating can be, we need to inspect, how we can store the ratings so that it is not possible to track back a rating to the rater or rate.

One possible way would be to use homomorphic encryption. By applying homomorphic encryption it is possible to know who has given a rating, and who was rated, but it is not possible to tell which rating the person has been giving. Any external party could compute the rating, but only the party with the private key could decrypt the rating. However, the problem with homomorphic encryption is that those algorithms are complex to implement and quite slow. Plus, if a large amount of ratings are aggregated, the rating loses accuracy. Furthermore, there must exist a central party, who manages the key. Those private keys can not be managed by the user itself, since the user could simply manipulate the rating by returning a wrong value. This would require an additional trusted party, which checks if the user is returning the correct rating or which decrypts the values on behalf of the user.

Another approach is to mask the ratee and the rater and keep the ratings visible. This is possible by using pseudonyms. Those pseudonyms are managed by the Auth Service. Because the Auth Service must be a trusted party if it wants to identify people, we can be sure that it would return the correct rating and not manipulate it. Moreover, the Auth Service has only access to the user's pseudonyms, which it manages itself. If the user decides the change the Auth Service, the Auth Service loses access to newer ratings. Because the ratings have an aging property, the old ratings, which the Auth Service knows would lose their value quite rapidly. This would make it useless for a malicious Auth Service to store the old ratings it once had access to.

It is important to note that the Auth Service is only responsible for the aggregation because it has no access to the actual riding information, and therefore it could not guarantee that the ride has actually been taken by the rater and ratee. The possible interaction is illustrated in Figure 4.9. Further, it requires that the carpooling platform makes sure that no invalid ratings are inserted.

The last open question is how a pseudonym is generated and what data it contains. The most important property of a pseudonym is that it is unique and readable to all parties. Therefore, the pseudonym must be encoded in base16. The pseudonym length must be at least 24 bytes, to avoid duplicates based on a too-small value space. The Auth Service is responsible for uniqueness. This would allow them to generate roughly 256^{12} pseudonyms without risking a coalition. Respectively, the chance would be lower than 1.27e-29. The simplest way is to generate the bytes randomly. However, because of an increasing number of users and the generation of multiple pseudonyms per user per week, it is recommended that the pseudonym's length be much longer. One way would be to generate a random string of 100 bytes and append the current timestamp. Afterwards, hash the value with the Sha3 512 algorithm and encode it in base16. Code Sample 4.2 gives an example implementation.

```
1  def generate_pseudonym() do
2    base = :crypto.strong_rand_bytes(100)
3    time_based = base <> Integer.to_string(:os.system_time(:millisecond))
4    :crypto.hash(:sha3_512, time_based) |> Base.encode16()
5  end
```

Code Sample 4.2: Generate Pseudonym written in Elixir

Now after generating a pseudonym, the Auth Service needs to prove that it was the party, which has generated the token. Without this proof, the carpooling platform could not detect if the pseudonym is valid or was generated by a party, which is not authorised to generate platform pseudonyms. Further, a malicious party could generate random pseudonyms and book or rate a ride with it.

To avoid this, the Auth Service is signing the pseudonym with its private key and additional data. The corresponding public key is accessible by the carpooling service. For Blockchain operations, the public keys would get hardcoded and updated if required. This is required because smart contracts are not capable of executing an HTTP request. However, the contract owner could call a method, to update the public key to validate signatures.

As shown in Code Sample 4.3, the signature contains the pseudonym itself, the timestamp when the pseudonym was generated. These timestamps can be used to limit the time, in which a pseudonym is valid. Moreover, the pseudonym is linked to a specific wallet address. This ensured that a pseudonym could not be stolen and misused by a malicious attacker.

```
{
  "hash_method": "sha3-512",
  "hash_compute": "pseudonym+auth_server+timestamp",
  "auth_server": "prototype auth service",
  "pseudonym": "d79d8c9d7b52faed3a309f55351a01d9...",
  "timestamp": 1694699013,
  "wallet": "0xabfa3ca...",
  "signature": "231cacc12..."
}
```

Code Sample 4.3: Pseudonym Data

5 Implementation

Because of the complexity of the authentication system and rating system, we use a substantial range of various technologies. Therefore, we take a closer look into those different technologies now.

5.1 Backend

The backend of the application is split into two components, as shown in Figure 5.1. The first one is the main logic including the external REST API of the Auth Server written in the Elixir programming language. This programming language uses the Erlang VM, which allows running the logic easily in a scalable, fault-tolerant and distributed way. Elixir uses a functional programming style and allows us to write complex code quite efficiently without a lot of boilerplate code, which would be required when written i.e., in Java [Tea23].

Another big advantage is the ability to generate code that can run in tausend of different processes at the same time [FCZ16]. This is quite important when a lot of pseudonyms must be created and managed. Because the carpooling platform relies heavily on a rating system, a lot of rating requests are sent to process at the same time and need to be handled concurrently. Another handy feature of Elixir and Erlang is the capability to connect multiple nodes to a united cluster, which allows one to scale up the system quickly.

One downside of Elixir is the lack of support for interacting with Blockchains via external libraries. Most libraries for interacting with such Blockchains are typically written in JavaScript based on the nature that Blockchains such as Ethereum are usually connected with the user's browser and

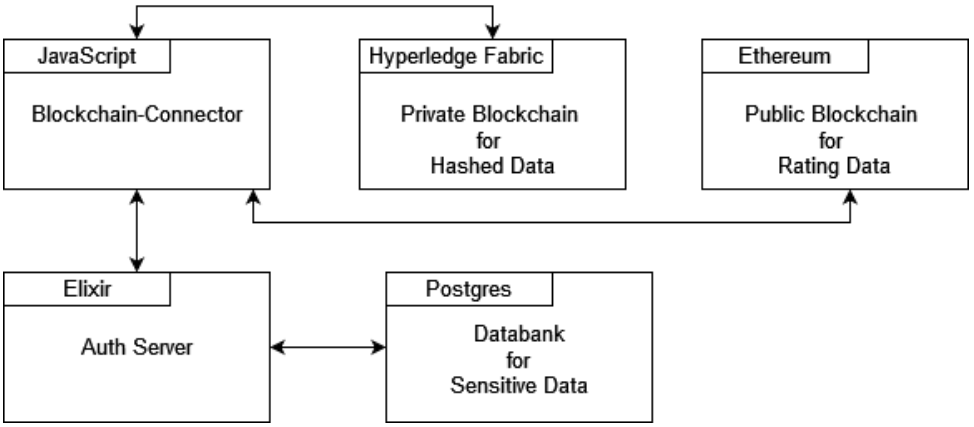

Figure 5.1: Technology overview

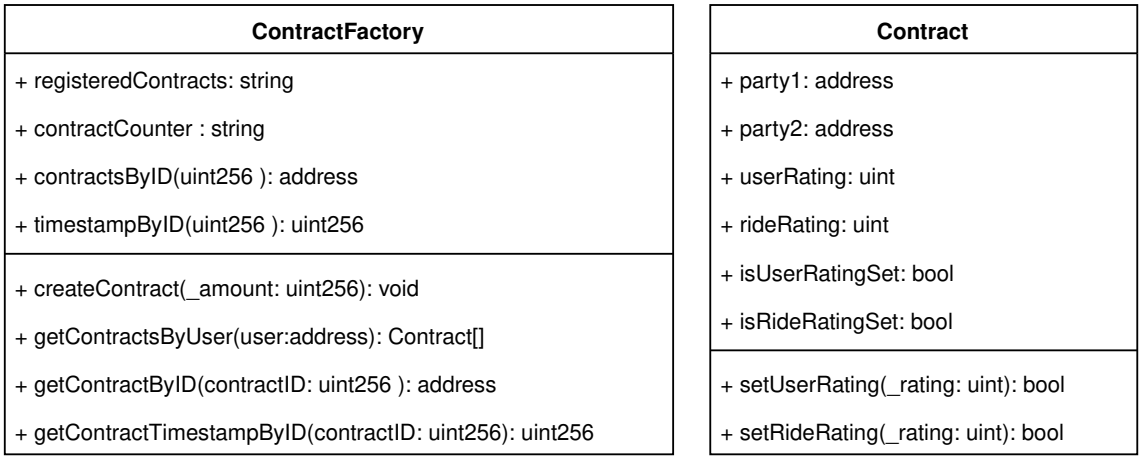

Figure 5.2: UML Class Diagram of Public Contracts

Wallet Extensions such as Metamask[1]. Moreover, we do not want to be tightened to one particular Blockchain technology, therefore it is simpler to use a JavaScript Bridge Project in order to interact with the Blockchain from the Auth Server business logic.

5.2 Blockchain

At the moment there exists a large range of different Blockchains. For our purpose, we need to choose two different Blockchains, which need to fulfil two different tasks and therefore require different technological features. Firstly, we need a Blockchain, which provides us with the transparency properties, which we require for an open platform.

This means we need to search for a public Blockchain, which needs to be able to run smart contract-based operations and ideally, it has the capability to send push notifications on changes. In practice, this first Blockchain is the main Blockchain for all carpooling application-specified operations, which only operates on anonymised data, without any capability to link any inserted data to a user. Moreover, the chosen Blockchain must be well known, because the users of the platform interact with the Blockchain directly, i.e. via their browser. Another important aspect of a carpooling platform is the capability to run the application in a scalable and distributed fashion.

Blockchains that would fit such a purpose are Ethereum and Solana. Solana is the newer of both and highly optimised for scalability and performance, which has a huge advantage over Ethereum in practice and that is the price per transaction. Which would make Solana the preferred choice in a real-world scenario. However, in our situation, in which we are only interested in a prototype, Ethereum is the better option. This is because both Blockchains offer quite similar features but Ethereum is the more mature platform and therefore has more development tools, which makes it easier to develop a prototype, such as Ganache[2] and Remix[3], which enables to deployment

[1] https://metamask.io/
[2] https://trufflesuite.com/ganache/
[3] https://remix.ethereum.org

and testing of smart contracts locally. Furthermore, we have assumed in the introduction that there already exist smart contracts in the Ethereum Blockchain, which can provide us with ratings, therefore it is handy to operate on the identical public blockchain.

Ganache is used to create an Ethereum network locally and initialise wallets for test purposes with enough credit, to perform transactions on that network. It is easy to install and to setup and therefore avoids a lot of trouble when you just want to test whether the contracts would work in a production network.

Remix on the other hand is a webbrowser tool, which enables us to interact with an Ethereum-based Blockchain, including local ones. It keeps track of the created contracts within the current session so it is possible to interact with all public methods and attributes of a contract without the need to set up any local environment.

Figure 5.2 is pointing out the used smart contracts on the public Blockchain. The upper half of the UML class diagram represents the properties of the contract, while the lower part are the operations, which can be performed on a contract. The contractFactory is responsible for generating new contracts and keeping track of existing contracts. Each car-pooling platform would own their contractFactory and be the only party who would have write permissions, while everyone has read permissions.

Each contract has an incrementing ID number, which allows retrieving the address and creation time of the contract in the time complexity class of O(1). This means, the caller would not need to iterate over all contracts to retrieve the lasted contracts. The last ID is stored in the contractCounter property. Furthermore, this combination enables Auth Services to retrieve all contracts cache them locally and update them once every hour by just retrieving new contracts and storing the last ID, they have fetched. This is important for computing the ratings later on, which are stored in the 'RideContract' smart contract. The 'rideRating' is given by the car (party1) and the 'userRating' by the user (party2).

In addition to data that shall be publically available, we need to store sensitive data which must be only visible to authorized parties. Such data would be hashes of the real identity of a person, which would need to be verified in case of registering with another Auth Service provider or if we need to verify data for fulfilling laws. The main requirements for this Blockchain are that it is private and has an authorization mechanism. One of the most established private and authorized Blockchain platforms is Hyperledge Fabric. This Blockchain allows to authorize only certified parties to join the ledger. Those certificates are based on a PKI [Hyp23b].

This limits the number of parties that can deploy smart contracts, add data to a contract and even read data is limited. In addition to the authorization mechanism, it provides a fault tolerance on multiple levels. It can provide a simple technical fault-tolerant, which requires that new data is verified by multiple nodes before any change is performed to the Blockchain or even a byzantine fault tolerance if required, which would protect against a malicious node [SBV18].

Figure 5.3 shows the used smart contracts on the private Blockchain. This contract has the responsibility to store hash values of the identification properties, such as the full name, date of birth, and place of birth. The hash value is computed by the Auth Service, which owns the data. The actual data is never stored on the ledger directly nor in the logs. Further, this requires that the Auth Service computes the hash correctly. For the computation, the sha3-512 algorithm must

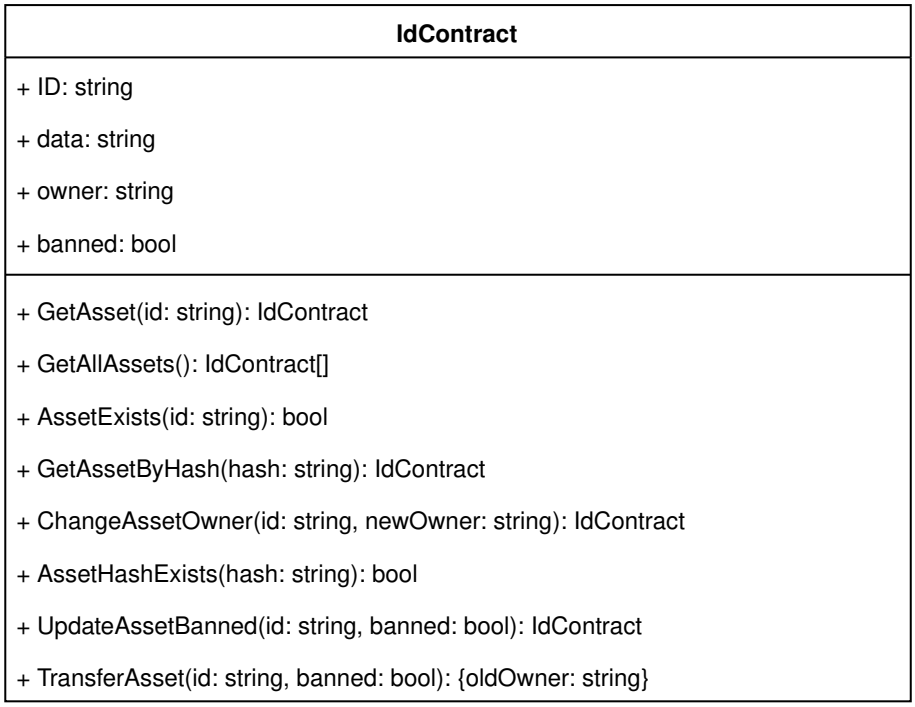

Figure 5.3: UML Class Diagram of Private Contract

be applied to the values retrieved by the eID service provider. For consistency, the value must be preformatted in JSON with alphabetically ordered keys and the hash value encoded in base16. In Elixir, the hashing algorithm implementation would look as shown in Code Sample 5.1.

Data cannot be changed once added to the Blockchain. Even though Hyperledge allows overriding contracts if all parties agree, which would allow for change in the logic regarding the data [Hyp23a].

```
1  def encode_personal_data_to_json(key_map) when is_map(key_map) do
2      key_map
3      |> Map.to_list()
4      |> Enum.sort_by(fn {key, _value} -> key end)
5      |> Jason.OrderedObject.new()
6      |> Jason.encode!()
7  end
8
9  def hash_personal_data(data) do
10     value = encode_personal_data_to_json(data)
11
12     :sha3_512
13     |> :crypto.hash(value)
14     |> Base.encode16()
15 end
```

Code Sample 5.1: Hash ID Data

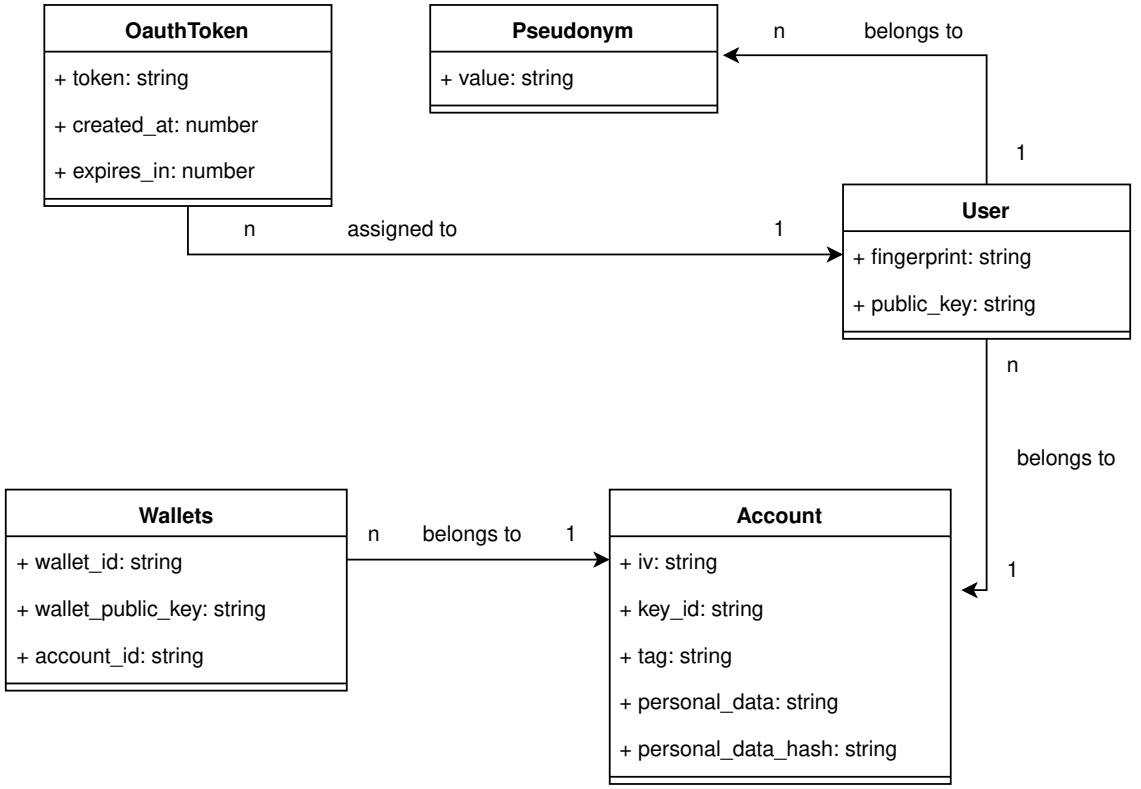

Figure 5.4: UML Class Diagramo of Database

5.3 Database

Since some data are still too sensitive to store on a Blockchain where data is immutable, even with limited access, we need to fall back to a centralized database for these data.

The main data that we need to store in the database are the login tokens, pseudonyms, wallet addresses and lastly the actual data stored on the ID cards. Because of the relatively small amount of data a single Auth Server stores, most NoSQL Databases would be large over-architecture. Therefore traditional SQL Databases fit our purpose just fine. In case more data would need to be stored by a single Auth Server, a Key-Value database, such as Riak, could be used. In the end, each Auth Service provider can choose the database that would fit their purpose and amount of data best. For our prototype, we use Postgres, which is widely supported.

In Figure 5.7, we can see the defined database tables and their relations as a UML diagram. Its central table is the account table, which defines what is assigned to an identified person. Each account can have multiple users. Those users are basically a public-private key pair, which is used to sign in. The private key is stored by the user and the public key and the corresponding public key fingerprint is stored in the database. The 'OauthToken' holds all active authentication tokens, which are issued on a successful sign-in attempt and expire after 120 minutes. In addition to the shown properties, that table also contains the scope for which a token is valid and by which application ID it was created. However, those properties are not used yet but belong to the OAuth 2.0

definition [Har12]. The pseudonyms table contains all generated pseudonyms, which are assigned to a particular user. Lastly, we have a table, which is responsible for keeping track of all wallets which belong to an account and are linked to a pseudonym.

5.4 App

HTTP Verb	Endpoint (Response Body)	Usage
POST	/api/auth/login/session (LoginSessionResponse)	Requests a session token from the backend. This token needs to be signed by the private key of the user.
POST	/api/auth/login/ (LoginResponse)	Sends the signed login requests to the server, which validates the signature and re<turns an OAuth 2.0 token if successful.
POST	/api/auth/accounts/request (CreateAccountRefResponse)	Requests a new account token. This token needs to be sent to the eID service such as Ausweis2App or BankID and is needed to map the response of those apps to the user who wants to register.
GET	/api/auth/id-reference/:reference (IdReferenceResponse)	Fetches the metadata to an id reference, such as account id if registration was successful
POST	/api/auth/accounts/create/:id_token (CreateAccountResponse)	This endpoint is called either by the eID server directly or through a proxy and contains the actual user data linked to the new account token. The token must exist otherwise the request is rejected. It is also important to validate the actual data, usually, the data contains a signature created by the eID server.
POST	/api/auth/users (UserResponse)	This endpoint allows it to add the first private-public key to the account. As long as no key is registered this endpoint can be called. Otherwise, an OAuth 2.0 token must be provided in the header and the /api/users endpoint must be called. This endpoint is usually called as soon as possible after a new account is created.
GET	/api/auth/rating/:address (RatingResponse)	Gets the rating of a particular user by own of its wallet addresses.
POST	/api/accounts/wallet/add (WalletResponse)	Adds a wallet address to an account after validating the signature
POST	/api/accounts/wallet/challenge (CreateWalletChallengeResponse)	Requests a challenge, which must be signed by the private wallet key
GET	/api/accounts/wallets (WalletResponse)	Gets all wallets of an account registered at the current Auth Service.
POST	/api/pseudonym (PseudonymResponse)	Generate a new pseudonym, this can be used for one-time actions, while the wallet remains the same for longer periods of time.
POST	/api/users (UserResponse)	Similar to /api/auth/users, however, the user requires a valid OAuth 2.0 token, for adding a new private key.

Table 5.1: Rest API Endpoints

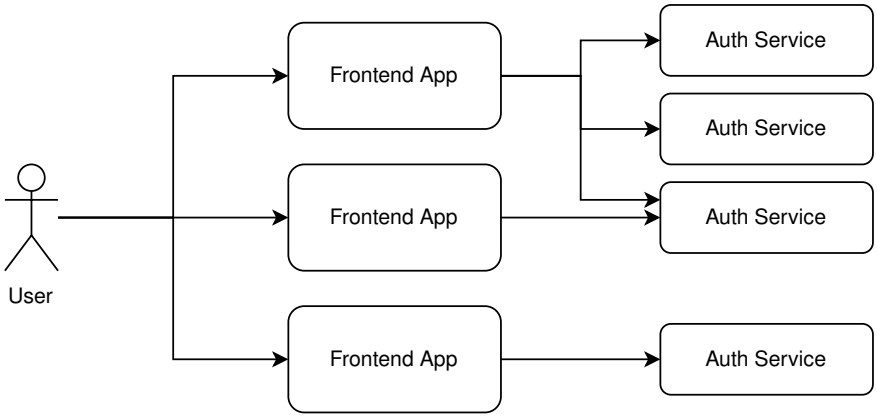

Figure 5.5: Frontend - Auth service relation

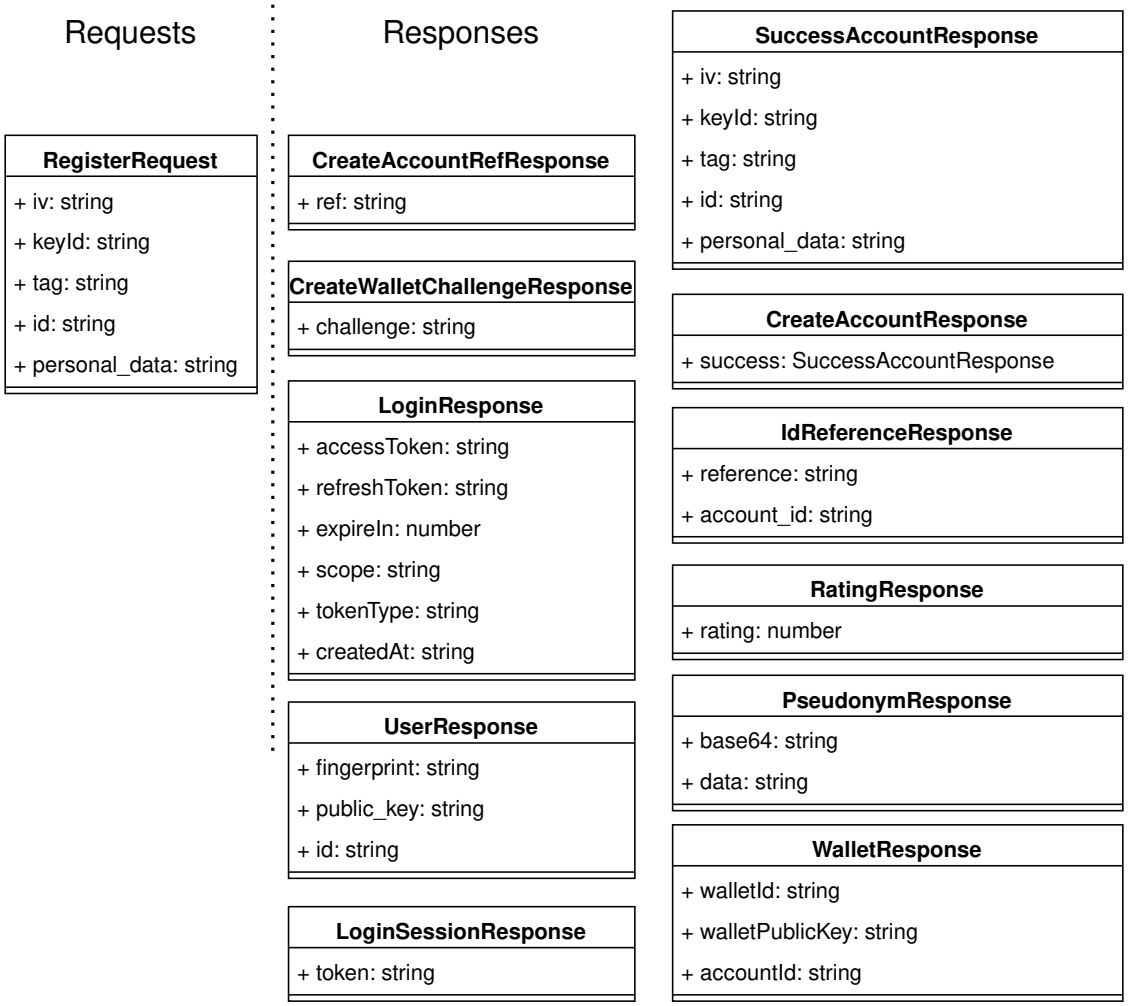

Figure 5.6: Response and Request Bodies

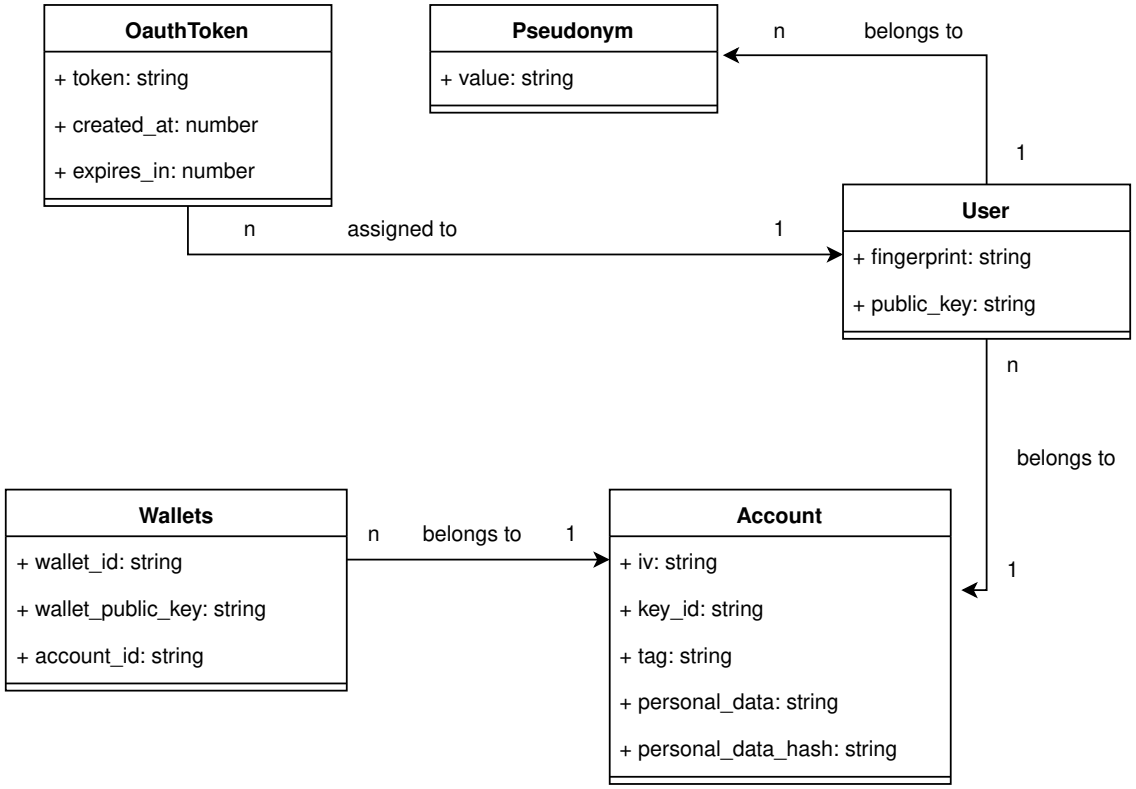

Figure 5.7: UML Class Diagram of Database

In addition to the backend, we need to implement a way for the end user to interact with the Auth Service itself. In a real-world scenario, the front-end is up to the Auth Service provider and therefore multiple front-end instances would exist, which is illustrated in Figure 5.5. The properties of the response and request bodies of the endpoints are given in Figure 5.6. This in particular means, that we only provide a reference implementation for the prototype. Additionally, all Auth Services, that implement the reference REST API Endpoints, can be added to the reference application. This would allow the user to switch the Auth Service in a single front-end application.

The 'personal_data' string in the 'RegisterRequest' is in JSON format and must contain the properties 'first_name', 'last_name', 'city', and 'date_of_birth'. All values must be encoded as strings. The dates should have the format 'DD/MM/YYYY' or 'YYYY/MM/DD'. Moreover, the city can contain additional information. Such as a district name or postcode. This property is used to notify local authorities if needed.

The prototype Application uses the following endpoints, which are shown in Table 5.1. All endpoints prefixed with '/api/auth' are callable without being authorized. All endpoints without this prefix require a login token in the Authorization Header. The authorization type is BEARER and the value is the OAuth 2.0 access token.

Because the application depends largely on cryptography, and private key management and its usage context has a mobility focus, our authentication application mainly be accessible through a mobile app. This is also required because most eID provider Apps, such as AusweisApp2 or

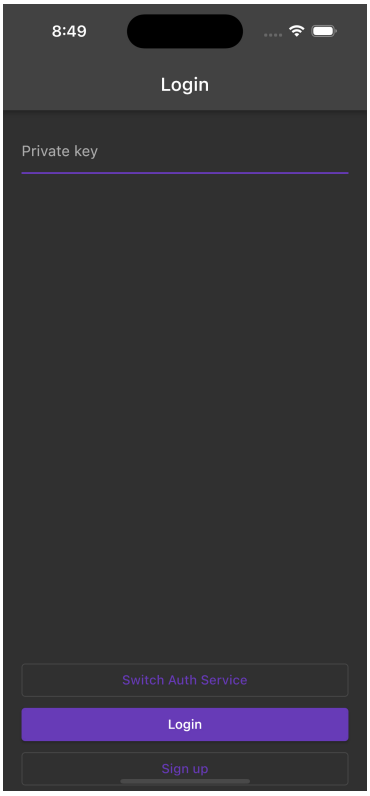

Figure 5.8: Start View

Figure 5.9: Authservice Selection

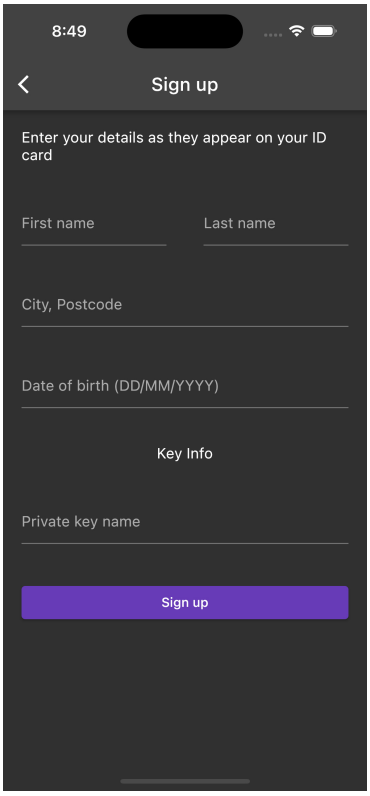

Figure 5.10: Register

BankID, only work on mobile devices or require additional hardware, such as a NFC card reader [KG23a]. To avoid prototyping for native Android and native IOS, we use the Cross-platform tool Flutter. Flutter is able to convert Dart code into semi-native code [Flu23].

The app itself provides the ability to create a new account, create an authorization token as well and generate pseudonyms, which can be used by other carpooling services. When the users have installed the app on their device and open the app, they see the login Page, shown in Figure 5.8. From here they can select the Auth Service they want to register or login with. They can select the Auth Service provider by clicking on S̈witch Auth Service.̈ Now they see a list of all available Auth Service providers, comparable to Figure 5.9. The user can choose the wanted service. Each service can use its own identification method. For now, we use the P̈rototypeÄuth Service, which uses a dummy identification service.

This Auth Service provider is only available in a development environment and allows one to register a user without an actual identification method. When the user now clicks on S̈ign Up̈they see the form, presented in Figure 5.13. Those data are usually required and validated by an eID provider. For testing purposes, it takes all data without any validation. The reason for skipping an actual identification method for now is the regulation, which is required for actually reading the German ID cards. Including receiving test ID cards. Swedish BankID itself provides a Test Environment, but also just for dummy certificates, which are used for identification, as shown in

Figure 5.11: Home View

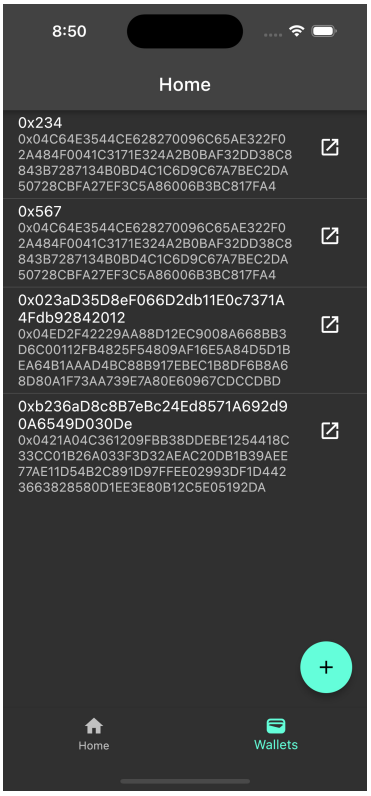

Figure 5.12: My Wallets

Figure 5.13: Carpooling App Link

Figure 4.4. Implementing BankID for the prototype would not bring us any advantage in proving the concept, it would just prove the way, how BankID works. Therefore we skip the implementation of it for now.

The private key name is an important field and defines the identifier for storing and accessing the private key, which is required when logging into the app, as shown in Figure 5.8. After entering the private key name in the login page, the user clicks loginänd lands on the main page, visible in Figure 5.11. The main page lists all carpooling apps, which accept the authentication app. A click on any item on the list copies the link to the app. This link can then be copied to a web browser that supports crypto wallets, such as MetaMask.

A click on the wallets navigation bar item takes the user to its linked accounts wallets. Those are their private and public Ethereum information.

A click on the wallets navigation bar item takes the user to its linked accounts wallets. Those are their private and public Ethereum information. A click on any of the linked wallets generates a new pseudonym and verifies that the pseudonym is related to the public key. This pseudonym is added to the clipboard, as shown in Figure 5.15 and can be used to log in to the actual carpooling app.

If the user wants to add a new wallet, they need to click on the plus icon and afterwards copy the private key and the account address in the provided fields, and example is provided in Figure 5.15. After clicking on 'Add' the public key and address are validated by the server. Therefore, the server

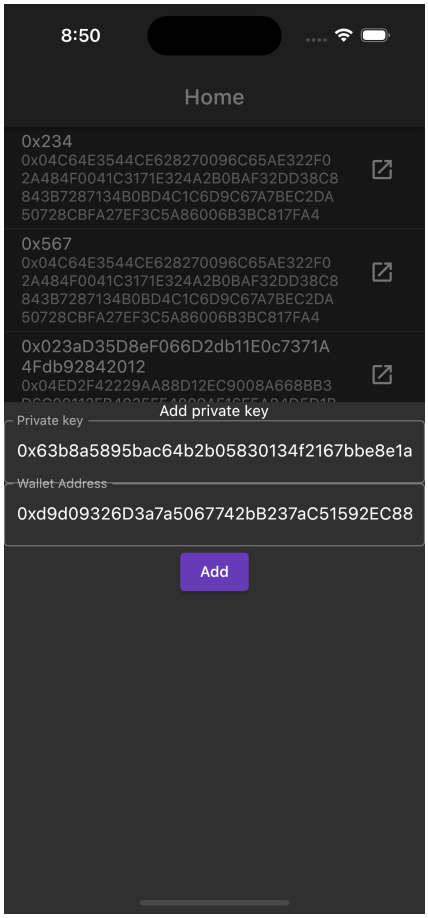

Figure 5.14: Add Wallet to Account

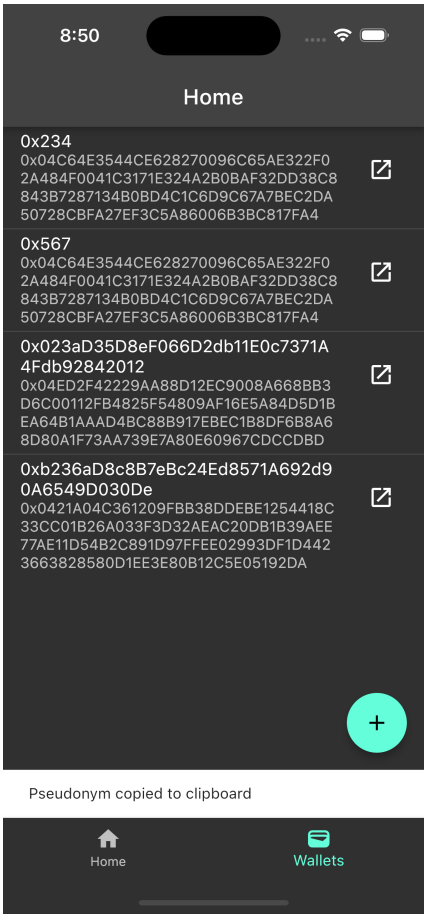

Figure 5.15: Copy Pseudonym

validates if the private and public keys match by a signing challenge, which the apps perform in the background. If it matches the public key and the address are added to the user's linked wallets and can be used to generate pseudonyms.

5.5 Cryptography

Because of the significant use of Blockchains and privacy-focused authentication, the Auth Service depends to a great extent on cryptography.

For the authentication process, we use the commonly used RSA algorithm with a 2048-bit key length. RSA is an asymmetric encryption and signature algorithm [VYGS19]. For the login operation, we make use of the RSA signature scheme. The RSA algorithm is based on the problem, that is hard to factorize large numbers. This, however, means in case that this algorithm would need to be replaced as soon as quantum computers become large and stable enough, which can break RSA by using Shor's Algorithm [DLQ+20].

Besides RSA we apply ECDSA to register an Ethereum private key with a user account. The private keys are wallet keys that are used by the Ethereum Blockchain for signing transactions. Therefore, we use those private keys and the corresponding secp256k1 curve, used by Ethereum [Afr22].

Independent of RSA and ECDSA we use the SHA3-512 Hash Algorithm for creating hash values from the data we need to sign. Sha3 is based on Keccak with small changes to the padding, specified by the NIST [SSD20]. Keccak is also used by the Ethereum Blockchain. However, based on the padding changes both hash functions return different values [Dev22a]. Table 5.2 shows the difference in the produced output by both algorithms. The value column is the exact value, which has been hashed with both algorithms.

Besides asymmetric cryptography, the Auth Server needs to encrypt sensitive data such as passport data. Those data are only stored on the databases that are owned by the Auth Server itself. Therefore symmetric encryption is the ideal choice. The most reliable ciphers are AES and ChaCha20. While AES is the most well-known and most widely implemented algorithm, Chacha20 is faster than AES if no hardware acceleration is available, which makes Chacha20 interesting for IOTs [Dev22b; DSS17].

However, since we only want to store data on a server, which usually has hardware acceleration, we apply AES-256 in GCM mode. The GCM mode is an authenticated encryption mode, which encrypts the data while guaranteeing its integrity. Moreover, the BSI is recommending only AES in their official technical guidelines [Sic23a].

In addition to cryptographic algorithms, TLS, a cryptographic protocol is used to secure the communication channel between the mobile devices and the servers. Which handles encryption on the transport layer instead of the application layer, and has its focus on protecting a connection between a client and the servers and preventing third parties from accessing any of the transported data [Sic23d].

Value	Keccak-512	SHA3-512
A value	c6c3ded0dbb480e8bb37bf04 bae7aaa31033a7ff85bcd6970 8f368338a79302f5fb7c027d 065bf849d56e21eef 9c7c239f5a7a6d5c2 96297885aa8cda855b90a	8e340b188a2392adb8 38c36deaecfdc6361169 ebcf2e64e02d9cf655c64b fb671ef974209ec7e07e70c a7b977e2716ab1934ff2 fe8804ceb9b7e1ec742fd9d0
Another value	2daa80e8320887060b 2aa3ffacefec02cbd9b1 1d2f1e3fdf24616e7108b8cd 27501666c56f0fb5283101a9 e936414ad6523f44671 a3024f8c2e3f9d5ff24803b	525dd93a5654765ed1 d96c84584730899c2a35 f38d5189d18e61c2a59c1b c1d77418ba3b8d2445b344d2f8 18c1194e8e7a59b6adf c8471d8e5504c3584996b00
The user data	ae05f24e213c1163dfd c5ffd6940cb984a05a6 9266b2aaa488c201ad0d15 c8a5dec26728023b97c7f01c76 c7c2db7363c07e2bf237 6fc68f050c8bf855737b60	c08b33132d616234b60 b8fe71ce57f427fa68b 7c6573e65f4a9d8d87ae5d0e 3be620956bcf8d7c85d9b7e8 da3abe15482e966ad4191 7d5227948295a765e22b1

Table 5.2: Keccak-512 vs. Sha3-512

6 Evaluation

Now, after we have taken a closer look into the implementation, we evaluate our concept and prototype against the predefined requirements and potential security and privacy problems.

6.1 Prototype

In addition to the theoretical concept, we have implemented a functional prototype, which enables users to register, authenticate and issue new pseudonyms, which can be used to interact with a carpooling platform. Moreover, each user can receive ratings and request a rating for another user without revealing any identity.

In detail, the prototype uses a dummy identification service, which basically always returns valid data. Therefore the mobile app is handling the signing of the data, while the server just validates the data. However, the system is designed in such a way, that the verification process is customizable for other and real eID service providers. This can easily archived by overriding the validate_signature function in the Accounts Module and adding a conversion function, for receiving the right properties. The current implementation expects a JSON format with the properties 'first_name', 'last_name', 'city', and 'date_of_birth'. In case the eID service provider, returns other fields, and data, those data must be verified before and can then be signed by the provided ID server module on the server side. The city can contain additional information such as a postcode, separated with a ','. This is handy for bigger cities such as Berlin or Munich, where people with the same name and even birthdays could be living. This city property is mainly used when contacting a legal authority is needed.

The prototype is also capable of providing a way of login only based on a private key, so no password or email addresses are used. After login, it is possible to generate as many pseudonyms as someone wants, which then can be used to book a ride at a carpooling platform. The carpooling platform itself only can use the pseudonym, which is a random token and does not provide any way to track back to a real user. Only our Auth Server is capable to collect all pseudonyms and know the real identity behind them. The Auth Server, however, do not know what rides which user has taken, therefore it only has access to the information the user is giving at the registration plus the rating values of a single user.

Lastly, it is possible to add a wallet address to an account, which is required to generate a pseudonym, to avoid, that illegal ratings getting injected and stored on a public Blockchain and damaging the reputation system. The user can add as many wallets as they want but needs to select an active wallet address in the Auth Service app. This address is then used for the pseudonym linking.

6.2 Requirements

Now we inspect whether we could fulfil all of our preset requirements for the system. Requirement 1 and 5 have their focus on the anonym usage of the user regarding a carpooling platform. Because the user only reveals their identity to the Auth Service and authenticates with that service, the carpooling platform itself does not know who exactly the user is. Furthermore, it only handles pseudonyms and wallet addresses, while the Auth Service has no access to the actual journey data. Therefore, no party can gain any information except those given voluntarily by the user.

Moreover, the described Auth Service identifies the user by their real identity. Because of this, it can notify locale authorities about incidences, without revealing the identity to the carpooling platform. The carpooling platform can notify the Auth Service about an incident and the Auth Service sends the incident information including the name and birth of a person to the local authorities. For finding the right local authority the city and postcode of the registered person are used. The country of the user does not need to be stored itself if only one ID method is used, which is restricted to a single country or nationality. The public authorities are then capable of finding all the required information such as the current contact address based on the provided information. Further, the carpooling platform gains no information about the actual identity of the involved persons and the Auth Service Platform only knows the ID information provided by the user at the time of registration, which fulfils requirement 2 as well as 3.

In order to fulfil requirement 6 we designed the system in a manner, that multiple different eID systems can be used by multiple different Auth Services. Further, hash values are stored in a distributed Blockchain ledger, which makes the system independent of a single infrastructure component. By the use of multiple different eID services, the concepted solution is easily expandable to all countries, which have at least one eID system and therefore fullfill the requirement 9.

Lastly, each party user, who has registered with any Auth Service is rateable by other users. This also includes ride provider respectively the car itself. That means we have been able to fulfil the final requirement, 4, too. By using pseudonyms, which do not give any information about the real identity, we archive that only the Auth Service, who is responsible for a user and generating pseudonyms, is able to calculate the actual rating for a person. Further, the ratee and rater do not gain any information about the given rating. By recomputing the rating just once in a regular interval, i.e. an hour, it is not possible to gain any information about the time either. Therefore, we have fulfilled the requirement 7. Moreover, it is not possible to run multiple accounts at the same time, because a hash value of the actual ID document is stored. If someone is trying to re-register, it would be denied the Auth Service or the already existing account would be used. Therefore, requirement 8 is successfully fulfilled.

6.3 Privacy and Security

Now we take a final look at the privacy of the user. If the user decides to join a carpooling platform, they need to register the Auth Service and only the Auth Service knows the real identity of the user. The carpooling platform only knows the pseudonyms and can group multiple rides by the public key of the user. This is a problem that is inherited by the Blockchain technology itself. Each user

must sign their transaction with their private key and the public key is used to verify the transaction [Wac23]. A solution to this problem is to link multiple private keys to a single account and replace the signing keys regularly.

A weakness of the system would be if we would allow a user to register with multiple services at the same time. This would allow the user to bypass any ban or reset and whitewash their reputation. The same would be the case if the user could delete any linked wallet address. This would be a risk for other users. Further, this results in a high level of trust, we need to have in the Auth Services. A solution to this problem would be to allow users to distrust other Auth Services, and so do not share a ride with people of those mistrusted Auth Services. But this is up to the implementation of the carpooling platforms, on which our authentication service has no influence.

Because a user can only register once and only with one Auth Service at the same time, there would be the risk that an Auth Service could ban a user by their own intention and the user would have no chance to switch the Auth Service provider if the same authentication method is used, i.e. if both use BankID. However, we can solve it by limiting the time a user can be registered with a single Auth Service as well as limiting the time a ban can last.

It is also important to note, that the security of the used identification method takes a large part on the security of the system as a whole. If the identification method is returning wrong information or inconsistent information, an Auth Service can not guarantee that a user is actually unique and trustworthy. Further, this implies that solely well-established ID methods should be used by an Auth Service.

Besides the required level of trust towards the Auth Service provider, the security depends largely on the applied encryption algorithms. AES is widely analysed and there are still no serious and efficient attacks known on this block cipher. However, RSA and ECC are proven to be insecure against Quantum attacks, which means, that the cryptography would need to be adapted. The problem here is that we depend on Blockchains for archiving the shown distributed approach, and we have no impact on the applied algorithms in Blockchains. But, since it is a general problem, there is already quite some research on this topic for preparing Blockchains for the post quantum[1] computing time [GCC+18]. Furthermore, the NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) is standardising the first post-quantum algorithms, which could then be adapted in this system [AAA+20].

Since we would have plenty of centralized Auth Services, which have relatively much power, there would be the need for one or multiple independent authorities, which issue certificates for joining the private Blockchain. Moreover, the authority must be able to deactivate the hash on the Blockchain, so that the user can re-register with another platform. In such a case the user would lose their reputation because the blocked Auth Service would not transfer the data voluntarily to another account. But because the initial rating is the maximal possible value, such an action would not harm the user. Moreover, ratings would lose their value over time anyway.

A related problem would be that the hash of the user ID on the private Blockchain has been in the possession of the blocked Auth Service, which means that the malicious Auth Service could publish those ID hashes. Which could create a potential security risk. Another malicious party would be able to find out, whether a user has been registered with any Auth Service if that party knows the ID

[1] time when a quantum computer can break classic cryptography

properties, such as the Name and date of birth. However, more than the information that a user is able to use a carpooling platform, can not be extracted. No pseudonyms are linked directly to any of those hashes. Only if the malicious Auth Service publishes their pseudonym table, the privacy of it's users would be at risk. This would be the case as well if an attacker is leaking the private database of an Auth Service. A benevolent Auth Service could encrypt all the pseudonyms in the database and decrypt it when needed or cache the decrypted value in RAM memory. In all cases, a malicious Auth Service could always store more information than it is allowed after the Blockchain certificate is granted. This is not preventable in a system with centralized components.

Another problem would arise if Self Sovereign Identities are used for user management, without storing any ID data in a centralized or decentralized database. In that case, a user could take back the permission grant, and our Auth Server could not report the identity to authorities. This means that Self Sovereign Identities would provide the highest possible way of privacy but have the trade-off that the platform security is decreased.

7 Conclusion and Outlook

After evaluating our concept and prototype, we can now give a conclusion and broach an outlook for further work regarding an authentication system for a privacy-preserving carpooling platform.

7.1 Conclusion

In the fast-living world, we encounter new technologies almost daily. With increasing research on self-driving cars and other vehicles, we encounter new possibilities for a better-connected world and vastly improved public transportation. Especially car sharing is used by more and more people, even though it needs at least one person, who is willing to drive the car and has the legal and physical requirements to do so. By removing this component, we may see a large improvement for people in more rural areas. This influences older and younger people alike, who struggle with a lack of public transportation.

Because this type of transportation would allow us to keep track of the movement of people, we need a way to anonymise and impersonalize sensitive information. We have defined requirements for handling the problems and have shown with a concept and implemented prototype, that we were able to fulfill those requirements and provide a way to use a carpooling platform in a private way.

For solving the problem, we have mainly focussed on technologies with regard to identification and authentication methods, which already exist and could be used to implement a prototype and even extend the prototype, so that it would be possible to run it in a test or limited production environment. Furthermore, we have identified future problems with the system, regarding security and privacy and hint at possible solutions to those problems.

At this stage of development, we have implemented the minimal foundation for an anonymous user management tool. This includes the private and public smart contracts as well as the semi-central server-side implementation.

However, there are still some elements that are not fully implemented. So, we write and read into a SQL database for avoiding to setup a reliable private Blockchain instance in a production environment. This is not an issue by itself the concept, but more a deployment issue, which would take more time to test properly in a production environment. Nevertheless, it is possible to run everything in the test environment, including the private blockchain and its smart contracts.

Additionally, we mock the eID Server responses at the moment with a custom response. In reality, each Auth Server would need to implement the interfaces of the eID service, which they want to use. To simplify this process for potential eID providers, the project would need to be extended by sample eID services such as BankID or the German electronic ID card. Or in general the most common eID provider for each country.

Furthermore, the current implementation has not focussed on efficiency. Meaning, that smart contracts, which run on the public should be optimised before being deployed in an actual environment. Otherwise, the operators of the Auth Service would pay higher fees as actually required. Also, proper error handling, which would be required for a production environment, is not yet fully implemented.

Because we only evaluate the concept for a single service, a public ban of a user is not yet implemented on the server side, while the smart contracts would allow it, to set a user and its ID hash as banned. Since this feature would only be noticeable when a user is changing the Auth Service, we skipped the implementation for now, especially because it is only updating a property of an asset on the Blockchain. It is similar to the migration of the wallets from one Auth Service to another. This would be required to be able to keep the user's rating consistent.

Another important addition is the validation of wallet addresses. This validation is dependent on the used public Blockchain, and its used hash algorithm for deriving the address from the public key. For Ethereum, the Keccak Hash Algorithm is required. This could either be implemented via the 'JavaScript Blockchain Connector' or the 'Elixir Auth Server' component, which are shown in Figure 5.1.

Moreover, some features for better usability would need to be implemented. This would be a direct implementation of Metamask into the app. At the moment all links must be copied manually into the Metamask browser. And the wallets need to be copied manually from Metamask into the app. A direct connection with Metamask would enable the authentication app to interact with Metamask in the background, resulting in a reduced burden on the user. Moreover, it must be possible to add a new private key to the account if the user loses their private keys, i.e. if the phone gets lost or stolen.

All in all, we can summarize that the problem of an anonymised car-pooling platform is possible, but at the current state of technology, there are still components that would be centralized and therefore be trusted by other parties.

7.2 Outlook

Because of the early stage of Self Sovereign Identities, the shown concept focussed on centralized identification and user management. Yet, with the further progress and adoption of Self Sovereign Identities, the concept would need to be further expanded to handle problems with decentralized user management. So that a user could not just remove the permission to obtain its data, once committed an illegal action.

This work took a theoretical approach, further this means that no actual user feedback is taken into consideration. However, the user's opinion and the whole system as a united are required to actually measure the level of trust the users give the system.

Lastly, the Auth Services, conceived in this work, are relatively powerful. The registration allows those services to read the ID properties of a user, and store those. This means that the certification process of the Auth Services is important and would need to be further investigated. In particular, an independent authority needs to be found or founded.

Bibliography

[AAA+20] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, et al. "Status report on the second round of the NIST post-quantum cryptography standardization process". In: *US Department of Commerce, NIST* 2 (2020) (cit. on p. 69).

[ADA23] ADAC. "EU-Führerschein-Reform: Müssen Senioren bald zum Fahrtauglichkeits-Check?" In: (2023). URL: https://www.adac.de/news/rentner-fahrtauglichkeit-fuehrerschein/ (cit. on p. 17).

[Afr22] Afri. Feb. 2022. URL: https://dev.to/q9/finally-understanding-ethereum-accounts-1kpe (cit. on p. 65).

[AG23a] D. L. AG. *Self-Sovereign Identity (SSI): Autonomous Identity Management*. 2023. URL: https://www.dock.io/post/self-sovereign-identity#ssi-pillar-3-verifiable-credentials-vcs (cit. on pp. 37, 38).

[AG23b] intersoft consulting services AG. *GDPR Personal Data*. 2023. URL: https://gdpr-info.eu/issues/personal-data/ (cit. on p. 32).

[Bal17] Y. Balaj. "Token-based vs session-based authentication: A survey". In: *no. September* (2017), pp. 1–6 (cit. on p. 35).

[Ban23] BankID. *Bank ID in numbers*. 2023. URL: https://www.bankid.com/en/om-oss/statistik (cit. on p. 32).

[BBB+22] A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, V. Zucca. *OpenFHE: Open-Source Fully Homomorphic Encryption Library*. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915. 2022. URL: https://eprint.iacr.org/2022/915 (cit. on p. 24).

[BDW17] A. Bhawiyuga, M. Data, A. Warda. "Architectural design of token based authentication of MQTT protocol in constrained IoT device". In: *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*. IEEE. 2017, pp. 1–4 (cit. on p. 35).

[Bon+99] D. Boneh et al. "Twenty years of attacks on the RSA cryptosystem". In: *Notices of the AMS* 46.2 (1999), pp. 203–213 (cit. on p. 22).

[Cha83] D. Chaum. "Blind signatures for untraceable payments". In: *Advances in Cryptology: Proceedings of Crypto 82*. Springer. 1983, pp. 199–203 (cit. on p. 39).

[Clo23] I. Cloudflare. "What is Transport Layer Security (TLS)?" In: (2023). URL: https://www.cloudflare.com/en-gb/learning/ssl/transport-layer-security-tls/ (cit. on pp. 25, 26).

[Com23a] E. Commission. *Commission welcomes provisional political agreement on EU Digital Identity Wallet, Europe's first trusted and secure digital identity app*. 2023. URL: https://ec.europa.eu/commission/presscorner/detail/en/ip_23_3556 (cit. on p. 38).

[Com23b] E. Commission. *EU Digital identity: 4 projects launched to test EUDI Wallet*. 2023. URL: https://digital-strategy.ec.europa.eu/en/news/eu-digital-identity-4-projects-launched-test-eudi-wallet (cit. on p. 34).

[con23a] M. contributors. "An overview of HTTP". In: (2023). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (cit. on p. 26).

[con23b] M. contributors. "HTTP caching". In: (2023). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching (cit. on p. 26).

[con23c] M. contributors. "HTTP Messages". In: (2023). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages (cit. on p. 26).

[cor23] corwintines. *Proof-of-stake (PoS)*. 2023. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (cit. on pp. 29, 30).

[cyb22] E. U. A. for cybersecurity. *Leveraging the Self-Sovereign Identity (SSI) Concept to Build Trust*. 2022. URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Conformant+wallets (cit. on pp. 37, 38).

[D-T23a] D-Trust. *Sichere Identifizierung - eID-Verfahren im Überblick*. 2023. URL: https://www.d-trust.net/de/loesungen/identifizierungsdienste (cit. on p. 33).

[D-T23b] D-Trust. *Support AusweisIDent, eID-Service und Berechtigungszertifikate*. 2023. URL: https://www.d-trust.net/de/support/ausweisident-und-eid-service (cit. on p. 33).

[Dev22a] P. C. for Developers. 2022. URL: https://cryptobook.nakov.com/cryptographic-hash-functions/secure-hash-algorithms (cit. on pp. 24, 65).

[Dev22b] P. C. for Developers. *Popular Symmetric Algorithms*. 2022. URL: https://cryptobook.nakov.com/symmetric-key-ciphers/popular-symmetric-algorithms (cit. on p. 65).

[DLQ+20] Z.-C. Duan, J.-P. Li, J. Qin, Y. Yu, Y.-H. Huo, S. Höfling, C.-Y. Lu, N.-L. Liu, K. Chen, J.-W. Pan. "Proof-of-principle demonstration of compiled Shor’s algorithm using a quantum dot single-photon source". In: *Opt. Express* 28.13 (June 2020), pp. 18917–18930. DOI: 10.1364/OE.390209. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-28-13-18917 (cit. on p. 65).

[DS17] S. Dey, S. Sarkar. "Improved analysis for reduced round Salsa and Chacha". In: *Discrete Applied Mathematics* 227 (2017), pp. 58–69. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2017.04.034. URL: https://www.sciencedirect.com/science/article/pii/S0166218X1730224X (cit. on p. 23).

[DSS17] F. De Santis, A. Schauer, G. Sigl. "ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 2017, pp. 692–697. DOI: 10.23919/DATE.2017.7927078 (cit. on pp. 23, 65).

[Dwo07] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. 2007. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf (cit. on p. 23).

[Dwo15] M. Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. en. 2015-08-04 2015. DOI: https://doi.org/10.6028/NIST.FIPS.202 (cit. on p. 24).

[eth23] ethereum.org. "ZERO-KNOWLEDGE PROOFS". In: (2023). URL: https://ethereum.org/en/zero-knowledge-proofs/ (cit. on p. 25).

[FC23] B. Federal Ministry of the Interior, Community. *Data on the ID Card*. 2023. URL: https://www.personalausweisportal.de/Webs/PA/EN/citizens/german-id-card/data-on-the-id-card/data-on-the-id-card-node.html (cit. on p. 32).

[FCZ16] G. Fedrecheski, L. C. P. Costa, M. K. Zuffo. "Elixir programming language evaluation for IoT". In: *2016 IEEE International Symposium on Consumer Electronics (ISCE)*. 2016, pp. 105–106. DOI: 10.1109/ISCE.2016.7797392 (cit. on p. 53).

[Flu23] Flutter. *Writing custom platform-specific code*. 2023. URL: https://docs.flutter.dev/platform-integration/platform-channels?tab=type-mappings-java-tab (cit. on p. 62).

[GAGK21] A. Gupta, A. Anpalagan, L. Guan, A. S. Khwaja. "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues". In: *Array* 10 (2021), p. 100057 (cit. on p. 17).

[GCC+18] Y.-L. Gao, X.-B. Chen, Y.-L. Chen, Y. Sun, X.-X. Niu, Y.-X. Yang. "A Secure Cryptocurrency Scheme Based on Post-Quantum Blockchain". In: *IEEE Access* 6 (2018), pp. 27205–27213. DOI: 10.1109/ACCESS.2018.2827203 (cit. on p. 69).

[Gen09] C. Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009 (cit. on p. 24).

[Grö10] Å. Grönlund. "Electronic identity management in Sweden: governance of a market approach". In: (2010). DOI: 10.1007/s12394-010-0043-1 (cit. on pp. 31, 33).

[Gro16] T. N. Group. *TÜViT erhält Akkreditierung als eIDAS-Zertifizierer*. 2016. URL: https://www.tuev-nord-group.com/de/newsroom/news/details/article/tuevit-erhaelt-akkreditierung-als-eidas-zertifizierer/ (cit. on p. 33).

[Har12] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Oct. 2012. DOI: 10.17487/RFC6749. URL: https://www.rfc-editor.org/info/rfc6749 (cit. on pp. 35, 36, 45, 58).

[HBB22] O. Hasan, L. Brunie, E. Bertino. "Privacy-Preserving Reputation Systems Based on Blockchain and Other Cryptographic Building Blocks: A Survey". In: *ACM Comput. Surv.* 55.2 (Jan. 2022). ISSN: 0360-0300. DOI: 10.1145/3490236. URL: https://doi.org/10.1145/3490236 (cit. on pp. 18, 19, 38–40).

[HBFL18] J. Haupt, B. Bender, B. Fabian, S. Lessmann. "Robust identification of email tracking: A machine learning approach". In: *European Journal of Operational Research* 271.1 (2018), pp. 341–356. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.05.018. URL: https://www.sciencedirect.com/science/article/pii/S0377221718304120 (cit. on p. 19).

[HDPB19] E. W. Huff, N. DellaMaria, B. Posadas, J. Brinkley. "Am I Too Old to Drive? Opinions of Older Adults on Self-Driving Vehicles". In: *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '19. Pittsburgh, PA, USA: Association for Computing Machinery, 2019, pp. 500–509. ISBN: 9781450366762. DOI: 10.1145/3308561.3353801. URL: https://doi.org/10.1145/3308561.3353801 (cit. on p. 17).

[HNS+20] S. Hansen, K. B. Newbold, D. M. Scott, B. Vrkljan, A. Grenier. "To drive or not to drive: Driving cessation amongst older adults in rural and small towns in Canada". In: *Journal of Transport Geography* 86 (2020), p. 102773. ISSN: 0966-6923. DOI: https://doi.org/10.1016/j.jtrangeo.2020.102773. URL: https://www.sciencedirect.com/science/article/pii/S0966692319307732 (cit. on p. 17).

[HPL23] D. Hardt, A. Parecki, T. Lodderstedt. *The OAuth 2.1 Authorization Framework*. Internet-Draft draft-ietf-oauth-v2-1-09. Work in Progress. Internet Engineering Task Force, July 2023. 90 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/09/ (cit. on pp. 35, 36).

[Hyp23a] Hyperledger. "Fabric chaincode lifecycle¶". In: (2023). URL: https://hyperledger-fabric.readthedocs.io/en/release-2.5/chaincode_lifecycle.html (cit. on p. 56).

[Hyp23b] Hyperledger. "Identity". In: (2023). URL: https://hyperledger-fabric.readthedocs.io/en/release-2.5/identity/identity.html (cit. on p. 55).

[IBM23a] IBM. "What is a relational database?" In: (2023). URL: https://www.ibm.com/topics/relational-databases (cit. on pp. 27–29).

[IBM23b] IBM. *What is blockchain technology?* 2023. URL: https://www.ibm.com/topics/blockchain (cit. on pp. 29, 30).

[Inf23a] Infominer. *European Blockchain Services Infrastructure (EBSI) and the eSSIF*. 2023. URL: https://decentralized-id.com/government/europe/eu/ebsi-essif/ (cit. on p. 38).

[Inf23b] InformationNOW. "Driving as you get older". In: (2023). URL: https://www.informationnow.org.uk/article/driving-as-you-get-older/# (cit. on p. 17).

[Inn23] B. des Innern und für Heimat. "Bildmaterial zum Personalausweis". In: (2023). URL: https://www.personalausweisportal.de/SharedDocs/artikel/Webs/PA/DE/informationsmaterial/grafiken-bilder/bildmaterial.html (cit. on p. 32).

[KG23a] G. G. C. KG. *Kompatible Kartenleser*. 2023. URL: https://www.ausweisapp.bund.de/kompatible-kartenleser (cit. on p. 62).

[KG23b] G. G. C. KG. "So werden Sie Diensteanbieter". In: (2023). URL: https://www.ausweisapp.bund.de/so-werden-sie-diensteanbieter (cit. on p. 32).

[KK12] R. Kaur, A. Kaur. "Digital Signature". In: *2012 International Conference on Computing Sciences*. 2012, pp. 295–301. DOI: 10.1109/ICCS.2012.25 (cit. on p. 24).

[KM23] S. Khanum, K. Mustafa. "A systematic literature review on sensitive data protection in blockchain applications". In: *Concurrency and Computation: Practice and Experience* 35.1 (2023), e7422. DOI: https://doi.org/10.1002/cpe.7422. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.7422. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7422 (cit. on p. 36).

[KT21] E. Krysowski, J. Tremewan. “WHY DOES ANONYMITY MAKE US MISBEHAVE: DIFFERENT NORMS OR LESS COMPLIANCE?” In: *Economic Inquiry* 59.2 (2021), pp. 776–789. DOI: https://doi.org/10.1111/ecin.12955. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ecin.12955. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ecin.12955 (cit. on p. 19).

[Küs16] R. Küsters. *Oauth Analyse*. 2016. URL: https://publ.sec.uni-stuttgart.de/fettkuestersschmitz-ccs-2016.pdf (cit. on pp. 35, 36).

[Lim20] R. H. Limited. “What is a REST API?” In: (2020). URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (cit. on p. 26).

[LWX19] M. Liu, K. Wu, J. J. Xu. “How Will Blockchain Technology Impact Auditing and Accounting: Permissionless versus Permissioned Blockchain”. In: *Current Issues in Auditing* 13.2 (Aug. 2019), A19–A29. ISSN: 1936-1270. DOI: 10.2308/ciia-52540. eprint: https://publications.aaahq.org/cia/article-pdf/13/2/A19/56496/ciia-52540.pdf. URL: https://doi.org/10.2308/ciia-52540 (cit. on p. 37).

[LZW21] M. Li, Z. Zeng, Y. Wang. “An innovative car sharing technological paradigm towards sustainable mobility”. In: *Journal of Cleaner Production* 288 (2021), p. 125626. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2020.125626. URL: https://www.sciencedirect.com/science/article/pii/S0959652620356729 (cit. on p. 18).

[Mar22] C. C. C. Martin Tschirsich. *Video Ident Verfahren*. 2022. URL: https://www.ccc.de/system/uploads/329/original/Angriff_auf_Video-Ident_v1.2.pdf (cit. on p. 32).

[MGGM18] A. Mühle, A. Grüner, T. Gayvoronskaya, C. Meinel. “A survey on essential components of a self-sovereign identity”. In: *Computer Science Review* 30 (2018), pp. 80–86 (cit. on pp. 18, 34, 37).

[MGV20] J. A. Molina, J. I. Giménez-Nadal, J. Velilla. “Sustainable Commuting: Results from a Social Approach and International Evidence on Carpooling”. In: *Sustainability* 12 (2020). ISSN: 2071-1050. DOI: 10.3390/su12229587. URL: https://www.mdpi.com/2071-1050/12/22/9587 (cit. on p. 18).

[MK19] A. Meier, M. Kaufmann. *SQL & NoSQL databases*. Springer, 2019 (cit. on pp. 28, 29).

[MLKS08] G. Madlmayr, J. Langer, C. Kantner, J. Scharinger. “NFC Devices: Security and Privacy”. In: *2008 Third International Conference on Availability, Reliability and Security*. 2008, pp. 642–647. DOI: 10.1109/ARES.2008.105 (cit. on p. 27).

[NIS17] NIST. “Challenge-Response Protocol”. In: (2017). URL: https://csrc.nist.gov/glossary/term/challenge_response_protocol (cit. on p. 45).

[NIS23] NIST. *Withdrawn NIST Technical Series Publication*. 2023. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (cit. on pp. 22, 23).

[NL15] Y. Nir, A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. May 2015. DOI: 10.17487/RFC7539. URL: https://www.rfc-editor.org/info/rfc7539 (cit. on p. 23).

[Okt22] Okta. *Self-Sovereign Identity (SSI): Autonomous Identity Management*. 2022. URL: https://www.okta.com/identity-101/self-sovereign-identity/ (cit. on pp. 37, 38).

[PGH21] D. Pöhn, M. Grabatin, W. Hommel. "eID and Self-Sovereign Identity Usage: An Overview". In: *Electronics* 10.22 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10222811. URL: https://www.mdpi.com/2079-9292/10/22/2811 (cit. on p. 31).

[Pri21] B. d. I. u. f. H. PricewaterhouseCoopers GmbH. *PwC-Studie*. 2021. URL: https://www.personalausweisportal.de/SharedDocs/kurzmeldungen/Webs/PA/DE/2021/10_pwc_studie.html (cit. on p. 32).

[QPM21] C. Quirós, J. Portela, R. Marín. "Differentiated models in the collaborative transport economy: A mixture analysis for Blablacar and Uber". In: *Technology in Society* 67 (2021), p. 101727. ISSN: 0160-791X. DOI: https://doi.org/10.1016/j.techsoc.2021.101727. URL: https://www.sciencedirect.com/science/article/pii/S0160791X21002025 (cit. on p. 17).

[Rau22] I. Rauh. "Wie geht es mit dem ID-Wallet weiter?" In: (2022). URL: https://www.security-insider.de/wie-geht-es-mit-dem-id-wallet-weiter-a-8b1b1dd2a7c9765e7f229e16db0d42b7/ (cit. on p. 34).

[Reu23] M. Reuter. "Keine Strategie bei der elektronischen Identität". In: (2023). URL: https://netzpolitik.org/2023/online-ausweis-keine-strategie-bei-der-elektronischen-identitaet/ (cit. on p. 34).

[SBV18] J. Sousa, A. Bessani, M. Vukolic. "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018, pp. 51–58. DOI: 10.1109/DSN.2018.00018 (cit. on p. 55).

[SC19] S. Shaheen, A. Cohen. "Shared ride services in North America: definitions, impacts, and the future of pooling". In: *Transport Reviews* 39.4 (2019), pp. 427–442. DOI: 10.1080/01441647.2018.1497728. eprint: https://doi.org/10.1080/01441647.2018.1497728. URL: https://doi.org/10.1080/01441647.2018.1497728 (cit. on p. 17).

[SCB+18] S. Shaheen, A. Cohen, A. Bayen, et al. "The benefits of carpooling". In: (2018) (cit. on p. 17).

[Sic18] B. für Sicherheit in der Informationstechnik. *Technical Guideline BSI TR-03111*. 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_V-2-1_pdf.pdf?__blob=publicationFile&v=1 (cit. on p. 22).

[Sic20] B. für Sicherheit in der Informationstechnik. *German eID based on Extended Access Control v2*. 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/EIDAS/German_eID_Whitepaper_v1-4.pdf?__blob=publicationFile&v=2 (cit. on p. 32).

[Sic23a] B. für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=9 (cit. on p. 65).

[Sic23b] B. für Sicherheit in der Informationstechnik. *Proof-of-stake (PoS)*. 2023. URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Technologien_sicher_gestalten/Blockchain-Kryptowaehrung/blockchain-kryptowaehrung_node.html (cit. on p. 29).

[Sic23c] B. für Sicherheit in der Informationstechnik. *Qualifizierung als Vertrauensdiensteanbeiter*. 2023. URL: https://www.bsi.bund.de/DE/Themen/Oeffentliche-Verwaltung/eIDAS-Verordnung/Qualifizierung-als-Vertrauensdiensteanbieter/qualifizierung-als-vertrauensdiensteanbieter_node.html (cit. on p. 33).

[Sic23d] B. für Sicherheit in der Informationstechnik. *Technische Richtlinie TR-02102-2 Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=6 (cit. on p. 65).

[SPP22] K. Sen, G. Prybutok, V. Prybutok. "The use of digital technology for social wellbeing reduces social isolation in older adults: A systematic review". In: *SSM - Population Health* 17 (2022), p. 101020. ISSN: 2352-8273. DOI: https://doi.org/10.1016/j.ssmph.2021.101020. URL: https://www.sciencedirect.com/science/article/pii/S2352827321002950 (cit. on p. 17).

[SSD20] A. Sideris, T. Sanida, M. Dasygenis. "High Throughput Implementation of the Keccak Hash Function Using the Nios-II Processor". In: *Technologies* 8.1 (2020). ISSN: 2227-7080. DOI: 10.3390/technologies8010015. URL: https://www.mdpi.com/2227-7080/8/1/15 (cit. on p. 65).

[SSH23a] SSH. "SSH Key Management". In: (2023). URL: https://www.ssh.com/academy/iam/ssh-key-management (cit. on p. 35).

[SSH23b] SSH. "What is SSH Public Key Authentication?" In: (2023). URL: https://www.ssh.com/academy/ssh/public-key-authentication (cit. on p. 35).

[Sul13] N. Sullivan. *A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography*. 2013. URL: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/ (cit. on p. 22).

[Sul15] N. Sullivan. *Do the ChaCha: better mobile performance with cryptography*. 2015. URL: https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/ (cit. on p. 23).

[swe21] swedishimmigration.se. "How to get a Personal Identity Number (Personnummer) in Sweden". In: (2021). URL: https://www.swedishimmigration.se/all-topics/working-in-sweden/how-to-get-a-personal-identity-number-personnummer-in-sweden/ (cit. on p. 32).

[Tea23] T. E. Team. *elixir*. 2023. URL: https://elixir-lang.org/ (cit. on p. 53).

[Tha23] Thales. *eIDAS 2: the countdown to a single European Digital ID Wallet has begun*. 2023. URL: https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/identity/eidas-regulations (cit. on p. 34).

[Thu19] M. Thuret-Benoist. "What is the difference between personally identifiable information (PII) and personal data?" In: (2019). URL: https://techgdpr.com/blog/difference-between-pii-and-personal-data/ (cit. on pp. 32, 33).

[Voß21] O. Voß. "„Digitale Brieftasche" der Bundesregierung: Scheitern mit Ansage?" In: (2021). URL: https://www.tagesspiegel.de/wirtschaft/scheitern-mit-ansage-4280389.html (cit. on p. 34).

[VYGS19] Z. Vahdati, S. Yasin, A. Ghasempour, M. Salehi. “Comparison of ECC and RSA algorithms in IoT devices”. In: *Journal of Theoretical and Applied Information Technology* 97.16 (2019) (cit. on p. 65).

[Wac23] P. Wackerow. “Block explorers”. In: (2023). URL: https://ethereum.org/en/developers/docs/data-and-analytics/block-explorers/ (cit. on p. 69).

[WG18] K. Wüst, A. Gervais. “Do you Need a Blockchain?” In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. June 2018, pp. 45–54. DOI: 10.1109/CVCBT.2018.00011 (cit. on p. 37).

[WNK20] A. Wood, K. Najarian, D. Kahrobaei. “Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics”. In: *ACM Comput. Surv.* 53.4 (Aug. 2020). ISSN: 0360-0300. DOI: 10.1145/3394658. URL: https://doi.org/10.1145/3394658 (cit. on p. 38).

All links were last followed on September 28, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 03.10.2023 M. Fischer

place, date, signature

8 Attachments

8.1 Car Pooling - Ethereum Smart Contracts

We assume that contracts with the following properties and functions exist.

8.1.1 Ride Contract

This contract keeps track of the ride itself and stores the ratings of all parties, that participate in the ride.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RideContract {
    address public party1;
    address public party2;
    uint public userRating;
    uint public rideRating;

    constructor(address _party1) payable {
        party1 = _party1;
    }

    function setUserRating(uint _rating) public {
        // set the rating of party 1 by party 2
    }

    function setRideRating(uint _rating) public {
        // set the rating of party 2 by party 1
    }
}
```

Code Sample 8.1: Ethereum - Ride Contract

,

8.1.2 Ride Contract Factory

Used to create and manage the individual contracts by the car pooling platform and provides us the possible to interact with it.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./ride-contract.sol";

contract ContractFactory {

    address[] public registeredContracts;

    // Counter to keep track of the contract IDs
    uint256 public contractCounter = 0;

     // Mapping from contract ID to contract timestamp
    mapping(uint256 => uint256) public timestampByID;

    // Mapping from contract ID to contract address
    mapping(uint256 => address) public contractsByID;

    constructor() {

    }

    mapping(address => RideContract[]) public userContracts;

    function createContract(uint256 _amount) public payable {
        // creates a new contract
    }

    function getContractsByUser(address user) public view returns (Contract[] memory) {
        // returns all contracts addresses, which belongs to a user public key address
    }

    function getContractByID(uint256 contractID) public view returns (address) {
        // returns the ride contract with a particular block address
    }

    function getContractTimestampByID(uint256 contractID) public view returns (uint256) {
        // returns the time when a particular contract was created, based on its block address
    }
}
```

Code Sample 8.2: Ethereum - Ride Contract Factory

8.2 Auth Service - HyperLedge Fabric Smart Contract

The contract used by all auth service to register an ID card globally.

```
/*
 * SPDX-License-Identifier: Apache-2.0
 */

'use strict';

// Deterministic JSON.stringify()
```

```
const stringify  = require('json-stringify-deterministic');
const sortKeysRecursive  = require('sort-keys-recursive');
const { Contract } = require('fabric-contract-api');

class IdContract extends Contract {

    async InitLedger(ctx) {

    }

    // CreateAsset issues a new asset to the world state with given details.
    async CreateAsset(ctx, id, data_hash, owner) {
        const existsID = await this.AssetExists(ctx, id)
        const existHash = await this.AssetHashExists(ctx, data_hash);
        if (existsID) {
            throw new Error(`The asset ${id} already exists with ${existsID}`);
        }

        if(existHash){
            throw new Error(`The asset hash returns ${existHash} and therefore the asset already exists`)
;
        }

        const asset = {
            ID: id,
            data: data_hash,
            owner: owner,
            banned: false,
        };
        // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-
recursive'
        await ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(asset))));
        return JSON.stringify(asset);
    }

    // ReadAsset returns the asset stored in the world state with given id.
    async ReadAsset(ctx, id) {
        return (await this.GetAsset(ctx,id)).toString();
    }

    async GetAsset(ctx, id) {
        const assetJSON = await ctx.stub.getState(id); // get the asset from chaincode state
        if (!assetJSON || assetJSON.length === 0) {
            throw new Error(`The asset ${id} does not exist`);
        }
        return assetJSON;
    }

    async GetAssetByHash(ctx, hash) {
        return this.GetAllAssets(ctx).then((assets) => {
            let result = null;
            const ass = JSON.parse(assets);
            for(let i = 0; i < ass.length; i++) {
                if (ass[i].data == hash) {
                    result = ass[i];
                }
            }
```

```
            return JSON.stringify({result: result, exist: result != null});
        }
        );
    }


    async ChangeAssetOwner(ctx, id, newOwner) {
        const exists = await this.AssetExists(ctx, id);
        if (!exists) {
            throw new Error(`The asset ${id} does not exist`);
        }
        const assetString = await this.ReadAsset(ctx, id);
        const asset = JSON.parse(assetString);
        // overwriting original asset with new asset
        const updatedAsset = {
            ID: id,
            data: asset.data,
            owner: newOwner,
            banned: asset.banned,
        };
        // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-recursive'
        return ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(updatedAsset))));
    }



    async AssetHashExists(ctx, hash) {
            return this.GetAssetByHash(ctx, hash).then((asset) => {
                const data = JSON.parse(asset);
                return data.exist === "true";
            });
    }

    // returns the response as a string for parsing it by an external application
    async AssetHashExistsAsStr(ctx, hash) {
        return this.GetAssetByHash(ctx, hash).then((asset) => {
            const data = JSON.parse(asset);
            return JSON.stringify(data.exist);
        });
    }

    // UpdateAsset updates an existing asset in the world state with provided parameters.
    async UpdateAssetBanned(ctx, id, banned) {
        const exists = await this.AssetExists(ctx, id);
        if (!exists) {
            throw new Error(`The asset ${id} does not exist`);
        }
        const assetString = await this.ReadAsset(ctx, id);
        const asset = JSON.parse(assetString);

        // overwriting original asset with new asset
        const updatedAsset = {
            ID: id,
            data: asset.data,
            owner: asset.owner,
            banned: banned,
```

```
119         };
120
121         // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-
    recursive'
122         return ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(updatedAsset))));
123     }
124
125
126     // AssetExists returns true when asset with given ID exists in world state.
127     async AssetExists(ctx, id) {
128         const assetJSON = await ctx.stub.getState(id);
129         return assetJSON && assetJSON.length > 0;
130     }
131
132     // TransferAsset updates the owner field of asset with given id in the world state.
133     async TransferAsset(ctx, id, newOwner) {
134         const assetString = await this.ReadAsset(ctx, id);
135         const asset = JSON.parse(assetString);
136         const oldOwner = asset.Owner;
137         asset.Owner = newOwner;
138         // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-
    recursive'
139         await ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(asset))));
140         return JSON.stringify({oldOwner: oldOwner});
141     }
142
143     // GetAllAssets returns all assets found in the world state.
144     async GetAllAssets(ctx) {
145         const allResults = [];
146         // range query with empty string for startKey and endKey does an open-ended query of all assets
    in the chaincode namespace.
147         const iterator = await ctx.stub.getStateByRange('', '');
148         let result = await iterator.next();
149         while (!result.done) {
150             const strValue = Buffer.from(result.value.value.toString()).toString('utf8');
151             let record;
152             try {
153                 record = JSON.parse(strValue);
154             } catch (err) {
155                 console.log(err);
156                 record = strValue;
157             }
158             allResults.push(record);
159             result = await iterator.next();
160         }
161         return JSON.stringify(allResults);
162     }
163 }
164
165 module.exports = IdContract;
```

Code Sample 8.3: Auth Service - HyperLedge Fabric Smart Contract