Open Specification of a user-controlled Web Service for Personal Data

Master's Thesis
International Media and Computing

Author:

Gregor Jahn
gj@lucendio.com

HTW Berlin
University of Applied Sciences

Faculty 4: School of Computing,
Communication and Business

Supervisor:

Prof. Dr. Burkhard Messer
burkhard.messer@htw-berlin.de

Reviewer:

Prof. Dr.-Ing. Carsten Busch
carsten.busch@HTW-Berlin.de

March 2017

Abstract

Often data are referred to as *the oil of the 21st century*. But recovering petroleum or coal is governed by rules and legislation, whereas vast amounts of personal data are harvested from a variety of resources with no working restrictions or asking for permission. Well air-conditioned data centers are mining these data around the clock via dozens of CPUs, eager to discover even the tiniest correlations worth interpreting in enormous haystacks that are filled with data belonging to millions of individuals who have no knowledge of those computations. But these very computations, and thus their developers, almost inevitably, and without restraints, discriminate against these so called *data subjects*. Such practices cannot be accepted because these haystacks contain actual identities of human beings, including their personalities, but are nonetheless being recklessly monetized. To address this issue and reduce the possibility of discrimination, third parties have to be supplied with only a minimum of data. That is, data owners must be in charge of deciding which entity has access to what personal data of theirs. This project aims to specify a personal data service that empowers its user to regain full control over her personal data. It facilitates the regulation of data flows and provides detailed information on where the data goes so that a *data subject* can become her own data broker. In order to trust such a service, the user must be able to look inside and see for herself if it behaves in unexpected ways. Therefore the specification and all implementations are being open sourced and developed transparently in public, which also encourages self-hosting.

Acknowledgement

A quiet thought in the beginning emerged as something that felt right to do, but was still missing a clear outline. It took a while for both of us to understand what this work was about and where it would lead. Despite this, he gave me the space to write about this rather unconventional topic. A great thanks to my Supervisor for having all these constructive discussions and for not getting tired of making me rethink my thoughts.

I challenged myself to write this thesis not in my native language. What at first might sound insane turned out to be a huge learning process, but I would have never be able to come this far on my own. Thanks to Jag, a colleague of mine, who lent me his mother tongue from time to time. Without him I would have not be able to reach this level of proficiency.

Furthermore, I want to acknowledge all the amazing work that's been done to make the internet such an extraordinary common good. Thanks to everyone for contributing and sharing their knowledge and expertise to the rest of the world.

Thanks to Bruno and Eugen, two great minds, who I am honored to call friends, who have helped to review this work, making me change perspectives and rightfully forcing me to question my approaches. And thanks to another great wizard who helped me fighting a widely known markup language to layout this document.
I'm especially thankful to the one person having my back through the whole process. Tenaciously but very kindly bearing the temporary downgrade. All things come to an end, thanks to you, Liebste.
Finally I would like to thank my parents for allowing me to get this far.

Table of Contents

List of Tables

List of Figures

This page intentionally left blank

1 Introduction

1.1 Motivation

Nowadays, it is difficult to find a business that does not collect data about something; humans are particular targets of choice for the *Big Data Movement* [1]. Since humans are all individuals, they are distinct from each other. While subsets of individuals might share a minor set of attributes, the majority is still very unique compared to an individual, given that the overall variety of attributes is complex. That small amount of similarity might seem to be of less importance, due to the nature of inflationary occurrence, but the opposite turns out to be true. These similarities allow the determination of whether or not individuals are part of a subset. Stereotypical patterns are applied to these subsets and thus to all related individuals. This enriched information is then used to help predict outcomes of problems or questions related to these individuals. This is also known as *discrimination*, which

> *[…] refers to unfair or unequal treatment of people based on membership to a category or a minority, without regard to individual merit [2, p. 1].*

Discrimination is a serious issue in our society, caused by humans interacting with each other, directly or indirectly, but also when they leverage computers and algorithms to uncover formerly unnoticed information in order to improve their decision making. For example, when qualifying for a loan, hiring employees, investigating crimes or renting flats. The decision to approve or deny is based on computed data about the individuals in question [3, Chap. 5.6], which is merely discrimination on a much larger scale and with less effort (almost parenthetically).
The described phenomenon is originally referred to as *Bias in computer systems* [4]. What at first seems like machines going rouge on humans is, in fact, the *cognitive bias* [5] of human nature, modeled into machine executable language and built to reveal the patterns their creators were looking for. The *"Inheritance of humanness"* [6, Sec. 2], so to say.

In addition to the identity-defining data mentioned before, humans have the habit to create more and more data on a daily basis, both pro-actively (e.g by writing a post) and passively (e.g by allowing the app to access their current location while submitting the post). As a result, already gigantic databases grow ever larger, waiting to be harvested, collected, aggregated, analyzed and finally interpreted. The crux here is, the more data being made available [7] for mining, the higher the chances of isolating datasets (clusters) that differ from each other but are internally coherent. By defining those datasets, instead of distinguishing on an individual level, humans are being reduced to these set-defining characteristics in order to fit in these clusters.

In order to lower potential discrimination, either the responsible parts in these machines need to be erased while simultaneously raising awareness and teaching people about this issue of discrimination, or all the personal data needs to be prevented from falling into these data silos in the first place. Although both approaches are valid and should be pursued simultaneously, the latter will be addressed in this work.

1.2 Purpose & Outcome

At first glance, it might not be considered harmful to provide one's own personal data to third parties, at least from an individual's perspective, be-

cause free or improved services are eventually offered in return. For example, more adequate recommendations and fitting advertisement, or more helpful therapies and more secure environments. Gathering and processing data is essentially just mathematics and computer technologies. How those tools are utilized and what purposed they serve is within the decision of their developers. However, what data items are used and how they get processed should be determined by the data creators. Thereby allowing them an influence on the results of these processes and thus on decisions made upon them that impact their lives.

To address the described issue, the initial idea here is to (1) equip individuals with the ability to control and maintain their entire personal data distribution, in order to (2) reduce the amount of *potentially discriminatory* [2] attributes that could leak into arbitrary computations. For that, people need a reliable and trustworthy tool, which helps them to manage all their personal data and provides an interface for third parties to access their data, but on their own terms. The parties that would be responsible for such a tool would likely have the most accurate and reliable one-stop resource to an individuals' personal data at hand, while simultaneously being urged to respect their privacy. This approach comes along also with some downsides related to security and potential data loss. Elaborating on these issues and discussing potential solutions is part of the design process *(Chapter 5)*.

This way of addressing the described dilemma about personal data analysis is not new *(Chapter 2 - Related Work)*. Early work done in this field can be dated back to the Mid-2000s when studies were conducted, for example, about recent developments in the industry (e.g targeted ads) and the user's concerns about privacy [8]. At that time, the term *Vendor Relationship Management (VRM)* was first used within the context of user-centric personal data management, which then also led into the *ProjectVRM* [9]. A great amount of effort has gone into this area of research since then. In parallel, commercial products and business models try to solve some of the problems related to this. For instance, with concepts like the *Personal Data Store (PDS)* [10] or an implementation of the *MyData* concept [11] called *Meeco* [12], which are all be covered in detail in the following chapter.

The research work done for this thesis constitutes the foundation for an

Open Specification, which is a manual for implementing a concept called *Data Reservoir Flow Control*, henceforth called *ReFlowd*. Examining important topics like the architecture, where data can be stored, how to obtain data from the exposed API or what requirements have to be met by a user interface for personal data management, is part of this work. By the time this document has been submitted, most of the core issues should have already been addressed and can thus be outlined in a first draft of the specification. Only then can the task of actually implementing certain components begin. The reason for that is, when sensitive subjects, especially things like people's privacy, are at risk, all aspects in question deserve careful consideration, so they can be addressed properly. That is, adequate effort must be put primarily into the theoretical work. However, that does not mean writing code to test theories and ideas isn't allowed during the development and specification process. It is encouraged and might even help to spot flaws or perhaps trigger improvements of the specification.

To create confidence in this project and the software built upon it, it's vital to make all development processes fully transparent and encourage people to get involved. For this reason all related software and documents [13] are open source from day one.

In summary, this document lays the ground work and is intended to be the initial step in a development process fabricating a tool to manage personal data. The tool is controlled and administrated by the individual to whom the personal data belongs. It enables her to get a more precise understanding of what data is accessed by whom and how this might affect her privacy.

1.3 Terminologies

Web Service: A service, accessible by electronic devices over the internet, which would otherwise be out of reach or unavailable (from the current location). Interactions with a service usually happen through enriched websites or other web-compatible applications and interfaces.

Open Specification: A specification is a formal and very detailed way of describing a technology, its internals, and behaviour from external per-

spectives. It provides guidance for possible implementations to ensure a minimum viable level of interoperability. Structured in a formalized document and widely adopted, it might become a *technical standard*. *Open* in this case means that its accessible for everyone without restrictions. When it comes to the intellectual value itself, that might be handled differently, for example with an enclosed license.

Profile Data: A collection of data items reflecting an individual's inherent information and other basic predominantly static data items (no sets), which, in conjunction, uniquely relate to that individual.

Digital Footprint: Refers to data that is related to an individual. It is distinguished between an active footprint, which involves data and information about an individual who chose to share them publicly, and a passive footprint, which includes all data about an individual that is collected by third parties without the individual's knowledge.

Data Subject: An individual who first and foremost is the owner of all of her personal data; sometimes referred to as (data) owner.

Data Reservoir Flow Control (ReFlowd): A *web service* that is controlled, owned, and maybe even hosted by an individual. It provides access to the *data subject's* personal data and offers permission management as well as maintainability for those data. It can be seen as her personal agent; sometimes also referred to as the system.

Operator: A *data subject* that uses *ReFlowd* to control (and probably host) her personal data; sometimes referred to as (data) controller or (data) owner.

(Data) Consumer: Third party who requests permission or is already allowed to access parts of the *operator's* personal data through *ReFlowd*; sometimes referred to as (data) collector or vendor.

Data Broker: Third party with commercial interests in collecting, aggregating and analyzing information about humans from any possible resource in order to combine and enrich that data, to finally license those corpora to other organisations [14].

Permission Request: A formalized attempt made by a third party to re-

quest permissions in order to access certain data items on *ReFlowd*. The request has to include all the data items to which access is being demanded, as well as sufficient information about its underlying purpose. It requires the third party to already be registered as *data consumer*.

Permission Profile: A set of access rules and configuration tied to a *data consumer*. It determines what data is accessible by the related *data consumer* and for how long. The profile is the result of a reviewed and granted *permission request*.

Access Request: An attempt to actually access data provided by *ReFlowd*. The request primarily consists of a query that defines what data items are tried to be accessed. The access is only permitted if the query matches against the *permission profiles*.

Endpoint: An endpoint is defined as part of a URI that is uniquely associated with a single *data consumer*. In this case, it's the part of a URI referring to the host (domain incl. subdomains), whereas subsequent parts indicate different resources that might be available within that endpoint. It can also be viewed as group of resources whose access is restricted.

1.4 Scenarios

The following use cases portray different situations and possible ways in which the tool in question might be applicable, and shows in several ways that it can be helpful to be in charge of its own personal data. Some are more practical and realistic, like ordering and purchasing a product on the internet, while others might at the moment not seem to be very useful, but show a certain potential to become more relevant when new technologies and business models will occur, that are followed by new desires for data.

NOTICE: The subsequent phrasing is intentionally written in colloquial language in order to facilitate comprehensibility and to convey a visualizable idea of what it would look like to use such technology in everyday situations.

Ordering a product on the internet

The data subject searches through the web to find a new toaster because her old one recently broke. After some clicks and reviews, she finds her soon-to-become latest member of the household's kitchenware. After putting the model name in a price search engine, hoping to save some money, the first entry, offering a 23% discount, catches her attention. She decides to have a deeper look into the toasters, so she heads towards the original web shop entry. Finally she puts the item into her shopping cart, despite the fact that she never bought something from that online shop before. Then she proceeds to checkout so that she can place her order. The shop-interface asks her to either insert her credentials, proceed without registration or sign-in, or allow the shop to obtain all required data on its own by either scanning a QR-Code displayed below or insert a URL to her *Personal Data as a Service*. She opens up the management panel of her *ReFlowd* in a new browser window and authenticates herself to the system. Afterwards she creates a new entry in a list of *data consumers* who already get permitted to access certain characteristics of her personal data. As a result, she gets prompted with a URL, which she inserts as the shop interface requests her to do, only after she has convinced herself that the data exchange with the shop is based on a secure connection (HTTPS). Moving on to the next step after submitting the URL, the data subject is asked to decide how she would like to pay. The choices are: credit card, invoice, online payment or bank transfer. She chooses the last one, submits her selection and thereby completes the order process. She goes back to the kitchen. After some time, a push notification appears on her mobile device. The notification is about a *permission request* which has just arrived at her *ReFlowd*, asking her to grant permissions to the shop-system, where she earlier placed the order. The shop wants to access her full name, address and email, which are required to proceed with the order. Based on the information given in the request, she creates a new *permission profile* for the shop. Additionally, for the profile she can decide between three states of how long the permission is going to last: *one time only*, *expires on date* and *granted, until further notice*. Since she has never ordered at this shop before and probably won't do it again, she decides to grant access only for this specific occasion. The shop-system is then notified of the decision result. If the result is positive - which is the case here - the

data can be obtained and the order can be further processed. As a result, the data subject receives an email, containing information regarding her order, including the shop owner's bank details, which enables her to pay the due amount. After the shop-system receives the payment, the toaster is shipped.

In order to get a full impression of how the whole process would look like if the data subject would have chosen another payment method, the differences are described below. If the data subject had wanted to pay with her credit card, the shop-system would have asked to also access her credit card as well as its associated secret. When sending the email, the system would have omitted the information about the shop's bank details. Paying with invoice, would have been possible only if the *ReFlowd* initially had been able to provide certified profile data, which necessarily would have been rated trustworthy. That again would have reduced the risk taken by shop owner and would have enabled her to take action in cases of fraud or misuse. Choosing to involve an online payment service provider as a middleman for processing the payment would have required the data subject to have granted the provider access to her *ReFlowd* upfront. In that case, the shop-system would have asked for her payment provider account identifier, so that the system could have requested the payment directly from that payment service provider. This would have caused the service provider to consult the *ReFlowd*, which would have resulted in a second notification asking the data subject for permission to proceed. After a successful payment transfer, the shipment would have been initiated.

Interacting with a social network

The first entry to a social network requires either a URL to the data subject's *ReFlowd*, which has uniquely been generated for that purpose, or a QR-Code provided by the social network. The data subject receives a notification on her mobile device send from her *ReFlowd*, revealing what data that network wants to access and maybe even why. If her mobile device is currently not at hand, she can also use the management panel provided by her *ReFlowd*, which is accessible with a web browser on every internet-enabled device. Within that panel pending permission reviews are indicated. Regardless of whether the data subject has already reviewed the request, she should still be able to login. After doing so, she would see all her information, unless

she has not yet granted permissions to the social network to access her data *until-further-notice*. If this is done, after waiting a moment and then reloading the browser session, all her data should then show up. So every time someone on that network tries to access her information, with whom she has allowed to see that information, managed by each user within the network, the network pulls the required data from her *ReFlowd* as long as it is permitted to do so. It is also conceivable that the social network does provide a back-channel to the *ReFlowd*, so that all content she creates within that network, including all interactions with other users, can be stored in her *ReFlowd*, allowing it to be provided to other *data consumers*. The network itself only stores references to all these content objects. Whether it is an image, a post, or comment on somebody else's post, the actual content to be displayed is fetched from the corresponding *ReFlowd*.

Applying for a loan and checking creditworthiness

The data subject would like to buy an apartment. In order to finance the acquisition, she needs funding which, in her case, is based on a loan. During a conversation in a credit institute of her choice, an account consultant describes to her what data is required in order to decide her creditworthiness. While nodding consensually, she takes out her smartphone and brings up the management panel of her *ReFlowd*. The consultant flips his screen showing a QR-Code and the *data subject* scans it. The tool displays some information about the institute including a reference to this assignment and a list of all data points the institute would like to access in order to calculate her score, such as address, monthly income, relationship status and family, history of banking or other current loans. After some back and forth and solving some misunderstandings with the help of her consultant, she decides to allow only partial access to the requested data, and only for this time and purpose. The consultant kindly points out, that these decisions might impact her score and thus on the loan and its terms. While handling some more formalities and talking about other possible products she might be interested in, the consultant gets notified by his computer, confirming the access permission. Thereupon the two finish their meeting and the consultant informs the data subject of the next steps, which include a note about contacting her within the next few days, when the institute has come to a conclusion. In the case

of a positive outcome, a new appointment for handling all the paperwork and signing the contract needs to be made. From a technical point of view, two different ways of computing the score are imaginable. The first would be to just transfer the plain data including expiry date and information about how reliable the data is. However, the actual computations and analytics to obtain the score happen within the infrastructure of the credit institute. When this process has finished, all the personal data that have been transferred must be erased. An possible alternative that would prevent the data from leaving the *ReFlowd*, could be that the institute's request doesn't contain a data query, but instead coming along with some software and information on how to run it. The *ReFlowd* server will provide an isolated runtime in which the software can be executed. After that process is finished, the result is sent back to the credit institute's infrastructure.

Maintain and provide its own medical record

Some time ago on a hiking trip, in a moment of carelessness the data subject accidently broke her leg. She went to a hospital and straight into surgery, where the surgeon where able to fix the injury. Time went by and the leg healed completely. When she woke up today, she felt pain coming from the area where her leg was broken. She decided to call in sick and went straight to a doctor nearby. During her recovery she had been visiting that doctor regularly. At the reception desk, she opens up the *ReFlowd*'s management panel on her smartphone and searches through the list of data consumers. After she finds the entry for this clinic, she flips her phone to show the receptionist the corresponding QR-Code, which the receptionist scans immediately. However the receptionist was not able to see any data on the screen, because the access has already expired. The data subject only permitted access for the estimated time of recovery, which is already over. That's why she now gets a notification asking to re-grant some access. While going through the data items the clinic-system has requested, she notices that her address is incorrect. Last month she moved into a bigger apartment across the street. She must have forgotten to change that data. She immediately corrects the address, right before saving the *permission profile* for the clinic-system. She also includes access to all the data originating from the time after her accident. A moment later the receptionist confirms now being able

to see all necessary data. The data subject takes a seat in the waiting room. While passing time, she decides to take a deeper look into her list of data consumers. Some of them she couldn't even remember and for others she was surprised by which data items she had granted access to. She starts to restrict certain permissions, as she desires. She even removes some of the entries entirely. The appointment with her doctor went great. The doctor even had to review the x-ray images in order to make an adequate differential diagnosis. After the visit, she makes a quick stop at a pharmacy along the way to pickup the drugs her doctor had prescribed to reduce the pain. She has to wait in the queue since two other customers are in front of her. She realizes that it's the first time she's in this store. So she prepares a new entry in her list of data consumers, including all information about her prescriptions. By the time she gets served, she simply lets the person behind the register scan her QR-code. In the next seconds the data subject gets a notification about a *permission request* from this store, which she quickly reviews and confirms by making sure that the permissions in that profile are the ones she prepared minutes ago. A moment later the pharmacist comes back with her drugs, which she then pays in cash. The transaction is done and the data subject leaves the store.

Vehicle data and mobility

Let's assume a car has no hardware on board to establish a wireless wide area connection to an outside access node. One is only able to connect to the car from the inside (wired or wireless). After entering a car, on the data subject's mobile device a notification comes up from the car asking for permission to connect that mobile device. In addition to the expiration date, the data subject is provided with two additional options which she can enable or disable. First, a wifi network provided to everyone in the car can be enabled, which utilizes the uplink from the mobile device to the internet. Secondly, the car is allowed to use the uplink to open up connections so it can emit or receive data from the internet. As a result the device owner gains full control over any data the car might want to transfer. And this again would allow two things: (A) permission management for all outgoing data and (B) funnel all data generated and provided by the car towards the *ReFlowd* that is associated with the linked device. It might be feasible as

well to deny certain connections the car tries to open. Data will then be stored only in the *ReFlowd*. If a third party is interested in that data they have to ask for access permission. That same concept of movement tracking and vehicle data aggregation could be applied to driving motorcycles and bicycles as well.

2

Fundamentals

The following chapter shall provide basic knowledge of concepts like *Personal Identity*, *Big Data* and *Data Ownership*. It also explains what *Personal Data* is from a legal standpoint and covers some of the issues caused by different legislation colliding through the one global internet. This again requires an elaboration on how *Personal data* impacts our society as well as the economy. Overall, this chapter is meant to facilitate a common understanding of the stated issue and why it could be addressed as described later on. Furthermore, there is a summary of what research has already been done its current state. Finally, it gives a brief overview of standards and technologies that might be utilized for this project.

2.1 Digital Identity, Personal Data and Ownership

A **Digital Identity** is viewed as a non-physical abstraction of an entity, such as an organisation, an individual, a device or even some software. It is bidirectionally associated to its physical counterpart. In the context of this work, it only refers to human beings. Therefore a Digital Identity is

the representation of an individual in digital systems, consisting of identity-defining data, such as *personal information*, its own history and preferences [15]. *Personal information*, in this case, refers to inherent (e.g. date of birth) as well as imposed (e.g. credit card number) characteristics. The individual to whom those data relates to, is the owner of that *Digital Identity*. From a technical standpoint, a *Digital Identity* is essentially a collection of characteristics, attributes and time series data (e.g. interaction logs or bank transfer history). Based on a subset of those attribute values, a unique fingerprint can be easily generated. Depending on the data item and complexity of its value, such a fingerprint could require either a single, unique identifier on its own (e.g. social security number) or only a few. Hence, it doesn't take a complete set of attributes including all its values, but rather just a fraction of a *Digital Identity* in order to determine its rightful owner and physical counterpart. The *Digital Identity* can be viewed as an avatar in digital environments or even as the digital part of a persons's identity. That is, a *Digital Identity* of a living individual cannot simply be reduced to bits and bytes. Instead, it should be valued as an appropriate, and perhaps legal, representation in certain contexts and for a variety of purposes. In some of those situations it might be required (e.g. administrative purposes) to ensure a certain level of authenticity for a *Digital Identity* or for particular attributes of it. This means, to provide reliable confirmation that the attribute values are really the ones that belong to exactly the individual they supposedly belong to. An independent third party, who is trusted by all entities participating in such a construct, could somehow verify (or vouch for) the subject in question. On the other hand, this concept opens up at least one class of attack scenarios. The risk of identity theft, for example, increases dramatically when assigning such value to a *Digital Identity* because the attacker is no longer required to be physically present in order to impersonate that identity or "steal" certain unique identifiers from that person. Instead, it is sufficient to gain access to the areas where those sensitive data are stored. It is noted, that different technical solutions to these issues exist and will be discussed later on.

In the context of this project, and all related work, **Personal Data** is defined as the total amount of data that is part of either an individual's *Digital Identity* or her *Digital Footprint*. On the one hand, this includes all intellectual property (e.g. posts, images, videos or comments) ever created, and all

kinds of tracking data, interaction monitoring and metadata, that is used to manually or automatically enrich content (e.g. geo-location attached to a tweet as meta information). Moreover, it refers to data that is captured by someone or something from within the individual's private living space or property. *Personal Data* is basically understood as every data item reflecting the individual as such and her personality - partially or as a whole.

The european "Data Protection Regulations", on the other hand, defines *Personal Data* as follows:

> *'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person; [16, Para. 4]*

In contrast, the U.S.A. has little to no legislation defining *personal data* and thereby protecting the individual's privacy. There is at least no explicit federal law addressing such areas [17]. Though, some of the existing sectoral laws contain partially applicable policies and guidelines [18]; most of them addressing only data related to specific topics (e.g. health insurance, financial record or lending). In 2015 the White House has attempted to fill that gap with the "Consumer Privacy Bill of Rights Act", but to this date it has not left the draft state. According to the critics, the bill lacks concrete, enforceable rules which consumers can rely on [19]. The draft contains a general definition of *Personal Data*:

> *"Personal data" means any data that are under the control of a covered entity, not otherwise generally available to the public through lawful means, and are linked, or as a practical matter linkable by the covered entity, to a specific individual, or linked to a device that is associated with or routinely used by an individual, including but not limited to […] [20, Sec. 4a1].*

At the end a list of concrete data items follows. Examples include email or

postal address, name, social security number and so on.

Aside from government legislation, several third-party organisations in the U.S. are also allowed to define rules and policies that can overwrite existing laws, namely the *Federal Communications Commission (FCC)*, which recently released "Rules to Protect Broadband Consumer Privacy" for *Internet Service Provider (ISP)* including a list of categories of sensitive information [21]. Thereby the FCC wants *Personally Identifiable Information* (alias *Personal Data*) to be understood as:

> *[…] any information that is linked or linkable to an individual. […] information is "linked" or "linkable" to an individual if it can be used on its own, in context, or in combination to identify an individual or to logically associate with other information about a specific individual [22, Secs. 60–61].*

Despite minor difference in details, the FCC has a serious idea of what is included in *Personal Data* and to whom they belong. Although the FCC's legal participation might be somewhat debatable regarding limitation to certain topics, the "Communications Act" as a U.S. federal law qualifies the FCC to regulate and legislate within its boundaries.

Having a common understanding on what data items belong to a person is the foundation of defining a set rules on how to handle *Personal Data* appropriately. Hence, every business operating within the EU is required[1] to provide its users with a *Privacy Policy*, while in the U.S. for example, as mentioned above, only infrequently and depending on the class of data or context users, must inform its users about how their data get processed and what data items are involved [23]. A user typically agrees on a *Privacy Policy* by starting to interact with the author's business or platform. Thus every *Privacy Policy* is required to be publicly accessible. For instance, not in a restricted area after logging in, but before creating an account. The following example is taken from facebook's current landing page.

> *By clicking Create an account, you agree to our Terms*[2] *and that*

[1] according to article 12-14 of the "EU General Data Protection Regulation 2016/679"
[2] https://www.facebook.com/legal/terms

you have read our Data Policy[3], including our Cookie Use[4]. [24]

It can be viewed more like an information notice, which translates and specifies the prevailing legal situation, rather then a contract, which the user would be forced to read and accept before revealing her data; otherwise known from procedures like software installations, where the user might have to accept terms of use or license agreements. With a *Privacy Policy* at hand, it's up to each individual to decide if the benefits the service offers are worth sharing part(s) of her *Personal Data*, while at the same time reluctantly tolerating potential downsides concerning the privacy of that data. When the vendor considers its policies accepted by a user, her personal data must be processed as stated in those policies but also according to the law. If the policies violate existing law or the vendor effectively goes against the law with its actual doings, penalties might follow. Depending on level and impact of the infringement in addition to what the law itself says, the vendor, while revisiting its wrong-doings in order to improve, might have to compensate affected individuals, pay a fine, or have its license revoked.

Not only privacy laws, but every legal jurisdiction has its limitations - such as its territorial nature - which makes it an inappropriate tool for addressing existing issues and strengthening individuals' privacy and rights in a global context like the *World Wide Web*. Whereas the EU has approved extensive regulations, mentioned above, that are supposed to provide privacy protection and defines the handling of personal data, the U.S., on the other hand, has only subject-specific rules which merely apply to its own citizens. Even though the definition of personal data included in the EU regulation is almost identical to the one introduced for the context of this work, it only applies to vendors and individuals who are part of the EU. Even privacy policies won't help if the vendor is registered in a different area of jurisdiction than the user's location. For those circumstances, international agreements might be established [25], but this approach might still be useless if it either fails to provide proper tools for users to enforce their rights or it is simply ignored by contract partners with or without legal foundation [26] [27].

While the legislation mentioned above is in place, *Ownership* of *Personal*

[3]https://www.facebook.com/about/privacy
[4]https://www.facebook.com/policies/cookies/

Data has no legal basis whatsoever. The concepts of intellectual property protection and copyright might intuitively be applicable, because the data that is defined through the sole existence of the data subject (Digital Identity) and the data that is created by her seems to be her *intellectual property* as well. Such property implies that it is the result of a creative process, but unfortunately facts, which most *Personal Data* are, don't show a *threshold of originality* [28]. Thus, the legal concept of *intellectual property* does not apply. However, Ownership in the context of this work, is understood as a concept of having exclusive control over its personal data and how those data get processed at any given point in time. The exclusive control is emphasized as (A) the right to do what ever is desired with its property and (B) by which rules and mechanisms the ownership can be assigned to someone [29]. This might result in a logistical challenge in which the data subject has to allow data access without losing the exclusive control over that data. In any case some effort might be required in order to preserve ownership as described, caused by the general characteristics that data has.

The european "Data Protection Regulations" mentioned before contain only one occurrence of the word *Ownership*, and it's not even related to the context of *Personal Data* or the data subject. It merely states that *"Natural persons should have control of their own personal data"* [30, Sec. 7]. Whereas Commissioner J. Rosenworcel of the FCC wants *"consumers […] to […] take some ownership of what is done with their personal information."* [31, p. 207] Despite those two exceptions, elaborations on *data ownership* are almost non-existant in current legislation. Instead, the question is typically addressed in *Terms of Service (ToS)* provided by data consumers, which an individual might have to accept in order to establish a (legal) relationship with its author. The individual should keep in mind, that *Terms of Services* can change over time; not necessarily to the users advantage. The contents of a *ToS* must not violate any applicable or related law, otherwise the terms might not be legally recognized. Taking for example the following excerpts from different *Terms of Services*:

> *You own all of the content and information you post on Facebook, and you can control how it is shared […] [32, Para. 2].*
>
> *You retain your rights to any Content you submit, post or display*

> *on or through the Services. What's yours is yours — you own your Content [33, Para. 3].*
>
> *Some of our Services allow you to upload, submit, store, send or receive content. You retain ownership of any intellectual property rights that you hold in that content. In short, what belongs to you stays yours. ([34], under "Your Content in our Services")*
>
> *Except for material we may license to you, Apple does not claim ownership of the materials and/or Content you submit or make available on the Service [35, Para. 5H]*

All those statements are essentially superceded by a subsequent statement within each *ToS*, stating that the user grants the author a worldwide license to do almost any imaginable thing with her data. In case of Apple for example, if the user is *"submitting or posting […] Content on areas of the Service that are accessible by the public or other users with whom [the user] consent to share […] Content"* [35, Para. H1]. It is worth to be noted though, that every *ToS* only refers to the content created by the data subject instead of all her personal data. As mentioned above, personal information are not the subject of intellectual property, but still play an important role in data analytics. Which is why *Privacy Policies* are in place, to ensure at least some enlightenment on the whereabouts of the user's personal data, even though it doesn't compensate the absence of control. Additionally, neither the meaning of *Ownership*, to which the quoted terms refer to, is sufficiently outlined, which results in ambiguity, and thus leaves room for interpretation, nor the proposed definition of *Ownership*, as described earlier, is applicable to these *Terms of Services*, since the data subject loses all its control by design. Handing over data to a consumer effectively disables the exclusive control over the data and eliminates the ability to assign such control. Hence, the *Ownership* to that data doesn't exist any further. That is, no (legislation based) way exists to establish a feasible concept of *Ownership*, unless the data consumer has a motivation to promote the data subject to the sole owner of her data and to honour these privileges.

Leaving the legislation aside for a moment to move away from the top view; data consumers might argue that they have invested a lot in order to enable themselves to collect, process and store personal data, hence it belongs to

them. Whereas from the data subject's point of view this might only be acceptable as long as she herself benefits from that arrangement somehow. Which would be the case if, for example, the data subject uses products, services or features offered by consumers whose quality depends on personal data. If the data subject then chooses to move to a competitor, she would want to bring her personal data with her. Again, the former data consumer would object that competitors would then benefit from all the investment the consumer has already made, but without any effort. While not entirely wrong, two aspects need to be emphasized here. In order to get high quality analytics and therefore be able to make accurate decisions to gain overall improvement, it is (A) vital to put a huge amount of effort in developing the underlying technologies, rather than primarily collecting masses of personal data. But this effort is just the precondition and applies to all of its customers, which again weakens its argumentation. It is followed by (B) an ongoing and recurring process of collecting, aggregating and analyzing actively and passively created data and metadata (e.g. food delivery history or platform interactions and tracking). According to the definition of *Personal Data* through legislation, it appears to be only a fraction of the data, namely personal information, that is involved in these kind of processes. The larger part, which is defined as the subject's *Digital Footprint*, consists of highly valuable metadata [36] [37], but is not covered by law. If the data subject ends the relationship with a collector, at the very least, the personal information should be erased and all remaining data sufficiently anonymized or even handed back. Which again can currently be enforced only by legislation because the data subject has no access to the collector's infrastructure where the data about her is stored.

In summary, approaching the issues from a legislative angle has shown to be fraught with problems on a variety of levels. Not least because it can hardly be proven that vendors behave accordingly. Thus data subjects should not depend on the collector's willingness to respect *Ownership* of *Personal Data* as stated here. Instead a technical approach is proposed to embrace the data subject as the origin of her personal data and to proactively regain control.

2.2 Personal Data in the context of the Big Data Movement

The term **Big Data** initially has been referred to as a *huge amount of data*, containing more or less structured datasets [38], whose size, over time, has started to limit its usability because it exceeded the capabilities of state-of-the-art standalone computer systems and storage-capacity. Instead of reducing the overall size, the issues are addressed by utilizing distributed storing and parallel computing. Aside from challenges in logistic and resource management when for example information retrieval needs to get automated on a large scale [39], this strategy still doesn't answer the question of how to extract useful information from such deep 'data lakes. What questions need to be asked to get answers whose usefulness has yet to be known of? To discover knowledge in order to back decision-making processes, technologies from the fields of *Data Mining*, *Artificial Intelligence* and *Machine Learning* (e.g. Neural Networks) have been adapted. Taken together, this is nowadays known as *Big Data (Analytics)*. Additionally it is a collective term for the practice or approach, described here, as well as the attitude of massively collecting data while tending to neglect people's privacy.

Big Data analytics serve the prior purpose of extracting useful information, whose results depend on the question initially being asked as well as what datasets the corpus contains. General steps involved in such a *Knowledge Discovery* process, can be outlined as follows [40] [41],

1. find and understand problems; formalize question(s) which the results have to answer
2. design data models accordingly and test/correlate them against sample data
3. collect and prepare data
4. process data (data mining)
5. analyse and interpret results
6. utilize discovered knowledge (e.g. make appropriate business decisions)

In general, the majority of businesses are required to have customer relationships. Such relations are based on the transfer of valued goods (e.g. services,

products, etc.) in exchange for compensation (e.g. money). In order to process such a transfer, the vendor requires certain information about the involved customer. Since all entities related to this concept, including the customers, are considered to be human beings, such information most likely includes *Personal Data*. A business normally is eager to grow and, if it has commercial interests as well, it aims for profit maximization. So the business needs to improve and, therefore, it requires the knowledge of what and where its flaws are and how it can improve. To gain such knowledge, analytics based on *Big Data* approaches are part of various business strategies. But this also means that a lot of Personal Data get collected as part of those analytics, since that data is part of many business processes.

As a result of humans being primarily responsible for all money flow in this globalized environment, they also decide on the success of business. This basically means, every process analysis with an underlying commercial dependency somehow involves Personal Data [42] [43] [44] [45], whether this data is mandatory in that process or additionally obtained to,for example, measure and analyze customer behaviour. Common data items involved in big data analytics start with gender, age, residency or income, goes on with time series events like changing current geo-location or web search history and continues all the way up to health data and self-created content like posts, images, or videos. Depending on the data item though, those data might not be easy to collect. In general, most businesses obtain data from within their own platforms. Some data might even be in the customer's range of control (e.g. customer or profile data), but most of the data is created during direct (content creation, inputs) or indirect (transactions, meta information) interaction with the business. The sensitivity level of involved Personal Data is determined by how big the benefit is for the customer in comparison to what the vendor's demand is from the customer's commitment (e.g. required inputs, or usage requires access to location information).

From a technical perspective, collecting indirectly created data is as simple as integrating logging or debugging statements in the program logic. Since most vendors and organisations nowadays have a business that is partially happening through the internet or is even completely based on it, most scenarios of transactions utilize server-client architectures. Furthermore, the *always-on* philosophy has evolved to an imperative and implicit state of

devices. Standalone software, installed on a personal computer, calls the vendor's infrastructure that is located in the cloud on a regular basis, just to make sure that its user behaves properly while, en passant, the vendor is provided with detailed user statistics. The web-browser already invokes a common narrative that requests happen sporadically in the background, although they are preventable. But when it comes to native mobile applications, it's almost impossible [46] to notice such behaviour or even prevent them from exposing potentially sensitive information. Those developments in architecture design have enabled a system-wide collection of potentially useful information on a large scale [47]. Logging transactions triggered by the user on the client and forwarding the resulting logs to the back end infrastructure, or keeping track of all sorts of transactions directly in the back end; all these collected chunks of data are then enriched with meta information before running together in a designated place where they are finally stored and probably never removed, waiting to be analyzed some day.

Within *Big Data Communities* sometimes the *big* is misinterpreted as, regardless of what the problem is that needs to be solved, speculatively gathering as many data items as possible with the hope that, in the future, those data might become valuable. This mindset is reflected by the oft-cited concept of the three *Vs* (Volume, Velocity, Variety) [48]. It is not entirely wrong though, because it's the nature of pattern and correlation discovery to provide increasing quality results [49], when the overall data corpus is enriched with larger and more precise datasets. When new technologies emerge and are hyped, questioning their downsides and potential negative mid- or long-term impacts is typically not a high priority. The focus is instead to experiment and try to reach and perhaps breach boundaries while beginning to evolve. Non-technical aspects such as privacy and security awareness don't come in naturally, instead the adoption rate has to increase before more and diverse research occurs that might uncover such issues. Only then can they be addressed properly and on different levels - technical, political, and social. Hence, the *Big Data Community* itself is able to evolve, too.

Big Data and *Knowledge Discovery* is a balancing act between respecting the user's privacy and having enough data at hand so that initial questioning can be satisfied by the computed results. Aside from having specific domain knowledge of the used technologies, people working in these fields need to be

aware of downsides or pitfalls and also have to be sensible about the ramifications of their approaches and doings. Such improvements are already happening, not only originating from the community's forward thinkers [50], but also advocated by governments and consumer rights organisations, as stated in the previous section (see *Legislation*). Even leading Tech-Companies have begun trying to do better [51] [52] [53].

In various science and research areas it's also very common to gather and store enormous amounts of data. In these contexts the data and its analytical results are used to, for example, run complex simulations (e.g. weather, population, diseases, hardware, physics), monitor and analyse complex proceedings (e.g. nature, infrastructure, behaviour), explore the unknown (e.g. universe), or even to imitate a famous painter [54]. But unlike in the academic sector, the commercial interest within the private sector is much larger, therefore Personal Data are in great demand, because forecasting customer behaviour rather than global warming is much more valuable in that context.

It is the logical conclusion to distribute and scale horizontal when the data demand exceeds latest hardware capacity and capability, but it's no justification for thoughtless data collecting. Ultimately, it's not the amount of data that counts, but how it is handled (to not loose its usefulness); what data flows into those data lakes is up to the questioner, which might not act in the interest of everyone. Therefore data subjects need to regain control and actively participate in formulating these questions.

2.3 Personal Data as a Product

The *Big Data* paradigm itself, as mentioned before, merely provides a structured and technically-based method to uncover non-obvious or non-visible information from self-made data silos in order to assist in making (correct) (business) decisions. Though, when asking data collectors about their actual motivation, most likely the answer would be something along the lines of typical PR-phrasing such as *"we want to have a better understanding of our customers"*. In the long run, this means to increase revenue, but in the short term, to do what exactly? Maybe to predict what might be the next thing people are supposed to buy, or what things they would probably like

to consume but do not yet know of?

In order to try comprehending such perspective, let's take a look at some examples.

A) An advertising service utilizes tracking data for targeted advertising. The more information the service has on an individual, the more accurate decisions it is able to make about what ads are the ones that the individual most likely will click on and disclose with a successful purchase. As a result the placed advertisement becomes more valuable to the advertiser, because of its high precision. This again causes the service to increase the charges for serving and presenting the ad, because the overall quality of its service - the product - has improved.

B) Content recommendation engines of large streaming providers, regardless of the content, serve as another example. This feature is also underpinned by extensive data aggregation of customer information (user profiling), such as consumption histories (e.g. watch list), favoured content, friend's consumption, or any kind of trackable platform interaction. This means that the more information the service has on the user, the more precise are the assumptions about current mood, taste and interests, which leads to more suitable recommendations. At the end the user feels well taken care of and therefore values the service.

C) Another example is *Google Traffic* [55] [56], a service that is integrated as a feature in *Google Maps*, a web-based mapping service from Google. *Google Traffic* visualises real-time traffic conditions, while using *Maps* as a navigation assistant in order to provide the user with a selection of possible paths. These paths are enriched with a duration that takes the traffic conditions into account. The data required to offer such information is supplied by mobile devices, constantly sending its current position including a timestamp to Google's infrastructure. This however, is only possible, because Google's services are widely used in addition to the significant market share of mobile devices [57] that are driven by Android - a mobile operating system developed by Google that deeply integrates with its services. The same assumptions can be made as stated in the previous examples. The more geo-location data is obtained, the more precise the information about traffic conditions

are. Since this scenario demands real-time information, it adds the *time* component as an additional level of complexity to the problem.

The impact on the society of the first group of examples might be questionable and it is doubtful if proper applications even exist. However, an adjustment of perspective reveals additional categories of scenarios, for example:

D) Planning and managing human resources for special occasions with big crowds, such as huge events or emergency situations where attendees might become endangered and require help [58]

E) Predicting infrastructure workloads (e.g. power grid) [59]

F) Making more accurate diagnostics to improve a patient's therapy [60]

G) Finding patterns in climate changes, which would otherwise not have been detected [61]

Even though the described examples require a large corpus of data and utilization of *Knowledge Discovery*, some of them might not necessarily depend on *Personal Data*. For other scenarios, Personal Data are indispensable, and still other scenarios only implicitly rely on data collected from individuals. As noted in the previous section on Big Data[5], it depends on the purpose, which can be defined i.a. through a existing *business model*. But at least in those examples it seems to be common motivation to primarily improve and enhance the collector's product in order to satisfy its customers - yet again that depends on what is considered as product and who are the customers, generally indicated by the money flow.

Generalizing businesses based on its affiliation to an industry sector is hardly an argument for utilizing Personal Data, but empirically a rough attribution often suffices in order to get a picture of possible business models. With that said, the following observation can be made. When putting a Top 10 List of industries utilizing *Big Data* [62] and a visualization showing categories of Personal Data targeted by data collectors [63] side by side, it appears to be that at least seven[6] of these industries can be identified as data collec-

[5]personal-data-in-the-context-of-the-big-data-movement

[6]Banking and Securities; Communication, Media & Entertainment; Healthcare Providers; Government; Insurance; Retail & Wholesale Trade; Energy & Utilities

tors, whereas less then a half[7] are participating in being a Data Broker, but almost all of them are suspected to target people's Personal Data, whether obtained by themselves or acquired from *Data Brokers* (more examples are mentioned in *Corporate surveillance, digital tracking, big data & privacy* [64]). Therefore, it's safe to say that Personal Data is considered either as the direct product, especially from a Data Broker's point of view, or an indirect product due to its essential part in *Big Data* approaches. The former generates direct revenue by selling these data and the latter might affect a business in a positive matter.

To conclude, the possession of more and precise information on individuals leads to increased revenue for the possessor. However, using Personal Data to improve a product does not necessarily become an issue unless the data owner is not the customer. Taking a closer look at the business model often reveals the role of Personal Data. Thus Personal Data becomes the currency and its owner becomes the product, while the possessor becomes controller and profiteer. This rather unsatisfying situation is going to be addressed further on. For example, by shifting the individual's role to offering its own data to those who have previously collected and sold them.

2.4 Related Work

NOTICE: All research, projects, studies and work mentioned in this section represent only a fraction of what's already been done in this field and should be therefore seen as an excerpt containing a selection of the most related and relevant approaches.

The idea of a digital vault, controlled and maintained by the data subject, is not new. It holds her most sensitive and valuable collections of bits and bytes, protected from all the data brokers, collectors and authorities, while interacting with the digital and physical world, opening and closing its door from time to time to either put something important to her inside or releases important information for someone else. In the mid and late 2000s the growth of computer performance and capacity crossed its zenith (see

[7] Banking and Securities; Communication, Media & Entertainment; Insurance; Energy & Utilities

Moore's Law [65]). At the same time, the internet started to become a key part in many people's lives and in society as a whole. Facilitated by these circumstances, *Cloud Computing* has been on the rise ever since, causing the shift towards distributed processing and patterns alike, thereby making it possible to rethink solutions from the past and try to go new ways, namely a breakthrough 2007 in *Neuronal Networks* courtesy of G. Hinton [66]. As a result, fields like *Data Mining*, *Machine Learning*, *Artificial Intelligence* and most recently combined under the collective term called *Big Data*, have gained a wide range of attention as tools for knowledge discovery. In almost any industry a greater amount of resources is invested in these areas [67].

The initial motivation for this project can be understood as a counter-movement away from all the data silos in the cloud, returning to a focus on privacy, the individual, and her digital alter ego.

From simple middleware-solutions to full-fledged software-based platforms through to embedded hardware devices, a great variety of approaches have started to appear in the mid 2000s, continuing to this day. However a side effect was, over time, various research teams and projects have invented and coined different terms, which, in the end, all refer to the same (or similar) concept. The following list shows some of them *(alphabetical order)*:

- Databox
- Identity Manager
- Personal …
  - Agent
  - Container
  - Data Store/Service/Stream (PDS)
  - Data Vault
  - Information Hub
  - Information Management System (PIMS)
- Vendor Relationship Management (VRM)

One of the first occurring research projects was *ProjectVRM*, which originated from *Berkman Center for Internet & Society* at *Harvard University*. As its name suggests, it was inspired by the idea of turning the concept of a *Customer Relationship Management (CRM)* upside down. This puts the vendor's customers back in charge of their data previously managed by

vendors. It also solves the problem of unintended data redundancy - from the customers perspective. Over time the project has grown to the largest and most influential one in this research field. It has transformed into an umbrella and hub for all kinds of projects and research related to that topic [68], whether it's frameworks or standards, services offering (e.g. privacy protection), reference implementations, applications, software or hardware components. *VRM* became more and more a synonym for a set of principles, for example *"Customers must have control of data they generate and gather. [They] must be able to assert their own terms of engagement"* [69, Paras. 3–4]. These principles can be found in various forms across much work done in this area of research.

Another work of research worth mentioning, because of the foundational work it has done, is the european funded project called *Trusted Architecture for Securely Shared Service* (TAS3) [70]. This project has led to an open source reference implementation called *ZXID*.[8] The major goal was to develop an architecture, that generalizes various approaches towards a non context-agnostic solution fitting into more sophisticated and dynamic scenarios, while still respecting commercial businesses (vendors) and users (customers), but at the same time facilitating a high level of user-centric security and privacy i.a. by a developed policy framework. Due to these requirements the architecture ended up being rather complex [71]. The implementation, called *ZXID*, involves several standards, for example SAML 2.0[9] and XACML,[10] and has just three third-party dependencies, *OpenSSL*, *cURL (libcurl)* and *zlib*. As of now it even supports Java, PHP and Perl. The project has lasted for a period of 4 years. After completion in 2011, research has continued i.a. by the *Liberty Alliance Project*. It is now part of the *Kantara Initiative* [72], including all documents and results. These results are occasionally referenced, i.a. by the IEEE [73].

A research project, which is probably the closest to what this work aims to create, bears the name *openPDS* [74] and is done by *Humans Dynamics Lab* [75], which again is part of *MIT Media Laboratories*. Despite the usual concepts of a *openPDS*, it introduces multi-platform components and user

[8] more information on the project, the code and the author, Sampo Kellomäki, can be found under *zxid.org*

[9] Security Assertion Markup Language 2.0

[10] eXtensible Access Control Markup Language

interfaces including mobile devices, and also separates the persistence layer physically. This approach enables place- and time-independent administrative access for the data subject. Moreover, with their idea of *SafeAnswers* [76], the team goes a step further. The concept behind that idea is based around *remote code execution*, briefly described in one of the user stories in the first chapter. It abstracts the concept of a data request to a more human-understandable level: a simple question. This question consists of two representations: (A) a human-readable question of a third party, and (B) a code-based representation of that question, which gets executed in a sandbox on the data subjects's *openPDS* system with the required data as arguments. The data, used as arguments, is implicitly defined trough (A). The output of that execution represents both answer and response.

Aside from all the research projects done within the academic context, applications with a commercial interest have also started to occur in a variety of sectors. Microsoft's *HealthVault* [77], for example, which aims to replace the patient's paper-based medical records and combine them in one digital version. This results in a patient-centered medical data and document archive, helping doctors to make more accurate decisions on medical treatment, because they have more knowledge obtained, for example, from a personal medical history.

Meeco [78] [79], based on the MyData-Project [whitepaper_2014_mydata-a-nordic-model-for-human-centered-personal-data-management-and-processing], essentially just replaces platform-agnostic advertisement service providers with a closed environment and serves ads on its own. Although data subjects in this environment are provided with more control over what information they reveal, this approach doesn't take the eagerly demanded next step of getting rid of the advertisement market as a revenue stream and, instead, finding a suitable business model that focuses on the data subject, not surrounding them with just another walled garden.

A recently announced project, sponsored by Germany's *Federal Ministry of Education and Research*, but developed and maintained primarily by *Fraunhofer-Gesellschaft* in cooperation with several private companies like *PricewaterhouseCoopers AG*, *Volkswagen AG*, *thyssenkrupp AG* or *REWE Systems GmbH*, is the so called *Industrial Data Space* [80]. The project uni-

fies both, research and commercial interests and runs over a time period of three years until the third quarter of 2018. It aims *"[…] to facilitate the secure exchange and easy linkage of data in business ecosystems"*, where at the same time *"[…] ensuring digital sovereignty of data owners"* [81, p. 4]. Here, "data owners" seems not referring to *data subjects*, but to organisations who are in possession of the demanded data. So, it will be interesting to see how these two related but distinct objectives come together in the future. Based on the white paper, the project's focus mainly seems to lie in enabling and standardizing the way companies collect, exchange and aggregate data with each other across process chains to ensure high interoperability and accessibility.

The following is a list of further research projects, work and commercial products regarding the topic of *personal data*:

Research

- Higgins[11]
- Hub-of-All-Things[12]
- ownyourinfo[13]
- PAGORA[14]
- PRIME[15]/PrimeLife[16]
- databox.me (reference implementation of the *Solid framework*[17])
- Polis[18] (greek research project from 2008)

Organisations

- Open Identity Exchange[19]
- Qiy Foundation[20]

Commercial Products

[11]https://www.eclipse.org/higgins
[12]http://hubofallthings.com/what-is-the-hat
[13]http://www.ownyourinfo.com
[14]http://www.paoga.com
[15]https://www.prime-project.eu
[16]http://primelife.ercim.eu
[17]https://github.com/solid/solid
[18]http://polis.ee.duth.gr/Polis
[19]http://openidentityexchange.org/resources/white-papers
[20]https://www.qiyfoundation.org/

- MyData[21]
- RESPECT network[22]
- aWise AEGIS[23]

2.5 Standards, Specifications and related Technologies

This project strives to involve well known and commonly used standards, instead of reinventing technologies that already exist, to acknowledge the work that's been done, to increase the chances of interoperability, and to lower the required effort to integrate with third parties or other APIs. The following section briefly describes potential technologies and outlines the purposes they might serve and why.

As it is assumed that data is transferred over a non-private channel, several features might be desired in order to provide trustworthy and secure communication. One important aspect might be that no one except sender and receiver are able to know and see what the actual data are. Utilizing cryptographic technologies can provide such properties. **Symmetrical Cryptography**, as one option, provides a possible solution. It states that sender and receiver arrange a common secret, which is used to encrypt as well as to decrypt the data. Anyone who is not allowed to access that information must not be in possession of the secret. To agree on a secret without compromising it during that process, both entities either need to switch to a private medium (e.g meet physically and exchange) or have to use a procedure in which the entire key is not exposed to others at any single point in time. Such a procedure could be, among others, the **Diffie-Hellman-Key-Exchange** [82], which is based on the mathematical discrete logarithm problem. It allows two parties to agree on a common *secret* while using a non-private channel. The data exchanged during the process alone cannot be used to exploit the secret. The possibility to securely communicate on a non-private channel is also one of the strengths of **Asymmetrical Cryptography** *(or public-key cryptography)* [83], which is underpinned by a *key-pair* for every communication participant; one key is *public* and the other one must be

[21] https://mydatafi.wordpress.com
[22] https://www.respectnetwork.com
[23] http://www.ewise.com/aegis

kept strictly *private*. The *public* key is used to *encrypt* the data, and only the *private* key can be used for *decrypting* the cipher. With **RSA** [84], one of the *cryptographic systems* used in asymmetrical cryptography, it's even possible to switch the roles of the keys, meaning regardless of which key is used to encrypt the data, only the corresponding part is able to decrypt the cipher. If this technique is combined with the concept of digital signatures (encrypted fingerprints from data), the result is a so called certificate, that provides integrity and authentication. It consists of the *public key* attached with owner information and a digital signature of that key.

*Hypertext Transfer Protocol (**HTTP**)* [85], commonly known as part of the *Application Layer* in the *Open Systems Interconnection model (OSI model)* [86], is one of key technologies the *World Wide Web* is based on. This stateless protocol is mainly used to transfer any media type reliably between internet endpoints. It most likely will fulfill the same purpose in the context of this work, because it implements a server-client pattern at its very core. This results in a communication scenario with a one-to-one relationship. Whether internal components, locally on the same host or as part of a distributed system, talk to each other or data consumers interact with the system, this protocol transfers the data that needs to be exchanged. The relevance and use cases of features introduced with Version 2 [87] of the protocol are not yet explored.

Putting *HTTP* on top of the **Transport Layer Security (TLS)** protocol [88] results in secure *HTTP*, also know as *HTTPS*. TLS provides encryption during data transport, which reduces the vulnerability to *man-in-the-middle* attacks and thus ensures not only confidentiality, but data integrity too. *Asymmetric cryptography* is the foundation for the connection establishment in *TLS*, hence it also allows to verify the relation between the identity of the communicating party and the presented certificate. This procedure can be performed by both parties. Depending on the integration, it could even be used to authenticate that party. Though relying on those cryptographic concepts requires additional infrastructure. Such an infrastructure is known as *Public Key Infrastructure (PKI)* [89]. It manages and provides keys and certificates for a dedicated scope of entities in a directory, including information related to their owners. *PKI*s are commonly utilize a hierarchical chain of trust, although cross-signing is possible as well. A *Certificate Authority*

(CA), as part of that infrastructure, issues, maintains and revokes digital certificates. The infrastructure that is needed to provide secure HTTP connections for the internet is one of those *PKI*s - a public one and probably one of the largest. It is based on the widely used IETF[24] standard *X.509* [90].

Unlike *HTTP*, **WebSockets** [91] provide the concept of ongoing bidirectional connections on top of a TCP[25] connection, though connection establishment is happening via HTTP(S). The result is a connection *upgrade*, which enables both protocols running side by side on the same server port. This makes it also possible to use TLS for encrypting connections. So, instead of mimic real-time data exchange and pending remote procedure call responses with long-polling over HTTP, the secure WebSocket protocol (*wss*) provides such patterns as well, but in a more efficient way. It is conceivable to use *WebSockets* as the transport technology to communicate between the system's components and even with external parties.

While the transport of email-based communication is also underpinned by *TLS*, the emails themselves still exist in plain text at all times. Utilizing end-to-end encryption for such communication is another example of applying *public-key cryptography*. Here, a more decentralized and liberal approach of a *PKI* is enforced. Instead of a *trust chain* structure, public keys are typically signed or cross-signed with no hierarchical order, such as known from the *PKIX (or PKI based on X.509)*. This approach, known as **web of trust**, consists of multiple *public key servers*, in which all entities (user; senders and recipients) are signing each other's public keys. The more users have signed a public key, the higher the level of trust is that the encrypted content can only be read by whom the signed key-pair belongs to. Public keys are simply uploaded by its owners to the key servers mentioned before. If someone wants to write an email to others, for every recipient the relating public key can be obtained from these public servers, so that the email is then encrypted with those keys - for every recipient individually. Therefore the email is only readable by the owner of the key's private part.

[24] Internet Engineering Task Force; non-profit organisation that develops and releases standards mainly related to the Internet protocol suite

[25] *Transmission Control Protocol*, provides a reliable host-to-host connection; part of the Internet Protocol Suite (see RFC 761)

In the past years different countries around the world started to introduce information technology to the day-to-day processes, interactions and communications between public services and their citizens. Processes like changing residence information or filing tax reports, are all summarized since then under the name *E-government*.[26] One of those developments is the so called **Electronic ID Card**, hereinafter called *eID card*. Equipped with storage, logic and interfaces for wireless communication, those *eID cards* can be used to store certain information and digital keys, or to authenticate the owner electronically to a third party without being physically present. Such an *eID card* was introduced also in Germany in 2010. The so called *nPA*[27] has been an important step towards an operational *e-government*. Aside from minor flaws [92] and disadvantages [93] an *eID card* might come along with, the question here is, how can such technology be usefully integrated in this project and is it plausible to do so. As an official document, the card has one major advantage over inherent, self-configured or generated secrets like passwords, fingerprints or TANs.[28] It is *signed* by design, which means, by creating this document and handing it over to the related citizen, the third party (or *'authority'*) - in this case the government - has verified the authenticity of that individual. Although, this procedure does not guarantee that the third party has no copy of the private key(s) stored on the *eID card*.

Another technology emerging as part of the *e-government* development, is the german **De-Mail** [94]. It's an email service that is meant to provide digital infrastructure and mechanisms to exchange legally binding electronic documents. When registering for a De-Mail account, the provider is compelled to verify the identity of the applicant (e.g. government-issued identification document) in order to ensure sender and receiver's authenticity, whereas for sending a De-Mail, only the account credentials (identifier, password) are required. A successful authentication is considered sufficient to prove the user's identity. The De-Mail itself is not encrypted at any time, only the transport layer, on which the document is transferred, is based on TLS and thereby encrypted. Upon arrival at the providers De-Mail Server, only the provider signs the De-Mail and then sends it to the server of the receiver's provider. That is, only De-Mail providers are able to sign their users

[26] Electronic government
[27] in german so called *elektronische Personalausweis (nPA)*
[28] Transaction authentication number

De-Mails. It is optional and up to every user to utilize *end-to-end* for complete confidentiality, or *Qualified Electronic Signatures (QES)* [95], which is a capability of the *nPA* and can be used to verify the author's identity.

JSON[29] is an alternative to the *Extensible Markup Language (XML)*, which in this context it is only referred to as another text-based data serialization format. The JSON-syntax is inspired by the JavaScript object-literal notation and is therefore more readable compared to XML. It's heavily used in web contexts to transfer data via *HTTP*. Like XML, its structuring mechanisms allow i.a. type preservation and nesting, which enable the representation of more complex data structures, including relations.

If, due to being part of a distributed software, some components are required to not maintain any state, either that software architecture needs to be changed so that the state is no longer needed in that component, or the state needs to be embedded into the process communication so that it's passed from one component to the other. This is a common use case for a **JSON Web Token (JWT)** [98]. A *JWT* is formatted as *JSON*, but URL-safe encoded with *base64url* [99], a version of the widely used *Base64* encoding, before it gets transferred. The token itself contains the state. Here is where the use of *HTTP* comes in handy because the token can be stored within the HTTP header, which consists of additional meta information about the HTTP connection and data that is exchanged between client and server. Therefore the token can be passed through all communication points, where the data can then be extracted and verified. Such a token typically consists of three parts: (A) information about itself *(header)*, (B) a *payload*, which can be arbitrary data such as user or state information, and (C) the so called *signature*, which effectively is a *hash-based message authentication code (HMAC)* [100] preserving the integrity of header and payload. All three parts are separated with a period. Additional standards define encryption *(JWE*[30]*)* to ensure confidentiality, and digital signatures *(JWS*[31]*)*, which enables others than only the signer to validate the *signature*. Using a *JWT* for authentication purposes is described as *stateless authentication*, because the verifying entity doesn't need to be aware of session IDs nor any other infor-

[29] The JavaScript Object Notation (JSON) Data Interchange Format; ECMA Standard [96] and Internet Engineering Task Force RFC 7159 [97]

[30] JSON Web Encryption, Internet Engineering Task Force RFC 7516 [101]

[31] JSON Web Signature, Internet Engineering Task Force RFC 7515 [102]

mation about a session. So, instead of the backend interface being burdened to check a state on every incoming request in order to verify permissions, which required to maintain a state in the first place, it just needs to decrypt the token and proceed according to the contained information.

Furthermore, the open standard **OAuth** defines a process flow for authorizing third parties to access externally hosted resources, such as the user's profile image on a social media platform. The authorization validation is done by the help of a previously generated token. However, generating and supplying such tokens can be initiated in a variety of ways depending on the underlying architecture and design, for example, with the user entering her credentials. This design seems to misleadingly encourage developers to integrate *OAuth* as an authentication service [103] rather then an authorization service, whether as an alternative or as an addition to existing in-house solutions. In doing so, the application authors pass the responsibility on to the OAuth-supporting data providers. OAuth *version 1.0a* [104], which is rather considered a protocol, provides confidentiality by encrypting data before it gets transferred, and integrity of transferred data by using signatures. Whereas *Version 2.0* [105], labeled as a framework, requires a *TLS* encrypted communication channel and thus passes on the responsibility for confidentiality and integrity to the transport layer below. It also supports additional process flows for scenarios involving specific platforms such as *"web applications, desktop applications, mobile phones, and living room devices"* [106, Para. 1].

OpenID on the other hand, as another open standard, is explicitly designed to validate the authenticity of a requesting user. An in-depth description of the whole process can be found in the protocol's same-titled specifications [107]. With decentralization in mind, the protocols's nature encourages to design a distributed application architecture, similar to the idea behind a *microservice*,[32] but without owning all services involved - *decentralized authentication as a service* so to say. An application owner doesn't have to write or implement its own user authentication and management system, instead it is sufficient to merely integrate those parts that are needed to support signing in with *OpenID*, which is typically a client interacting with the

[32] a pattern to split a monolithic application into smaller, independent, parts that are easier to maintain but still communicating with each other

Identity Provider. Equally the user is not required to register an account for every new app, instead she can use her *OpenID*, already created with an Identity Provider, to authenticate with the application. The extension *OpenID Attribute Exchange* allows the importing of additional profile data, similar to what OAuth is meant to be used for. *OpenID Connect* [108] is the third iteration of the OpenID technology. While *facebook login* [109] (formerly *facebook connect* [110]), for example, uses OAuth also for authentication (known as pseudo-authentication [111]) instead of just authorising entities, *OpenID connect* on the other hand, provides authentication in an additional layer built upon *OAuth2.0* and *JWT*. Previous versions of OpenID have provided the concept of extension in order to add functionality such as accessing profile information. This ability is now part of the core facilitated by OAuth, so that a user's identity can share certain data with third parties via *REST* interface.

The concept, known as *Representational State Transfer*, or **REST**,[33] refers to a common set of principles on how to design web resources and their interaction. It primarily defines server-client communication in a more generic and therefore interoperable way. Aside from hierarchically structured URLs, which can reflect semantic relations or hierarchical order between data items, it involves a rudimentary vocabulary[34] for HTTP requests to provide basic CRUD-operations[35] across distributed systems. The entire request needs to contain everything that is required to be processed on the REST-server, for example state information and possibly authentication parameters. A RESTful API typically has the purpose of exposing certain features provided through the platform or service to third parties in order to synergistically integrate with them. But utilizing these principals for all internal server-client interaction is also very common. This concept can also be understood as a proxy to the actual business logic in the back end.

The concept of an application (or software) **container** is about encapsulating runtime environments by introducing an additional layer of abstraction. A container bundles just those software dependencies (e.g. binaries) that are absolutely necessary so that the enclosed program is able to run prop-

[33] *Representational State Transfer*, introduces by Roy Fielding in his doctoral dissertation [112]

[34] HTTP Methods or Verbs [113] (e.g. GET, OPTIONS, PUT, DELETE)

[35] Create, Read, Update, Delete

erly. The actual separation from the host system is done, aside from other technologies, with the help of two features provided by the Linux kernel. *Cgroups*,[36] which defines or restricts how much of the existing resources can be used by a group of processes (e.g. CPU, memory or network). Whereas *namespaces* [115] defines or restricts what parts of the system can be accessed or seen by a process (e.g. filesystem, user, other processes). The idea of encapsulating programs from the operating system-level is not new. Technologies, such as *libvirt*[37], *systemd-nspawn*[38], *Jails*[39], or *hypervisors* (e.g. VMware[40], KVM[41], VirtualBox[42]) have been used for years, but some of them were usually too cumbersome and never reached a great level of convenience, so that only people with expertise have been able to handle systems built upon virtualization, whereas people with other backgrounds couldn't and weren't very interested, at least until *Docker*[43] and *rkt*[44] emerged. After some years of separated work, both authors, accompanied by additional parties, recently joined forces in the *Open Container Initiative* [116], which promises to harmonize the diverged landscape and start building common ground to ensure a higher interoperability. That, in turn, started to raise the demand for sophisticated orchestration. It also marks the initial draft of specifications for runtime [117] and image [118] definition for *application container*, which are still under heavy development. This concept of *containerization* also has the side-effect of making the application platform-agnostic, because it allows a certain set of software to run on a system which might otherwise not support that software (e.g. mobile devices); it just requires the runtime to be supported.

The *query language (QL)* **GraphQL** [119], developed by Facebook Inc., abstracts multiple data sources into an unified API or resource. The additional abstraction layer allows different storage technologies to be seamlessly queryable without using their native *query languages*. The result is provided in JSON format, which naturally supports graph-like data structures. This

[36] control groups [114]
[37] https://libvirt.org
[38] https://wiki.archlinux.org/index.php/Systemd-nspawn
[39] https://www.freebsd.org/doc/handbook/jails.html
[40] https://www.vmware.com
[41] https://www.linux-kvm.org
[42] https://www.virtualbox.org
[43] https://www.docker.com
[44] https://coreos.com/rkt

is utilized in GraphQL and implicitly embraced through its purpose of abstraction. Data items that might be somehow related but stored in different locations, can be obtained so that both end up in the same object through which they are related or indirectly linked to each other. The shape of a query is later mirrored by the result. GraphQL is not only an abstraction towards a more generic query language. It also separates almost any operation and the flow control from the the *QL* and moves it into a second layer. The so called *GraphQL* server is responsible for resolving and executing queries.

The term **Semantic Web** bundles a conglomerate of standards released by the *World Wide Web Consortium (W3C)*.[45] It is based on an idea called *web of linked data*, which aims to *"enable people to create data stores on the Web, build vocabularies, and write rules for handling data"* [120, Para. 1]. The standards address syntax, schemas, formats, access control and integrations for several scopes and contexts. Among others, the following three technologies are essential for the *Semantic Web*. *RDF*[46] basically defines the syntax. *OWL*[47] provides the guidelines on how the semantics and schemas (vocabulary) should be defined and with SPARQL [123], the query language, data can be retrieved. The idea of the *Semantic Web* tries to utilize the web as a database of queryable data with URIs that are embedded in arbitrary websites. Imagine a person's email address, which is available under a specific domain (preferably owned by that person) - or to be more precise, a URI *(WebID) [124]* - and provided in a certain syntax *(RDF)*, tagged with the semantic *(OWL)* of a email address, all together embedded in an imprint of a website. This information can then be queried *(SPARQL)*, if the unique identifier of that person *(URI)* is known. While defining the standards, a main focus was to design a syntax that is at the same time valid markup. The vision behind this: embracing the concept of a single source of truth (per entity) and embedding or linking data items rather then creating new instances of the same data that might be only valid at that point in time - in short, preventing redundant work and outdated data. Related to the *Semantic Web* is the a project called **Solid**.[48] Backed by the *WebAccessControl* [126] system and based on the *Linked Data* principles including the stan-

[45] international community that develops standards for the web
[46] Resource Description Framework [121]
[47] Web Ontology Language [122]
[48] social linked data [125]

dards just mentioned, the project focuses on decentralization and personal data management while developing a specification around this. A reference implementation called *databox* [127] follows the specification process.

3

Core Principles

Right from the start, a set of principles have been built as the cornerstones and orientation marks of the idea behind the *ReFlowd*. These are meant to be reflected by the upcoming *Open Specification* and will be explained further within the following sections.

3.1 Data Ownership

Depending on the perspective, the question about ownership of certain data might not be trivial to answer. As stated in *Chapter 3 - Digital Identity, Personal Data and Ownership*, ownership requires a certain amount of originality to become intellectual property, which is not the case for personal data - at least for all the non-creative content. Thus, there is no legal ground for an individual to license those data that obviously belong to her. Switching the perspective from the *data subject* to the *data consumer*; for them, several laws exist addressing conditions and rules regarding data acquisition, processing and usage.
Putting aside the absence of legislation regarding data ownership, it cannot

be denied that it seems unnatural to not be the owner of all the data that reflect a person's identity and the person as an individual. So instead of defining rules meant to protect data subjects but demand data consumers to comply, the proposal here is to put the entity to whom the data is related, in control of defining who can access her data and what accessors are allowed to do with it. This would make the *data subject*, per definition (see Ownership), the effective owner of those data. Although, it is to be noted, that the legislation for data consumers mentioned before, remains highly important since this project is not able to cover every single use case that might occur.

Promoted from the data subject to the data owner and therefore being the center of the *ReFlowd*, the *operator* of the *ReFlowd* gains the ability to have as much control as possible over all the data related to her, in order to determine, in a very precise way, what data of hers can be accessed by third parties at any point in time and also to literally carry all her personal data with her.

3.2 Identity Verification

If an instance of such a system *(ReFlowd)* is going to be the digital counterpart of an individuals identity or her *"personal agent"* [3, p. 46], then everyone who relies on the information that agent provides must also be able to trust the source from where that data originates and vice versa; the *operator* too need to verify the authenticity of the requesting source; regardless of whether it's during initial registration or further *access attempts*. Based on these mechanisms, the system can also provide an authentication services that third parties might utilize to outsource the authentication of its owner, including enabling additional security factor steps.

3.3 Reliable Data

Being able to verify the authenticity of a communication partner is only half of the task. Data consumers also need to trust the data itself, according to the following criteria. (A) *integrity* - meaning the recipient can verify that the

received data is still the same or if someone has tampered with the obtained data. (B) *authenticity* - meaning it is somehow ensured, or the recipient must be certain that the received data actual belongs to the individual, meaning the data truly reflects attributes of the individual from whom the data come from. A negative result of that check should not cause a termination of the process, but instead should warn data consumers unverified authenticity, so that they themselves can decide if and how to proceed.

3.4 Authorization

Controlling its own data might be the most important ability of such a system, because the data owner is enabled to grant permission to any entity who wants to obtain certain information, in a potentially automated way, about her. She can authorize, as precisely as desired, how long and what data (sets, items or fields) are accessible by a single entity. Thereby, the data owner is able to change the *access permissions* for any entity at any point in time, for example, when motivated by a noticed incident.

3.5 Supervised Data Access

Rules and constraints might be one way to handle personal data demands of third parties. But a plain *query-and-respond-data* approach could be replaced by a more goal-driven concept that prevents data from leaving the system. It allows the execution of a small program within a locally defined environment, computing only a fraction of a larger computation initiated by the *data consumer* beforehand; similar to a distributed Map-Reduce concept [128]. The opposite, but also conceivable, approach would be to provide either some software to the *data consumer*, which is required in order to access the contents of a response, or a runtime environment that queries the system by itself.
In general, it is not very likely that data consumers, who have been granted certain access, would renounce their privileges. Thus it is vital that the data owner is the one in charge of canceling access permissions or applying

appropriate changes to them. Supervising methods provide an appropriate way to make data available to those who are eager to consume them.

3.6 Containerization

Abstracting an operating system by moving the bare minimum of required parts into virtualizations results in an environment setup that can, depending on the configuration, fully encapsulate its internals from the host environment. This approach yields some valuable features, such as

(A) Effortless portability, which reduces the requirements on environment and hardware

(B) Higher flexibility in placing components, through which advantages can be made out of characteristics that other devices might bring with them

(C) Isolation and reduction of shared spaces and scopes, which, for example, can prevent side effects.

Together, these lead also to an overall security improvement, or at least it enables new patterns to improve those aspects. Furthermore, allows for more versatile and diverse scenarios, like storing data decentralized, scaling during unexpected workloads, or getting used as a medical record due to higher security precautions. The convenience of precisely assigned environment resources might also become relevant for cases where a device's hardware specification are poor. Building a system upon a container-based philosophy and enclosing components in their own environment offers a variety of design and architectural possibilities without the necessity of increasing the overall system complexity.

3.7 Open Development

When developing an *Open Specification* it is natural to build upon open technologies that are recognized as *open standards*, *open source* and *free software*; *open* in the sense of *unrestricted accessible by everybody*, and *free*

as in *freedom of use*. Advocating such a philosophy allows not only the development of implementations in a collaborative way, but also enables working transparently on the specification itself. Such an open environment makes it possible for anyone who is interested to participate or even to contribute to the project. Thus, to lower the barrier, usable and meaningful documentation is vital. Such an openness ensures the possibility of people looking into the source code and getting a picture of what the program actually does and how it works. This allows source code review to become feasible, and any security flaw thus uncovered can be fixed immediately. Furthermore, this approach allows data subjects to setup their own infrastructure and host such a system by themselves, which gains even more control over the data and increases the level of trust, instead of using a *SaaS*[1] solution that is hosted by a possibly untrusted entity. It also encourages any kind of adjustments or customization to the software in order to serve operator's own needs. Enabling an open development enables users and contributors working together and collectively improve the project in a variety of ways.

[1] Software as a Service

4

Requirement Analysis

Derived from the *Core Principles (Chapter 3)*, this chapter begins by describing the perspectives of various target groups, and continues, based on all preceding work, by formalizing the requirements, which then build the foundation of further discussions.

4.1 Target Groups Perspective

The subsequent views of a *data subject*, a software developer, and a *data consumer* describe their motivation, why they would or would not use the proposed system or participate in creating such software, and what circumstances may lead to their decision. Those statements are meant as a generalized addition to the *Scenarios* describe in *Chapter 1*.

NOTICE: The following statements are written in first-person perspective for empathizing purposes only. They do not necessarily reflect the sole perspective of the author, but should be understood as general opinions of the respective user types.

As a data subject, privacy is a major concern for me. With this software I would not only have one single place to maintain all my personal and sensitive data while carrying the actual data with me, but I would also be able to control, on a very precise level and under my conditions, which vendor has what data about me or access to that data. I, myself, can even collect and submit data to my system, completely automated, whether it's movement data, my browsing history or any other data I am creating. And since I have full control over the system I could even analyze the data about me independently. It's of no particular concern to me how exactly the *ReFlowd* works, but it needs to be reliable and I have to be able to trust it with my data. Furthermore, I like to be able to control my *ReFlowd* regardless of where I am. If somebody wants to access my data, I would like to be informed about that and would like to decide, on my own, how to proceed. Convenient setup and usage is vital, otherwise I would probably resign before I got it to work. If the *ReFlowd* can meet these demands, then I would definitely want to use it, because it maintains my data's freedom.

From a developer's perspective, the most important aspects of an open source project are, in my opinion, a friendly and cooperative community combined of users and developers, as well as a detailed specification and documentation for what already exists. I also have to be a user of the developed software, which, in general, is a valuable perspective for every developer. As such I must be able to understand the specification as well as the overall architecture and design. That is, the more sophisticated these aspects are, the more effort is required to get involved. The software can still have many components though, but the complexity of the relations between them should be simple and explicit. The barriers for participation or contribution must be as low as possible. For example, instructions on how to setup a local development environment with few external dependencies can be very helpful and time-saving. The specification can either be restrictive or more flexible in its description. It depends on the developer's personal preferences which is preferred. But it must describe very precisely how the internal interfaces between components and external interfaces should look like. This is needed in order to enable several people to work on different components at the same time. Implementations based on different languages can increase user adoption and also encourage developers from other communities

to join. Alternative implementations can serve as templates or inspirations for new ones. Those aspects not only apply to core developers but also to those who write *consumer clients*, which is a library that abstracts the consumer-system-relation and should provide effortless integration into the consumer's existing infrastructure. Here it is particularly important to provide those who are going to integrate the client with detailed instructions and documentations to prevent misunderstandings and mistakes during integration. If all those aspects, especially detailed and useful documentation and a helping community, are in place, I, as a developer, and probably many like me, would gladly contribute to this project.

As a data consumer, it would only be reasonable for me to support interoperability with a system that regulates my access to user data if my business model is not contrary to that behaviour. For example, directly monetizing the collected data won't be possible anymore, unless the data subject enables me to do so by giving me raw data. If, instead, I wanted to enrich and improve my service(s) or product(s), for example, this approach is, to me, equal to storing the data on my own. Furthermore, this strategy may indirectly reduce resources and therefore expense, because it's essentially the equivalent to the procedure of outsourcing. The amount of effort needed to integrate the required interface must be appropriate. Support from the community may be necessary. In order to not violate the constrains for *Supervised Code Execution*, a detailed description, as well as a diverse set of examples, should be available. Failing repeatedly to get this procedure to work can be very frustrating, which may discourage adoption. The most important aspects for me are reliability and trust. The possible latency between requesting data and actually obtaining them could become an issue. Therefore I have to either adjust my process design so that it's not time-critical or, if it's unavoidable, the data subject needs to be informed about such sensitivities, so she can try to adapt to this, or a data resource controlled by the data owner might not be an adequate alternative to my current approach.

4.2 Requirements

Based on the Target Groups Perspective and other previous work, the subsequent requirements shall be served as a list of features on the one hand, to get an idea of how the open specification and thus the resulting software might look like, and, on the other hand, to give an overview about priorities (can/could, may/might, should, must/have to). Other chapters may contain references to specific requirements listed below.

Architecture & Design:

***S.A.01* - Accessibility & Compatibility**
Since the internet is one of the most widely used infrastructures for data transfer and communication, it is assumed that all common platforms support underlying technologies, such as HTTP and TLS. Thus the emerging system must implement a service based on web technologies, that provides supervised access to personal data.

***S.A.02* - Portability**
All major components should be designed and communicate with each other in such a way that individual components can be relocated without compromising the system's full functionality. It must be possible to build a distributed system, which may require the placement of some components into different environments/devices/platforms.

***S.A.03* - Roles**
The system has to define two types of roles *(Chapter 1 - Terminologies)*. The first one is the *operator*, who is in control of the system and, depending on the architecture, must be at least one individual but could be more. The *operator* maintains all the data that gets provided through the system and decides which third parties get access to what data. The second type is a *consumer*. These *consumers* are external third parties that desire access to certain data about or from the *operator*. While the consumer has to interact with the system only via plain HTTP API, the operator must be provided with graphical user interfaces, possibly for multiple platforms.

***S.A.04* - Authenticity**

Since they have to rely on the data, both entities - everyone who belongs to one of the roles (S.A.03) - must be able able to verify the authenticity of the other's identity as well as of the received data, in order to be certain that this data is real and belongs to the sender. It should be possible to opt out to that level of reliability if it's not necessary, or to opt-in selectively for certain types of data. However, if one of the parties demands the other to provide such a level but the other doesn't, then the access attempt has to fail.

***S.A.05* - Availability**
When third parties are requesting data, it's very likely that those procedures are triggered automatically or at least machine-supported. Hence those requests can arrive at the *ReFlowd* at any point in time. Therefore the *ReFlowd*, or at least parts of it, should be available all the time. Even if the request can't be processed completely, the system is still able to inform the *data subject* about that event; somewhat like an answering machine. This also enhances the *ReFlowd* as a serious and reliable data source. It also relates to the topic of *failure safety and redundancy*.

***S.A.06* - Communication**
To elaborate on S.A.01 and S.A.02 and ensure internal interoperability, all communication between components has to happen on HTTPS, in case they don't run in the same environment.

Persistence:

***S.P.01* - Data Outflow**
Data may leave the system only if it's absolutely necessary and no other option exists to retain the goal of the related process. But if data still needs to be transferred, none, other than the data consumer, must be able to access that data. Confidentiality has to be preserved at all costs.

***S.P.02* - Data Relationship**
Data structures and data models must show high flexibility and may not consist of strong relations and serration. If the syntax of the data representation also provides structural elements then it should be utilized to also embed semantics.

S.P.03 **- Schema and Structure**
The operator can create new data types (based on a schema) in order to extend the capabilities of the data API. Structures and schemas can change over time (S.P.04). Every dataset and data item has to relate to a corresponding type, whether it's a simple type (string, integer, boolean, etc.) or a structured composition based on a schema.

S.P.04 **- Write**
Primarily the operator is the one who has permissions to add, change, or remove data. This is done either by using the appropriate forms provided by a graphical user interface or with the help of other import mechanisms. The latter could be enabled by (A) supporting the upload of files that containing supported formats, (B) data APIs restricted to the operator or (C) defining an external resource reachable via http (e.g. *RESTful URL*) in order to (semi-)automate further ongoing data imports from multiple data resources (e.g. IoT, browser plugin). Additionally, it might be possible in the future to allow *data consumers* create data to flow back into the operator's system, after she is certain that this data is valid and useful for her.

S.P.05 **- Data Redundancy**
Providing and managing data is the core task here. Hence the system needs to make backups or at least provide mechanisms and tools for the *operator* to do that. Different strategies are conceivable but have to respect related requirements (S.P.01, S.A.03) and specific environment conditions though. The least feasible solution would be a manual backup only allowed by the *operator*.

S.P.06 **- Data Precision**
As stated in the previous chapter about *Data as a Product*, the level of precision of data has a significant impact on the *data subject's* privacy. Therefore the *data subjects* must be provided with the ability to configure how precise certain data should be when it is provided to *data consumers*. Those adjustments must have no impact on the actual stored data. That is, adjustments have to be made after the data is fetched but before it is further processed.

Interfaces:

***S.I.01* - Documentation**
The interfaces from all components have to be documented and well specified so that the components themselves could be individually replaced without any impact to the rest of the system. This also involves comprehensive information on how to communicate and what endpoints are provided, including required arguments and result structure.

***S.I.02* - External Data Query**
Data consumer can request a schema in order to know how the response data will actually look like, since certain parts of the data structure might change over time (see S.P.03, S.P.04). In order to check if the access request is permitted, the system first parses and validates the query, and eventually proceeds to execution. When querying data from the system, the *data consumer* might be required to provide a schema, which should force him to be as precise as possible about what data exactly needs to be accessed. In addition to that, the consuming entity must provide some *meaningful* text, describing the purpose of the requested data. He should not be allowed to place wildcard selectors for sets of data into the query. Instead he must always define a more specific filter, or a maximum number of items if the query retrieves more then one element.

***S.I.03* - Formats**
When components communicate between each other or interactions with the system from the outside take place, all data send back and forth should be serialized/structured in a JSON or JSON-like structure.

Graphical User Interface:

***P.VIU.01* - Responsive user interface**
The graphical user interface has to be responsive to the available space, in order to acknowledge the diverse market of available screen sizes.

***P.VIU.02* - Platform support**
All graphical user interfaces must be implemented based on web technologies, which are then provided by a server and available on any system that comes with a modern browser. To enable additional features and deeper integration

with the surrounding environment, it is recommended - at least for mobile devices - to build user interfaces upon natively supported technologies, such as *Swift* and *Java*. The operator would benefit from capabilities such as *Push Notifications* and storing data on that device.

***P.VIU.03* - Permission Profiles**
The operator should be capable of filtering, sorting and searching through a list of *permission profiles*; for a better administration experience and to easily find certain entries in a continuously growing list of profiles.

***P.VIU.04* - Access History** The operator must be provided with a list of all past *permission and access requests*, in order to monitor who is accessing what data and when, and thus being capable of evaluating and eventually stopping certain access and data usage. Such tool should have filter, search and sort capabilities. It is built upon and therefore requires the access logging (P.B.01) functionality.

Interactions:

***P.I.01* - Effort**
Common interactions for use cases such as changing *profile data*, importing datasets or managing *permission requests*, have to require as little effort as possible. This means short UI response time on the one hand and as few single input and interaction steps as possible to complete a task. Given the circumstances (e.g. distributed architecture, involvement of mobile devices), the *permission request review* and *permission profile creation* might become a special challenge to hide complexity.

***P.I.02* - Design**
Graphical user interfaces must be designed and structured in such a way that is is highly intuitive for the user to operate. Thus, it is important, for example, to use meaningful icons and appropriate labels. This also refers to a flat and not crammed menu navigation. Context related interaction elements should be positioned within the area related to that context.

***P.I.03* - Notifications**
The user should be notified about every interaction with the system originated by a third party immediately after its occurrence, and she has to be

notified, at least, about every *permission request*. This behaviour should be configurable; depending on the *permission type* and on every *permission profile*. Regardless of the configuration, notifications themselves must show up decently and pending user interactions must be indicated in the user interface. Notifications do not necessarily require a reaction by the operator.

***P.I.04* - Permission Request & Review**
A process involving data transaction must be always initiated by the data subjects. So, before a *data consumer* is able to access data, first the *operator* needs to *invite* that specific third party by either telling him (per URL) where to register or asking him to provide all information required for a registration process upfront encoded in QR-Code. Both solutions need a secure channel for transport, which refers to *TLS*. The latter has to be ensured by the applying third party alone, whereas the first option requires the *ReFlowd* to provide a TLS-supporting endpoint as well. After a successful registration, the consumer can submit a *request permission*, which has to include information about the *consumer*, what data he wants to access, for what purpose, and how long or how often the data needs to be accessed. The operator then reviews this information and creates a *permission profile* based on that information. A very important attribute in such a profile has to be the definition of when this permission expires. The operator should be able to decide between three *permission types*:

- *one-time-only*
- *expires-on-date*
- *until-further-notice* After creating the profile, the *data consumer* must be provided with a response, which should contain the review result and the determined *permission type*.

***P.I.05* - Templating**
The operator should be able to create templates for *permission profiles* and *permission rules* in order to apply a set of configurations in advance before a *permission request* arrives and reduce recurring redundant configurations and interactions.

Behaviour:

***P.B.01* - Access Logging**
All interactions and changes in the persistence layer should be logged. All data requests must be logged. Such a log is the foundation of the *access history*, so that the user is able to keep track of occurring accesses and look up past ones.

***P.B.02* - Real time**
Real time communication might be essential for time-critical data transaction. Hence graphical user interfaces should be communicating with the server through an ongoing connection to enable real time support. Consider the following example scenario: a permission request is reviewed on a mobile device, but the notification indicator in the desktop browser still reflects 'pending'. If just one of the user interfaces does not have real-time capabilities but all the others do, the user interface might get into an undefined state presenting the user with wrong information, which will decrease user experience dramatically. This means, either all user interfaces have to provide real-time feedback or none.

5

Design Discussion

The following chapter contains the greater part of the project's conceptual work. It documents the processes of a variety of design decisions, examines possible issues emerging alongside, and discusses different solutions obtained from several perspectives in order to evaluate their advantages and disadvantages. Not every issue gets its deserved room, but major aspects will be addressed.

The end of every subchapter includes a section containing a summary of conclusions based on the prior discussions related to that topic of that subchapter.

5.1 Authentication

Following the *Requirements* in *Chapter 4*, the system has to support two roles (S.A.03). Only one of these roles is assigned to an entity, hence entities that are trying to authenticate to the system might have different intentions depending on the characteristics of those roles. When inheriting the *operator role*, an entity gains further capabilities to interact with the system, such as

data manipulation. Another common scenario for the operator is to access the system from different devices in order to review *permission requests* in real time. Whereas a *data consumer* always uses just one origin and processes requests sequentially. Those very distinct groups of scenarios would make it possible to apply different authentication mechanisms that don't necessarily have a lot in common.

With respect to the requirements (S.A.01), the most appropriate way to communicate with the *ReFlowd* over the internet would be by using *HTTP*. Thus, to preserve confidentiality on all in- and outgoing data (S.P.01) the most convenient solution in this case is to use *HTTP* on top of the asymmetric cryptography based *TLS* (HTTPS). During connection establishment, the initial handshake requires a certificate issued and signed by a CA, which has to be provided by the server. This also ensures a reasonable level of identity authentication, rather effortless. If the certificate is not installed, it can be installed manually on the client. If the certificate is not trusted (e.g. it is self-signed), it can either be ignored or the process fails to establish a connection, depending on the client configurations. The identity verification in TLS works in both directions, which means not only the client has to verify the server's identity by checking the certificate. If the server insists, the client has to provide a certificate as well, which the server then tries to verify. Only if the outcome is positive, the connection establishment succeeds. According to the TLS specification [129] it is still optional though.

HTTP, as a comprehensive and flexible protocol, enables the use of several technologies for server-client authentication purposes. Some of them are built-in, others can simply be implemented on top of it.
Within the scope of this work, those technologies are categorized as the following types: (A) stateful and (B) stateless authentication. The first one (A) includes, for example, *Basic Access Authentication* (or *Basic Auth*) [130] and authentications based on *Cookies* [131]. Whereas the *two-way authentication* (or *mutual authentication* [132]) in TLS and authentication based on web-token are associated with the latter (B).
Basic Auth is natively provided by the *http-agent* and requires, in its original form, username and password, and a state on the server; at least when the system has to provide multitenancy. If, instead, common access restriction for certain requests would suffice, no state is required, because no

user-related data have to exist on the server. One of the most common implementations of a user-specific state is a *session* on the server, that contains one or more values representing the state and a unique identifier, with which an entity can be associated. A client has to provide that session ID in order to be provided with all session-related data hold by the server. This is typically done in a HTTP header, whether as *Basic Auth* value, a *Cookie*, which is domain-specific, or in some other custom header.
The *two-way authentication* is performed based on files containing keys and certificates, whose contents rarely change and don't typically require their state to be tracked in order to authenticate a user. Therefore this procedure is categorized as stateless. Order or origin of incoming requests have no impact on the result of the actual authentication process. The same applies to TLS features such as *Session ID Resumption* and *Session Ticket Resumption* [133], therefore they are left aside, because they serve the sole purpose of performance optimization. Similar to the *Session Ticket Resumption* [134] a web token, namely the JSON Web Token, also moves the state towards the client. A *JWT* carries all context relevant information, including possible states, and, if necessary, the token is symmetrically encrypted by the server. That is, only the server is able to see the actual data contained by JWT and therefore react accordingly.

Keeping track of one or multiple states on the server and maintaining the synchronization of involved data between server and client is, in regards to the additional required resources, expensive and far from trivial, because this pattern requires the server to be aware of all current states (sessions) and has to have them accessible at all times. A request might depend on preceding requests and their incoming order, which thus can have an impact on its response. Furthermore, those session data have to be stored from time to time. Otherwise, if the server fails to run at some point, data only existing in the memory would be gone with no chance to be recovered.
With stateless authentication, none of those aspects applies. Certificates and keys as well as web tokens are both carrying all necessary information with them. Even though stateless authentications, similar to stateful authentications, require an initial step to obtain some sort of token that is used to authenticate all subsequent requests, which can be considered a disadvantage.

Another aspect of resources are the computations based on asymmetric cryptography, which are usually slower than the ones based on symmetric cryptography [135], but since there are no timing constraints when interacting with the *ReFlowd*, regardless of whether it's external communication with data consumers or internal between components, parameters, that influence the level of security the underlying cryptographic procedures provide, can be as costly as the system resources allow them to be, thus the level of security can be increased.
So, when considering the disadvantages of stateless authentication, *public key cryptography* and web tokens are the preferred technologies for all authentication processes.

Except for *two-way authentication*, the initial step in all authentication technologies, as mentioned before, is a one-time procedure to obtains a token, which is used to authenticate all subsequent requests. This step lasts as long as the token is valid. The reason for such an approach is to increase convenience. Otherwise, the user is required to provide credentials on any request, which is, in terms of user experience, an unacceptable solution. Even though the *two-way authentication* also requires an initial step, that step is different to the one just mentioned. Instead of obtaining a token that proves the possession of the credentials, a certificate is obtained that is used to prove the identity of the communicating partner.
The step to obtain a token is commonly known as *sign in* and requires the entity to be authenticated to provide some credentials consisting of at least two parts. One part that uniquely relates to the entity but doesn't have to be private, and another part that only the entity knows or has. Typically that's a username or email address and a password or some other secret bit sequence (e.g. stored on and provided by a USB stick).
An *eID card* could possibly be used as secret (or private key) as well. Suitable use cases for such cards are (A) to let the operator login to the *ReFlowd* management tool or (B) to authorize *access requests*. How the actual login process (A) would look like partially depends on the *eID card*'s implementation, but in general, both are reasonable scenarios to utilize an *eID card*. When considering the german implementation *(nPA)* as an example, authenticating to the management tool requires either a card reader, preferably with an integrated hardware keypad, or a device able to communicate with

the RFID-chip, which would reduce the interaction duration to a minimum. Both scenarios (A and B) require the *nPA* to have the *eID* feature enabled. If a service wants to provide *nPA*-based online authentication *(eID-Service)*, which is defined as a *non-sovereign*[1] feature, it has to comply with several requirements [136] starting with applying for a permission to send a certificate signing request to a BerCA.[2] This request is sent from an *eID-Server* [137] in order to get the certificate signed, which has been previously generated on dedicated and certified hardware. This hardware is requested by the officials as part of an *eID-Server*. The key pair - re-generated and re-signed every three days - is needed to establish a connection to the *nPA*, which is then used to authenticate the owner of that *eID card*.
The described procedure appears to be highly expensive (regarding effort, hardware costs etc.), especially because every single operator would need to go through the whole process in order to support this authentication method; not mentioning the uncertainty of the official's decision on the permission application. Another approach could be to integrate an external authentication provider supporting the *nPA*, which would not only add an additional dependency, but also weaken the system. Both scenarios are fairly similar, insofar as they would use the same token (eID card) to initially authenticate to the system.

The concept of web tokens is fairly straightforward to integrate into the *ReFlowd*. But since web tokens ensure integrity and optional confidentiality only of their own carriage, not for the entire HTTP payload, the two requirements need to be addressed separately. Serving HTTP over *TLS* solves that issue though. For connections that use a web token, it should be sufficient to rely on the public HTTPS PKI. All information required for the actual authentication are provided by the token itself. However, the situation is different if *two-way authentication* is used instead. In this case, the system has to provide its own *PKI* including a Certificate Authority that issues certificates for data consumers, because not only the *endpoints* on the *ReFlowd* (server) need to be certified, all data consumers (clients) need to present a certificate as well. Only the *ReFlowd* verifies and thus determines (supervised by the *operator*) who is authorized to get access to the system. Hence the *PKI* needs to be self-contained and private in order to function

[1] in ger. "nicht hoheitlich"
[2] (german) Berechtigungszertifikate-Anbieter

independently so that only invited parties can be involved.
Referring to the statement mentioned above, data consumers also have to be able to verify the identity of the *ReFlowd*, in order to prevent man-in-the-middle attacks. Addressing this issue basically means, data consumers have to verify the certificate presented by the *ReFlowd*. This can be done in two ways. One way is by a certificate having been installed on the *ReFlowd* that is certified by a trustworthy public CA, as mentioned above. Then consumers use the CA's certificate to verify the *ReFlowd* certificate. The other way is to let the *ReFlowd* create an asymmetric key-pair including a certificate and sign it by itself. Before *consumers* are presented with the self-signed certificate of the *ReFlowd* during the initiation of the TLS connection, they already have to be aware of that certificate. That is, *consumers* need to be provided with that certificate on a private channel upfront. Otherwise the process would still be vulnerable to man-in-the-middle attacks.

In summary, if a certificate-based connection, performing a *two-way authentication*, establishes successfully, it implies that not only the identity of the requested source but also the identity originating the request is valid, and through additional features of *TLS* including integrity and confidentiality of the containing data is provided. Whereas on a token-based authentication every incoming request has to carry the token so that the system can verify and associate the request with an account. Furthermore data is not automatically encrypted and thus integrity and confidentiality is not preserved.

An advantage of token-based authentication over TLS-based *two-way authentication* is that the token can be used on multiple clients at the same time. Or an account that a token is associated with, can actually have more than one token. Whereas during the asymmetric cryptography-based *two-way authentication* the client's private key is required, which results in one-to-one relation. So if it's likely that an *operator* has several clients, the same private key has to be on those clients. A private key typically should not leave its current system or exist in multiple systems at the same time in order to prevent exposure, which any action of duplication implies. To reduce those risks, it's common practice to generate a private key at the location where it's going to be used.

An already existing specification explicitly addressing user authentication is

OpenID, which could be adapted to integrate operator as well as consumer authentication. Although it utilizes (sub)domains as entity identifiers, which is similar to how the relation between data consumer and *ReFlowd* in *two-way authentication* works, OpenID is underpinned by another motivation, which is providing decentralized authentications as a service, and that is contrary to the independent and self-contained model this project follows. Trying to adopt the standard might result in various adjustments to *OpenID* leading to an implementation that shares not much compliance to its origin, which is not the intention of a standard.

Conclusions: Based on the several requirements (S.A.03, S.A.04, S.A.06, P.I.04) and distinct advantages of the two authentication mechanisms, it is preferred to use *HTTPS* for the communication between the system and data consumers, where the system provides its own *PKI*. Whereas a token-based authentication on top of *HTTPS* and public CAs should be suitable for communication between the system and the operator, preferably based on *JSON Web Tokens*, because the session state is preserved within the token rather then having the system itself keeping track of it.
Furthermore, authenticating the operator is also doable on the TLS layer; but this approach on a web platform (browser) is rather inconvenient, if even possible, because a client certificate not only needs to be transferred to the server and probably signed before it and the corresponding private key gets installed manually by the operator. For a mobile platform this approach might be more feasible because this step can be automated during the application installation.
Addressing the requirement of consumers to verify whether the certificate presented by the *ReFlowd* can be trusted or not, both solutions, providing a self-signed certificate on a secure channel upfront, or using certificates certified by publicly trusted entities, are legitimate. However, the latter requires an automation depending on an external service that provides a new signed certificate whenever a new data consumer registers, such external dependencies should be kept to an absolute minimum.
To strengthen an authentication procedure, one or more factors are often added, such as an *eID card* or one-time password. This adds complexity to the procedure and thus increases the effort that is needed to perform a successful attack. But equally it also increases the effort to support those

factors in the first place. Using multi-factor authentication is generally valued and will be briefly noted as an optional security enhancement for the *operator role*. However detailed discussions regarding this topic are left to follow-up work on the specification.

5.2 Data Reliability

Using TLS-based authentication, as discussed in the previous section, brings the implicit advantages of confidentiality and integrity of transferred data. The techniques described therein are assigned to the different types of motivations to interact with the system and therefore to the roles and their various users, although authenticity of the actual data a *ReFlowd* provides has yet to be provided. Data reliability here refers to authentic and reliable data, which means (A) the data really represent the entity that is associated to the originating *ReFlowd* and is thus owned by that entity, and (B) the data are truthful at that moment when the data being accessed are queried from within the system.

The verification process of authorship and content validity in the context of the DE-Mail technology as described in *Chapter 2 - Standards, Specifications and related Technologies* can be valued as a negative example for preserving data reliability. Instead of declaring *end-to-end encryption* and *Qualified Electronic Signatures (QES)* mandatory, the creators of the corresponding law decided that it's sufficient to prove the author's identity based on her credentials when handing over the DE-Mail to the server, and that it's reasonable to let the provider sign the document on the DE-Mail server, and finally that this described implementation results in a legally binding document by definition of that law. The varying levels of mistakes made in conception and legislation are outlined in a statement from the CCC[3] [138], who has been consulted during the development of that law. As a consequence, DE-Mail as a technology has, in general, no relevance to this project. One aspect that can be noted though, is the concept of letting a server sign outgoing data. While in the case of DE-Mail the server is controlled by a potentially not trusted party and is typically multi-tenancy capable, the *ReFlowd*, including

[3]Chaos Computer Club e.V.

its back end, relates to one individual, who is also in control of the system. Thus, in order to provide a method to consumers, which enables them to verify if the received data is authentic while keeping the overhead for the operator to a minimum, a solution involving an automated signing procedure on the server can be reasonable.

Since the *operator* can change personal data at any point in time, this ability results in a mandatory process where, for example, a trustworthy third party verifies the reliability of the data in question. Depending on the design, that process can be in direct contrast to the discussion about the authentication system and why it should be designed in a self-contained way. If it's not required, though, to provide information to prove data reliability, this aspect won't be an issue.
The information about data reliability can be defined in a response as an optional property. Within the request the data consumer has to indicate whether the response should contain this information or not. Depending on what data is requested, the *ReFlowd* decides whether to provide proof of reliability or not. Based on the procedures that are available, the data reliability can then be verified by the data consumer.

But how would this reliability check look like? It comes down to two general steps. The first is matching the actual data involved in that request against a reference dataset. The second step is to hand over the information about the audit and its result to the data consumer. Depending on what information the consumer is provided with, it can then be used to verify that information. The data consumer decides on the value of that given information.

The following proposed methods vary in the provided level of reliability as well as in the amount of effort to support them and in the possible impact to their surrounding systems. Not all data items are necessary to test for reliability. Profile data for example are more likely to be tested, whereas consumption lists or location histories are more difficult to verify, because currently there is no reliable way to verify the origin of those datasets, and they are also not a primary focus here.

(1) **Local Verification by matching**
Probably the simplest and, at the same time, least reliable method is to just look at the existing data that are stored in the database and

match them against those data that are used to create a response.

(2) **Local Verification and signing**
An electronic ID card can serve as an authentication token for the operator, but it can also be utilized to verify the reliability of certain data. Using the german implementation *(nPA)* as an example, the *eID* feature would provide access to the owner's basic profile data, which can then be used to match against those data items that are both held by that *nPA* and affected by the *access request*, and therefore originated in the *ReFlowd*. If the result of that matching procedure is positive, the related data then gets signed with a *QES* courtesy of the data subject's *nPA*. That signature gets included in the response as well.

(3) **Remote Verification and signing**
Another method involves a trusted third party who also has the same data that need to be proved. The idea is to hand the data in question over to that party, who then tries to match against all those data items available in that context. The party also has the ability to sign data, which is what's happening, if the matching procedure has a positive outcome. Signing the whole response, or at least a replicable dataset that contains the data that were initially requested to access, is required. The party then hands everything back to the *ReFlowd* for further processing.

(4) **Recurring Certification**
The following method describes a modification of (3). Instead of signing every data access, the current state of the overall data is certified on a recurring basis. This does not necessarily mean that all data items get checked during the certification procedure. Rather, only those data items known to be reliable by the external third party are matched. This is done by either literally looking at the data or by automatically processing and matching against their own database. If that party is satisfied, a certificate for the corresponding data will be issued. This certificate contains an expiration date, which implies the consequence of going through this process again in the future, much like an issuing process of a common *Certificate Authority*. The certificate is installed

on the *ReFlowd* and served as part of a response.
Until now, the consumer can only decrypt the signature provided in the certificate, which, if it's successful, just states that the issuer has verified the truthfulness and relation between the operator and some data stored in the corresponding *ReFlowd*. Optionally, and only if the exact data items involved in the certification procedure are also part of the response, the consumer is able to completely verify the certificate, and thus the data reliability, by hashing that data on its own and matching the result against the one contained in the certificate.

Only one method per request should be used to verify data reliability, because every method can imply a different level of confidence. As described above, the response sent back to the data consumer has to indicate the method used. Based on that, the data consumer is then able to evaluate that level of confidence and can act accordingly (e.g. verify a signature).

Expanding those verification procedures is reasonable, but to keep it simple for now this aspect won't receive further attention, since the current requirement (S.A.04) is sufficiently met. It will be left to future work, however.

Conclusions: A signing procedure as part of the local verification method (2) involves private key and certificate stored on the operator's *eID card*. Every time the *ReFlowd* verifies data reliability that procedure has to be performed. Thus the *operator* is forced to interact wit the *ReFlowd*. Otherwise, the operator's private key needs to be stored somewhere within the *ReFlowd*. No matter where or how, this would potentially expose a highly confidential part of a cryptographic procedure. Not only would this reduce the overall security level of the system, it also makes every task this method is involved in vulnerable to certain attacks. Aside from that, it's highly unlikely that an *eID card* would allow extraction of the private keys it contains. This all results in increased inconvenience which is inevitable for this proposed method. The *Local Verification and signing (2)* method has the same dependencies mentioned in the previous discussion about requirements for using the (german) *eID card* as an authentication token. And since it was rejected because of those dependencies and because of the inconvenience mentioned before, that verification method ultimately is not going to be supported in the specification. Furthermore, the signing procedure based on the

QES can easily be spoofed by the data subject regardless of the matching result, because it would indeed prove that the *ReFlowd* really represents that individual (authorship of the response), but not the reliability of that data.

The *Remote Verification and signing (3)* method requires the external party to be an official authority, because no other entity has the data in question (primarily profile data), which makes them legally binding; and they are commonly trusted. The same goes for the *Recurring Certification (4)*, but while the *Remote Verification and signing* method introduces a very strong dependency to that external party, the *Recurring Certification* offers a simple, loosely linked dependency. Whose design would make it possible to obtain such a certificate manually, but also automatically. Nevertheless, both can provide a trustworthy certification.

Finally, the first method, *Local Verification by matching (1)*, which just performs a matching of two datasets against each other, that does not provide any proof to verify reliability of that. Those datasets are obtained from the same *ReFlowd* storage, but at different times; right before the request finally is sent on, though. The primary purpose of addressing the issue of data reliability rests on the data consumer's concern of accessing data that is intentionally tampered with (e.g. by the data subject), since integrity during transport is already ensured. In this light, even if the whole system would be compromised, which in that case might need more urgent addressing then ensuring data reliability, it won't mitigate the fact that the operator is the only one able to change data. Hence it provides the lowest level of reliability.

Certain fields of application of a *ReFlowd* as a data resource might already impose constraints about the level of reliability and maybe even how it can be provided. Violation of those constraints or other relevant legislation might prevent the *ReFlowd* from being put in use. Others - depending on their guidelines and business model - don't rely on a certain level of confidence. In general, data consumers are expected to already have a basic confidence in a *ReFlowd* and in the data originating there. Regardless of that, providing an indication on the reliability of data is valued as a first and important, but still not mandatory step towards a fully working feature. All of the proposed verification methods have some downsides, though, the *Recurring Certification (4)* method would be the least invasive and therewith an ade-

quate choice, even though it might only provide trust rather than verifiable proof.

A primary goal for the *ReFlowd* is to preserve all data owned by the data subject and giving her control over where the data might go; not providing sufficient proof for the data reliability. Although it is still important to provide data consumers with an information about the level of reliability, it's up to them how to rate that information and how to act on that.

5.3 Access Management

The subsequent section will discuss how several processes surrounding the topic of *data consumers accessing data on the ReFlowd* can be modeled, what consequences certain variations might have, and what issues need to be addressed.

Below, a general design is proposed of how *data consumers* get authorized and thereby are able to access the *data subject's* personal data, and how previous mentioned technologies *(Chapter 2)* can be assembled in order to meet the specified requirements *(Chapter 4)*.

Part One: consumer registration

0) The *operator* creates a new, unique URL in the system

1) **Prepare registration**; the operator has to tell the third party where and how to register as a data consumer by handing over a URL uniquely associated to the current registration process. Several things need to be noted here. First, the operator 'pulls' consumers into the system. This is the only way for a consumer to establish a relation. If consumers were able initiate this process on their own without the operator's involvement, it would be much harder for the system to detect spam or fraudulent requests. Second, handing over the URL must be done over a *secure channel*.

2) **Send registration attempt**; the third party then makes the actual

attempt to register as a *data consumer* by providing required information. With this information, an information about a feedback channel (e.g. URL) has to be included so that the system can get back to that third party. They also may contain a first *permission request*.

NOTICE: the two initial steps can be skipped if the third party would rather present the information mentioned in step 2), as a QR-Code so that the operator can obtain it and thereby is able to proceed. This approach would shortcut and hence simplify the process.

3) **Review registration attempt**; the operator gets notified about a new registration attempt, which she then has to review and decide whether or not to accept it.

4) **Create permission profile**; if a *permission request* is enclosed in the registration, it has to be reviewed as well. If it's not, the next step follows immediately. If permissions are granted, a new *permission profile* is created. Optionally, it could also be created if the permissions were refused. It's just meant for the operator to keep track of her decisions.

5) **Respond to third party**; regardless of the decision, the third party is then informed via feedback channel about the decisions and is also provided with further details required to obtain actual data.

Part Two: obtain data

0) A *successful registration* as a data consumer is required.

1) **Send request**; the data consumer sends the *access request* to the system, containing all information about what data is needed, how to process the data, and what the response should contain.

2) **Parse and check request**; after the system has received an *access request*, it first
authenticates the data consumer and checks related *permission profiles*. According to the defined *access rules*, the system decides how to proceed. Either it pauses, because it needs further attention from the

operator, or it can start to process and create the response.

3) **Compute response**; how the computation would look like mainly depends on the contents of the *access request* and also on what a *permission profile* determines (see 'types of access requests' below).

4) **Respond to consumer**; hand over the computed response back to the requester by proceeding with one of the two following options. Either the system responds with the current status of the process and where the consumer can find the demanded data, or the consumer includes a callback URL, which the system has to invoke with the demanded response.

With respect to the requirements (S.P.01), personal data should not leak into the outside. To tackle this issue, the following three types of *access requests* are defined, starting with the most sufficient solution:

a) **Supervised Code Execution**; *access requests* additionally come with an executable program - binary or source code - potentially including information about provisioning. After the required data is retrieved from the storage, the program gets invoked with the data locally on the system but within a completely separated environment *(sandbox)*. The result of that invocation gets returned to the outer system (*ReFlowd*).
b) **Data DRM**[4]; after data is retrieved from the storage it gets encrypted. The cipher is included in the response. Upfront, data consumers are equipped with a small program that can connect to the *ReFlowd* and has to wrap the consumer's own software that is meant to process the requested data. Now, when consumers receive the response, the program needs to get invoked with the cipher, so that, by priorly fetching the key from the *ReFlowd*, the cipher can be decrypted from within the invocation. Thus the data is made available to the wrapped software and only during runtime. After the invocation has finished, the program needs to propagate the results that are returned by the software back to the outer environment.
c) **Plain Forwarding [default]**; retrieve data from the storage, quick-checking the result, adding an expiration date and forwarding it di-

[4] *Digital Rights Management* - set of technologies, that are used to control access to data or content that is restricted in certain ways (e.g. content provided by video streaming)

rectly into response.

In case the data consumer provides no preferred access type, a fallback type must be applied so that the data won't leave the system unless it's absolutely necessary to pursue the goal of the access request. The overall confidentiality of all personal data is still preserved, however, because all communications to and from the *ReFlowd* are happening over HTTPS, which means the data is encrypted during the transport.

The concept of authorizing a data consumer for the ability to access personal data is fairly trivial. During (or after) the *registration*, consumers have to provide detailed information about their intentions so that the operator is confident about the required permissions when reviewing them. The created *permission profile* reflects the result of that review. Such a permission profile defines what data items are permitted to access, how often they can be accessed, how long those permissions last and when responded data expires. The latter is defined as *permission type* and is either one of the following:

one-time-only access permissions are hereby granted for just a single *access request* (tolerating certain errors regarding the communication layer)

expires-on-date access permissions are hereby granted until the defined point in time has arrived

until-further-notice access permissions are hereby granted until the *permission type* has changed or the *permission profile* has been deleted

NOTICE: The default permission type should be configurable. The operator can change all permission profiles at any point in time.

Among other information, an access request contains the *data query* that shows very precisely what data items are affected by that request. So if an *access request* arrives at the *ReFlowd*, assuming the data consumer has been authenticated sufficiently, the system (0) searches for a permission profile that corresponds to the data consumer and the requested data items. If it fails to find one, the access request is refused. But if it does, then it checks (1) if the permission type suffices at that moment and (2) if the query only contains data items that are enabled in the profile as well. Here, the order matters because it is imaginable that the operation behind (1) is

less complex then operation (2). Running (1) before (2) can result in a faster response-time, if operation (1) already results negative. If all operations have a positive result, access is granted.

As stated in the section about data reliability *(Chapter 5)*, the data subject is able to add, change, or remove all her data or even the permission profiles at any point in time. This raises the question of how to solve the situation were access requests are being processed while those changes are happening and might affect the result of those requests. The first and simplest approach would be to not address this issue at all, but that would be unreasonable because providing data to the consumer normally means for the data subject to get something in return or to somehow benefit from that. So that approach is not an option. Using a failure of reliability verifications as a mechanism to re-request data won't work either in that case, because it would be based on a wrong assumption, since that failure can have multiple causes, not only the issue here in question.
A stateless solution seems to not fit due to the time-related dependency. So the only currently perceivable way is to keep track of all momentarily processing or pending *access requests* to detect those who are affected by the changes so that each of them can be aborted and processed again. Here it is important to determine the right moment, when all changes are done, otherwise the system might end up restarting those computations repeatedly within a short amount of time. The described issue relates to both *personal data* and *permission profiles* because either can impact the response that is returned to the data consumer. Furthermore, it needs to be ensured that only after *permission requests* are reviewed and *permission profiles* are created, data consumers receive their credentials or a notification to get started.

It is up to data consumers to decide which data they are requesting to access, but how do they know what data can be requested? The only option is to expose information about data availability (S.I.02), which can be done in a variety of ways. First, that information can be made publicly available via URL, providing a machine-readable format so that information can be processed automatically by consumers. It is also feasible to restrict that access to registered consumers only, in order to prevent that information from being crawled. They could be valuable as meta data and therefore used in undesired processing that could raise privacy concerns. It is imaginable to

allow operators to restrict the access to such availability information on an individual consumer-basis or system-wide, and furthermore, to set default configurations for this behaviour. Depending on those configurations a request might fail, thus the requester needs to be provided with meaningful errors. Http error codes [139] might be a sufficient fit for this purpose.

An already standardized way to implement authorization is the OAuth Specification. And since the TLS layer is already in place to handle authentication, the choice would be to use version 2 of the standard, because it relies on HTTPS. Only two of the four *grant types* provided by OAuth would match with the process design introduced earlier. The types are `password` and `client_credentials`, which basically require identifier(s) and secret or credentials to directly obtain a `token`. The other two types define additional steps and interactions involving consumer and operator before getting the `token` because they are intended to be used for processes involving untrusted environments (e.g. browser, third party apps). Aside from not fitting into such scenarios it would also make the proposed process undesirably more complicated. Although the proposal includes user interactions like selecting and confirming requested permissions.
According to the documentations [140] [141], both OAuth versions (1.0a and 2) require the client (here *data consumer*) to register to the authorization server upfront (to obtain a `client_id`), before initializing the authorization process. However, as stated earlier, the concept of the data subject 'pulling' a data consumer towards the *ReFlowd* is preferred over letting data consumers try to 'push' themselves towards the system. The reason to prevent undesired registration attempts is that they all have to be reviewed by the data subject. Furthermore, it is not within the scope of the OAuth Specification to define how this should be accomplished. Thus, such step needs to be added in addition to an entire OAuth-Flow, which might cause otherwise avoidable overhead in user interactions. Moreover, the proposed design does not include that specific registration process either. Instead, this process is not needed at all because, according to the proposal, client identification happens implicitly as a result of how the resource owner (operator) obtains the registration request from the client (see *Part One: consumer registration, step 0 to 2*).
Further investigations show that the semantic of an `access_token` from the

perspective of a resource server consists of authentication *(Does this token exist?)* and authorization *(Is this token valid and what does it permit?)*. Those aspects are in part already provided by the proposed way of using the TLS layer. Because every data consumer has its own endpoint to connect with the *ReFlowd* and the certificate used by the consumer is signed by a signature only used for that endpoint. This means the consumer is already authenticated when the TLS connection has been successfully established. And since *permission profiles* relate to a specific endpoint, it would make providing an `access_token` obsolete.
To summarize, implementing OAuth would introduce several mechanisms that otherwise can be provided by the combination of *two-way authentication* in TLS, dedicated endpoints and certification.

Conclusions: In the preceding text, various solutions were examined, resulting in the following three available options to provide authorization and access management:

a) OAuth 1.0a and HTTP
b) OAuth 2 and HTTPS (public Certification and PKI)
c) HTTP over TLS with *two-way authentication*, private PKI, subdomains as dedicated endpoints

The solutions a) and b) require an extra step in which data consumers need to register themselves at the *ReFlowd*. This already must be done on a secure channel to prevent man-in-the-middle attacks. Furthermore, in that step, option a) obtain a symmetric key for creating signatures used to ensure confidentiality and integrity in subsequent steps. All those cryptographic procedures need to be adopted when implementing the here proposed specification and also when interacting with those implementations. While this can cause much more harm, it is proposed to leave as many of these sensitive parts as possible to existing implementations who have already proven themselves. Thus HTTPS is mandatory, which makes b) more suitable over a), because it's also more flexible and easier to implement.
Solution c) moves the complete authentication procedure to a different layer. Hence, it results in separating authentication and authorization from each other. This opens the authorization design up to other implementations that might be more suitable for certain *data types*. In addition, it would require

little effort to support the case where multiple data consumers share the same *endpoint* and thereby the same *permission profiles*.
Combining b) and c) would result in significant redundancy, since both solutions have much overlap in their provided features, even though b) aims to be a framework for authorization. The process description in the beginning of this section is used as the foundation of *access management* in the *ReFlowd*. Implementing OAuth based on this design would leave nothing from the framework but a simple request returning an identifier associated with permissions. And even these identifiers are obsolete when combining TLS with dedicated consumer-specific endpoints, as c) states. So there is not much benefit in using OAuth, other than developers might be familiar with the API. This can be addressed by a detailed specification for this project, hence c) is preferred over b). In the end, the only suitable use case from OAuth this work would consists of just a request that obtains a token after authenticating with the provided credentials. OAuth only provides a framework for how to authorize third parties to access external resources, leaving the procedure of how to actually verify those access attempts up to its implementers [142] [143]. In the context of this project OAuth does not comply with the rest of the design aspects.

How the first steps of a registration look like, is up to the consumer, though the option involving a QR-Code might result in a nicer user experience from the perspective of a data subject. In any case, a secure channel is vital.

When accessing personal data, preventing those data from leakage is almost impossible, which originates in the nature of digital data being able to be effortlessly copied. Nevertheless, it is possible to make it much more difficult, so that it becomes inefficient to bypass those mechanisms. At the same time it requires some effort to establish, run, and maintain the infrastructure needed for those mechanisms. In case of the *Data DRM* proposal, that effort is not proportionate, because it requires additional infrastructure, interfaces, and cryptographic procedures and therefore introduces new attack scenarios. For now the only approach being considered is *Supervised Code Execution*, aside from defaulting to simple forwarding.
When implementing this approach, two directions might need to be considered. Alongside the executable program, data consumers either provide all dependencies so that everything is bundled up, or don't provide any depen-

dencies at all. The latter is preferred, because it reduces the amount of potentially malicious, flawed, or needless components, so that the data subject, supported by her *ReFlowd*, has more supervising capabilities and thus more control over her personal data.
Since the overall goal here is to prevent the data subject from losing control over her data, it is conceivable that certain categories of personal data, representing a higher level of sensitivity, require a minimum viable *access type*. If the data consumer does not comply, access will be refused.
Also, depending on which category the personal data relates to, the *ReFlowd* might somehow anonymize certain types of data, if it is even capable of doing so, because the consumer at least supposedly knows what individual is behind the *ReFlowd* it is currently interacting with. The field of *data anonymization* is a large research area on its own, which recently started to gain a lot of traction due to emerging privacy concerns about *Big Data*. Thus it will be left for future work.

5.4 Data

The core task of a *ReFlowd* is providing data, *Personal Data*, which is the digital manifestation of an individual, a person. One party creates the data, another one obtains and processes it. Thus, both need to agree, or at least need to know, how that data looks like, how is it structured, and what their semantics are. The following section is intended to discuss different technologies used to create queries that obtain desired data items. Further on, it describes some basic data types and schemas, that might be useful in the context of *Personal Data* and also for previously introduced scenarios *(Chapter 1)*.

First of all, to address the need of portability, which has to be satisfied by those components that are storing and providing *Personal Data*, it is essential to abstract the actual storage from how it gets accessed. This makes it possible to relocate that storage onto other platforms and environments. Thereby the *Personal Data Storage* itself becomes platform-agnostic from an outside perspective, in other words, portable. In order to reduce possible issues related to unsupported communication protocols it might be reason-

able to enforce HTTP - over TLS, if they don't share the same environment - even if the storage therefore requires an additional driver or proxy layer, like, for example, a mobile app.

Considered technologies are, *GraphQL* or the *SPARQL*, which is part of the *Semantic Web Suite*. Both are query languages underpinned by the concept of a graph. This means, relations between data items are embedded within the data structure itself. That means, in terms of a graph, relations are *edges* and data items are *nodes*. As a consequence, the structure of a query itself reappears in its result, which means the originator of that query knows exactly what to expect for the response. Therefore it's not necessary to provide any additional information about how to handle and interpret the response data. The code examples (Code 01 and Code 02, Code 03 and Code 04) give a first impression of how it might look like when a consumer obtains the name of the data subject and a bank account of hers that supports online payment.

Without going into great detail, the syntax of the SPARQL query (Code 01) already shows its nature of decentralization. This aspect at the same time introduces additional external dependencies. Because the query language itself has no concept of schemas or any kind of semantic, it needs to be made aware of them. SPARQL queries typically return XML which then can be rendered into (HTML) tables. JSON and RDF are also supported. The reason for performing two queries in the example instead of just one is because the result would otherwise have returned multiple 'rows' with redundant data if more then one bank account supported online payment; the bank account data would be different, but the profile information would be repeated.
The same phenomenon occurs when requesting a *RESTful* API to obtain that information, because the returned data structure is defined by the responder, not the requester.

The GraphQL query syntax (Code 03) compared with its result (Code 04) shows a remarkable resemblance. By defining `paymentMethod` as an argument, the resolver for `bankAccounts` then implements an instruction to match the value of that argument (`'online-service'`) against the whole set. GraphQL's server then knows from which resources the data in question

need to be pulled and how they need to be aggregated.
While SPARQL has a full-featured query language syntax including all sorts of controls (e.g. aggregation, operators, nested queries etc.), GraphQL's syntax instead is more rudimentary because all of its functions and logic have been abandoned from the language itself and put into the server. With this concept of separation it is straightforward to validate queries because it essentially means matching against types.
Both query languages share a comprehensive understanding of a type-system that encourages the creation of all kinds of data types. While in GraphQL common schemas still need to be created, SPARQL already provides a reasonable amount of vocabulary [144]. However, when comparing the results of both languages, some distinctions appear. While in GraphQL the characteristics of graph-structured data remain, SPARQL's output is missing a certain level of depth. The reason for that originates in the design of the query language and its syntax. SPARQL is able to notice implicit relations between data items, though its query language is not capable of grabbing and presenting them. Thus the result (Code 02) only consists of two dimensions.

It is crucial for the *ReFlowd* to provide the data subject with abilities to create her own data types and schemas (S.P.03). Thereby she is enabled to serve data items according to her own needs and terms. In order to interact with their customers or users, data consumers might as well develop and provide schemas for their requests. This can help data subjects to speed up the process of permission granting and to more easily understand what data items are affected. Data types and schemas are the key to validate incoming and outgoing data. If data violates the underlying schema or no appropriate schema exists, the data transfer fails.
Other missing data types could be developed by a community, because not every data subject might be capable of modeling her own data types. Thus everyone can benefit from that effort taken by a few. As a result, the ones that are widely used might then become de facto standards. Moreover, it's also possible that several data types will emerge, that are based on common standards, for example *medical record* [145], *point of interest* [146] or *bank transfer* [147]. With that approach those data types can be viewed as something like a plugin or add-on to the *ReFlowd* ecosystem.

In order to avoid confusion about the differences between types and schemas

and to simplify their relations, the following two definitions are henceforth being used. A (data) *type* is the superior term; hence refers to both of them.

[**primitive**] most basic or atomic data type; defined as either *String*, *Boolean*, *Integer*, *Float* or *Nil (null)*

[**struct**] combines multiple types in order to define more complex data types; typically composed of *primitives*, but can consist of other structs as well

Based on these two concepts almost any imaginable data type can be modeled. A selection of such types can be found in the list of suggested structs (List 01), whereas an extract of (sub-)categories that might be useful in a *ReFlowd* are specified in a list of data categories (List 02). Additional examples for *structs* are a *data subject's* profile (Code 05), a contact (Code 06) and bare position information (Code 07). All those examples and lists are only to be understood as a starting point that should cover basic scenarios as well as to give a first impression of what data types a *ReFlowd* could provide.

List 01: Suggestions for useful structs

- Address
- Contact
- Location
  - Country
  - City
  - Position
- Media
  - Audio
  - Video
  - Photo
- Organisation
- Date
- TimeRange
- Language
- Diseases

List 02: relevant (sub-)categories of data

- Finance
  - Income
  - Bank transfers
- Shopping history
- Product
- Things/Objects
- Media consumption
  - Music playlist
  - Watchlist
- Favorites/Interests
  - Music genres
  - Songs
  - Movies
  - Books
  - Travel destinations
  - Topics
- Curriculum vitae (CV)
  - Educational level
  - Visited schools
- Visited …
  - points of interest
  - countries
  - websites/URLs (browser history)
- Units (measurements)
- Organisations
  - Company
  - Bank
  - …
- Medical/Health Record
  - Diseases
  - Treatments
  - Visits to the doctor
  - Medication

The available *primitives* mainly depend on those who are supported by the query language itself. In this case, all *primitives* mentioned above are sup-

ported by *SPARQL* [148] and *GraphQL* [149]. When choosing a database system it has to be ensured that either the system already supports the required *primitives* or they can be emulated somehow with minimal drawbacks. When modelling relations between data items, one can use, for example, keys (or identifiers) to make references, or additional syntactical tools like *lists* (or arrays) and maps (or objects). Those tools facilitate readability so that relations are almost intuitively observable, hence they should be enforced. Whereas another known concept in data modelling, called *inheritance*, isn't required but could help to reason about certain *structs* and their representations. It might add complexity, though.

Apart from the subject's Personal Data, other information and data must be persisted as well. This includes for example:

- Application data
  - Templates (P.I.05)
  - Permission profiles (incl. versioning)
  - Consumer information
  - Meta data
  - Notifications
  - States
  - Tokes
  - Access logs
- Files
  - Cryptographic keys
  - Executable program
  - Container images
  - Configurations
  - User interfaces
  - Documentations

The list revels that not only a database system is needed to satisfy the requirements, but the environments filesystem might need to be utilized as well. This leads to the question of what requirements a database system has to satisfy. But first, it is pivotal to distinguish between the needs of a *Personal Data Storage (PDS)* and a general *Persistence Layer (PL)* for the system's backend.

Table 5.1: Features that a database system has to provide in order to be suitable for either of the two given purposes

Characteristic	Personal Data Storage	Persistence Layer
portable	-	-
advanced user & permission management	-	**X**
document-oriented	**X**	**X**
support common primitives	**X**	**X**
replication	-	**X**
efficient binary storage and serialization	**X**	**X**
high performance	-	**X**
operations and transactions	**X**	**X**
background optimization	-	**X**
version control	-	-

Although most of the characteristics (in Table 5.1) are self explanatory, certain aspects need to be commented on. First, portability, an important requirement (S.A.02), that is oddly not marked in Table 5.1. This is because of the priorly introduced concept of abstracting the *Personal Data Storage* with an additional query language. Therefore the access to the *PDS* becomes platform-agnostic. Whereas the database system storing that data can be implemented with respect to the requirements while considering the environment constraints at the same time.
Basic permission management should suffice the *PDS*, since it's not accessed in multiple ways. It only relates to the query language abstraction. Data and especially its structure is expected to be highly fluctuant (S.P.02), thus advantages of relational databases (e.g. schema-oriented and -optimized) would instead harm the performance and flexibility, as they are not primarily designed to handle schema changes. Database systems whose storage engine is built upon a document-oriented approach are a more suitable choice.
Replication can be utilized for horizontal scaling, federation, or backups (S.P.05). The focus here is on the latter, because without *PL* the *ReFlowd* wont be able to function. In case of irreversible data loss, the whole system

state is gone, which then has to be reconfigured and reproduced from the ground up. Such effort can be spared by introducing a reliable backup strategy. With the *PDS*, on the other hand, replication is not necessary, but, in order to prevent data loss, still needs to be addressed. Therefore every database system that might be used for the *PDS* must provide a mechanism to backup or at least to export the data, which can be triggered and obtained through the operator's management tool. Another approach imaginable could be to not only store the actual data written to the *PDS* but also to save all queries in a chronological order that have somehow changed the data. Therefore the current state can be restored just by running those queries against the *PDS*. It is reasonable to store the queries of the abstracted query language not the ones the query language is transformed into. If a mobile device is part of the *ReFlowd*, another approach for the operator could be to perform regular device backups. Those strategies are all just initial thoughts which might be sufficient only as a starting point. Other solutions are imaginable. Though, elaborating on those is beyond the scope of this work.

Depending on the technologies that are being used, it might be necessary from a conceptional perspective to split the *PL* into two parts. One part is a database system and the other is represented by the environment's filesystem. It might not be an alternative when it comes to key files, which are typically accessed as files, or configuration files for certain technologies. In any case, both *PL* and *PDS*, have to be able to store files of any kind, which is required, for instance, to support the scenario of medical records. File size restriction should be mandatory though, because the *ReFlowd* has no intention to replace existing *file hosting* solutions.

Being able to undo changes to certain data items or to review the change-history of those data can be very useful; not only when those changes were persisted mistakenly. This behaviour might not be necessary for all data, especially when it comes to application configuration or logs. Also, not every operator might require these features. Because of this and because database systems with no alternative might not be able to provide this capability, it's not required by either *PDS* or *PL*. If a history is not natively supported but still desired, it has to be considered if, for example, frequent backups would suffice or if a implementation on the application-level is required.

Before serving data, it needs to be at first inserted into the *PDS*. This can be done in three different ways:

a) The data subject uses forms provided by the graphical user interface to insert data about her, for example her profile information (Code 05). This data is then submitted into the *ReFlowd* which takes care of storing it.
b) The data subject is in possession of file(s) or string(s) containing a data format that is supported by the system. The graphical user interface provides a mechanism to either upload the file(s) or insert the string(s). Thereby the data is then sent into the system. If these raw data are not self-explainatory to the system, the data subject has to provide more information on the context of those data.
c) Third party software, for example a browser plugin, is used to provide the *ReFlowd* with data; in this case, a browser history. This software uses a restricted API provided by the *ReFlowd* to let data flow into the system.

These three concepts, especially b) and c), are required to be inspected for malicious content and extensively validated against existing *structs*. Only if these checks do not fail, the submitted data can be stored. For scenario b), the data subject needs to be asked to review the imported data to make sure everything is as expected. When enabling third party software to submit data, appropriate authentication and permission mechanisms must be in place. That software is classed like all other operator front ends, but without permissions to obtain data. According to the discussion on authentication *(Chapter 5)*, these mechanisms thus can be implemented through *JWT*.

Conclusions: In order to gain flexibility in choosing technology and location for the *Personal Data Storage*, it is logical to abstract the interface from the database system. Introducing a separate query language is proposed as a reasonable approach. It can be chosen between two suggested query languages, *GraphQL* or *SPARQL*. Both provide the necessary features required to be integrated with a distributed system; *SPARQL*, with its concepts of URIs as identifiers and resources, and *GraphQL*, with its separation of query definition and execution. This also effects the process of query validation, which is much harder to do for *SPARQL*, because its syntax is more flexible

and allows some shorthands, therefore the possibilities are far too numerous. In general *SPARQL's* syntax is harder to comprehend compared to *GraphQL*. And even though the result of both languages is formatted in JSON (S.I.03[5]), only *GraphQL* preserves all the relations in the output, which are already embedded in the query syntax, whereas a *SPARQL*-based or RESTful API requires multiple round trips and additional effort to merge the data into the desired format. As a result, *GraphQL* (and its implementations) is the query language of choice for this project, which is going to serve as the data API for consumers, too.

When it comes to creating new structs, engaging a user community can compensate the lack of certain types. Examples for a potential starting point of *ReFlowd*-supported data types were showed before. Data Modelling in general is a large research field to its own. With regards to the *ReFlowd* it needs much more attention, though it's beyond the scope of this work. The basic approaches within this section should only be viewed as an introduction that gives an outlook of how it's imagined.

5.5 Architecture

By taking all requirements in previous sections and their discussions into account, this section has the purpose of figuring out how all the different concepts and conclusions from this chapter can fit together in an overall system architecture that is organized in either a distributed or a centralized manner. The outcome of this section should not impact results or conclusions from other topics related to the behaviour of the system's interfaces from a user point of view.

The foundation of this project is a server-client Architecture, which is chosen for providing availability (S.A.05) and separating concerns [150]. Such a distributed system provides various locations to place these concerns, which are, in fact, different environments with different properties. Those combinations of locations and environment properties are herein after called *platforms*. To further describe these *platforms*, characteristics such as architectural layer

[5] si03

and access possibilities to its host environment are taken into account. The resulting three *platform* types are shown in Table 5.2. All platforms are considered to be controlled by the data subject. Here, *trusted* refers to how likely it is that the host environment, and maybe even the platform, is unintentionally accessed by third parties, whereas *host access* describes the possibilities to access the host environment from within the platform (e.g. filesystem to store data).

Table 5.2: Platform types for the different components of the *ReFlowd* architecture

Platform Type	trusted	Host Access	Layer	Purpose
Server	yes	yes	back end	- business logic - third-party interfaces - data storage
Mobile	cond.	yes	front end	- based on host-specific native technologies - graphical user interface - data storage
Web	no	rest.	front end	- browser - based on web technologies - graphical user interface

The next step is to determine all the components that are required in order to cover most of the defined use cases. The conglomeration below highlights all major components, including information about which platforms they could be in, as well as further details about their major task(s), underlying technologies, and relation(s) to each other.

Web Server

Platform: Server

Tasks:

- serve web-based user interface(s)
- handle all in- & outgoing traffic (outmost layer)
- revers proxying certain traffic to different components
- en- & decrypt HTTPS traffic, thus authenticate consumers
- load balancing (if necessary)
- web notification
- spam protection

Relations:

- operator, consumers, third parties
- any front end platform (Mobile, Web)

Technologies:

- HTTP
- TLS
- WebSockets

Permission Manager

Platform: Server

Tasks:

- creating *permission profiles*
- access verification
- examine data queries
- queue *consumer* requests

Relations:

- Storage Connector
- Notification Infrastructure
- Persistence Layer
- Tracker
- Code Execution Environment

Technologies:

- any modern language/framework capable of parallel computing

PKI

Platform: Server

Tasks:

- CA
- manage keys and certificates per *endpoint*
- obtain trusted certificates from public CAs

Relations:

- Web Server

Technologies:

- X.509
- ACME [151] (Let's Encrypt)

Storage Connector

Platform: Server

Tasks:

- abstracts to system agnostic query language
- queries personal data, regardless of where it's located

Relations:

- Personal Data Storage

Technologies:

- driver for used database

Operator API

Platform: Server

Tasks:

- authenticates operator
- writes personal data through Storage Connector

- provides relevant data, such as history
- system configuration
- automated data inflow

Relations:

- Storage Connector
- Notification Infrastructure
- PKI
- Tracker
- Permission Manager

Technologies:

- JWT

Code Execution Environment

Platform: Server

Tasks:

- isolated runtime (sandbox) for computations/programs provided by consumers
- restrict interaction with outer environment to absolute minimum (e.g. no shared filesystem or network)
- one-time use
- monitor sandbox during computation
- examine and test the provided software

Relations:

- Permission Manager

Technologies:

- Hypervisor
- Container (OCI)

Tracker

Platform: Server

Tasks:

- log all changes made with *Storage Connector*
- tracks states for ongoing consumer requests
- log all *access requests*

Relations:

- Persistence Layer

Technologies:

- any modern language/framework capable of parallel computing

Personal Data Storage

Platform: Server, Mobile

Tasks:

- stores the *operator's* personal data

Relations:

- Storage Connector

Technologies:

- non relational database
- depending on host environment

Persistence Layer

Platform: Server

Tasks:

- stores Permission Profiles, History, Tokens, Configurations and other application data
- cache runtime data and information

- holds keys

Relations:

- Operator API
- Permission Manager

Technologies:

- non relational database
- Filesystem

Notification Infrastructure

Platform: Server

Tasks:

- notifies about everything that needs operator's approval (e.g. new registrations, new *permission requests*)

Relations:

- Web Server

Technologies:

- WebSockets for web UIs via local Web Server
- mobile device manufacturer's Push Notification Service for mobile apps

User Interface

Platform: Mobile, Web

Tasks:

- access restricted to operator only
- access & permission management
- data management (editor, types & import)
- history and log viewer
- system monitoring

Relations:

- Web Server
- Push Notification Service

Technologies:

- HTML, CSS, Javascript
- Java
- Swift, Objective-C

After outlining all different components, while keeping the aspect of portability (S.A.02) in mind, it becomes apparent which arrangements make sense and what variations might be possible. As a result, two more or less distinct designs are proposed. One is a rather centralized approach and the other involves more platform types and outlines a certain flexibility.

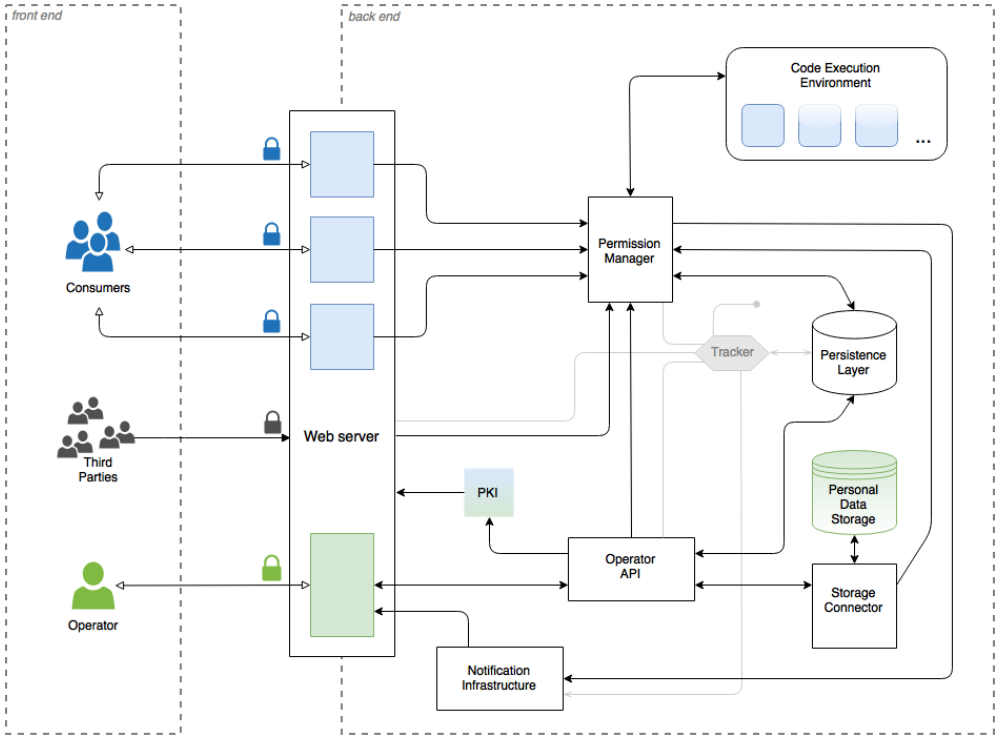

Figure 5.1: ReFlowd Architecture, centralized composition

The main difference between the two compositions is the non-existence of the mobile platform in the centralized approach (Figure 5.1). Although *centralized* (or monolithic) only refers to the components arrangement on a *server*

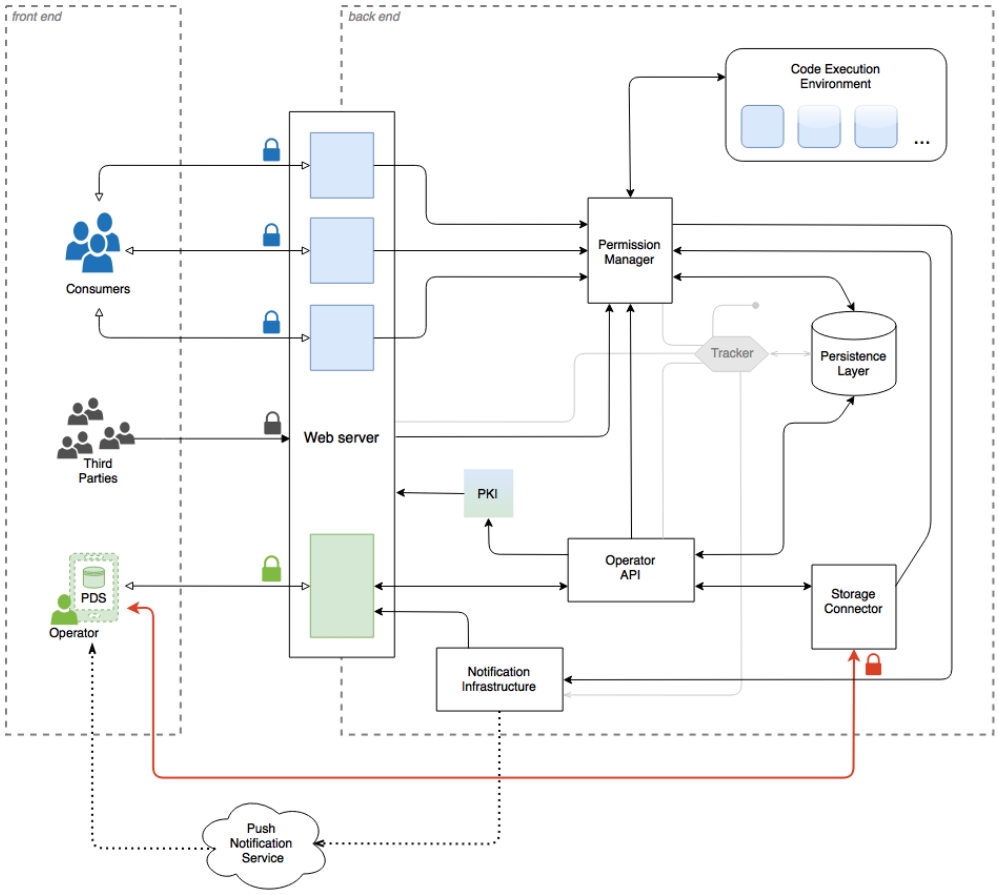

Figure 5.2: ReFlowd Architecture, distributed composition

platform, originally consisting of a single process that contains all components and is thus is responsible for every task.
It is also imaginable that all server components are note necessarily placed into one server environment, but being distributed over several virtual machines or containers, so that they can scale and run more independently. This can improve *redundancy* as well.

In theory, it's indeed possible to arrange the components in such a way that they are all located on the mobile platform, or even to a desktop device, which are henceforth considered equally in their platform properties (as of Table 5.2). However, this comes with downsides and major issues that are far from trivial to solve. Nevertheless, to not only ensure nearly 100% uptime and discovery in a landscape where NAT[6] and dynamic IPs are still common practice, for mobile platforms as well as on the desktop, all components except the user interface are therefore required to be implemented natively. From an operator's perspective, that would mean to have all components at hand and, therefore, full control over the *ReFlowd*. It would still raise security concerns, though, because the devices mentioned before might serve various distinct purposes and would therefore introduce potentially vulnerabilities.

Aside from providing the operator with a non-stationary and instantly accessible interface to her *ReFlowd*, involving a *mobile* platform primarily has the purpose of enabling the data subject to carry all her sensitive data along. This is considered a major advantage over the centralized approach, were all the personal data is located in the *'cloud'*. Depending on the perspective, it can either be seen as a *single source of truth* or a *single point of failure*. Regardless of that, it introduces the demand of a backup or some redundancy concept, which has briefly been touched on in the discussion about database system requirements within the section on *data (Chapter 5)*. A *mobile* platform as part of the system makes it more easy for the data subject to establish a security concept in which the relation between *Personal Data Storage (PDS)* and the rest of the system is much more liberated, so that all access attempts only happen under full supervision. It is debatable

[6]Network Address Translation; practice of routing traffic between and through distinct networks address spaces by remapping IPs from those different networks onto each other (e.g. by utilizing ports)

whether to place the *permission profiles* in the *Persistence Layer* among all other domain-related information, put it into the *PDS* too, or define it as having its own storage component, in order to be flexible in its placing.

The process of obtaining personal data determines that the *PDS* only is accessed by a component located on the server (*Storage Connector*). Now, moving the *PDS* away form the *server* makes it more difficult to access. Assuming the host environment provides a simple mechanism to configure its firewall and listen on a port for incoming connections, which is not very likely, because of host-wide security policies, the *Storage Connector* would still need the IP form the device, and even then, the already mentioned practice of NAT and dynamic IPs would require a signaling process, through which a connection to the *Storage Connector* is established.
Two approaches are proposed to circumvent this issue. Either an ongoing bidirectional connection (e.g. WebSockets) can be utilized to push data queries from the *server* platform, or the *Storage Connector* queues the queries and informs the operator via Push Notification on pending attempts to access the *PDS* located on her mobile device, so that she can then actively pull the queries and push the results back to the *Storage Connector*. Those are two distinct general approaches, details may vary.

Authenticating consumers is performed based on TLS by the Web Server and its configured subdomains including their individual keys and certificates provided by the *PKI*. The operator authentication is done either by the *Operator API* or by the *Web Server*, depending on the *Web Server's* capabilities. Though, it makes more sense to entrust the *Web Server* with that task, because it's the outermost component and it would prevent unauthorized and potentially malicious requests from getting further into the system. And as mentioned in the conclusion of *Chapter 5 - Authentication* it is reasonable for *mobile* platforms to change the operator authentication from JWT-based to TLS-based *two-way authentication*, which would otherwise be inconvenient when using web-based front ends.

If components are placed only on the server and require communication with each other but are separated into independent processes, then some inter-process communication need to be established (e.g. sockets). It is also conceivable that inter-communication between server components could just

be unidirectional. Approaches like changing configuration files by writing to the filesystem can therefore be feasible in some cases. Components that can vary in terms of their platform have to communicate to other components via *HTTPS*.

The architecture implicitly distinguishes between two different groups of endpoints. These endpoints that are made available by the *Web Server*, which reverse-proxys incoming connections to role-related (operator or data consumer) components. Starting from that, this separation can be driven further by simply encapsulating those components into services, that are related to one of the roles or used by both. This basically results in the *Web Server* communicating with the two role-grouped services in a bidirectional manner. The group of endpoints for data consumers mainly consists of those through which *access requests* and *permission requests* are coming in and the public one, that is used for when consumers apply for registration. The other one is a small group of endpoints required for all tools the operator might need; from data API or notification to authentication and web-based user interface. Furthermore, it might be considered to partially apply the principles of a RESTful design. However, this makes the API potentially more complicated and insufficient, since the *Operator API* does not follow the concept of resources. Instead, it requires a rather functional design, which makes, for example, *Remote Procedure Calls* or a *Publish-Subscribe* pattern more appropriate to use.

Considering the rapid growth of emerging website and applications, which all require user registration, users are getting tired of creating new accounts. Hence they tend to reuse their password(s). Providers started outsourcing that sensitive topic of user management by integrating third party authentication services, which not only makes that feature almost effortless to implement, but also leaves the responsibility as well as the accessibility to those service owners. Whereas users get the benefit of just using one account for all their apps - a universal key so to say, but only one exemplar. So the downside here is, in reality only a handful of third parties [152] provide those authentication services.
OpenID is designed with a very specific type of scenarios in mind, namely the one just described - bringing decentralization to the market of authentication services - which differs from the ones addressed by the *ReFlowd*;

at least when it comes to data consumer interactions. The *ReFlowd* has the ability to become the digital representation of its operator, therefore it can and should also be used to authenticate that individual against external parties.

Conclusions: Considering the amount of effort a single-platform composition, namely on a desktop or mobile device, would take to get fully operational with respect to the specification, it is not only reasonable but also more secure to involve a server platform with proper security measures, a static IP, and high availability, even if that server is a local machine connected to the operator's private network. That said, it is sufficient to start with the *centralized* approach and as suitable mobile applications emerge that are supporting major administration features, notifications, and *Personal Sata Storage*, it should be possible to migrate effortlessly towards the *distributed* approach that brings a higher level of confidence because all the sensitive personal data is not on some computer machine somewhere on the internet, but right in the hands of its owner. By the proposed architecture, all components (or groups of components) are portable and therefore relocatable among the suggested platforms; and with the introduced authentication methods for operators, multiple front ends for the same *ReFlowd* are thereby supported and can be implemented with almost no effort, which, in return, covers more use cases. As a supplement, an *identity provider* based on the OpenID standard would fit nicely into the existing arrangement and does not interfere with the other components. However, it is beyond the scope of this work to elaborate on this topic. For now it is stated as a feasible and logical addition to the *ReFlowd*.

5.6 Environment and Setup

As stated in the project's core principles *(Chapter 3)*, *Open Development* is vital for the project to gain trust. Interestingly, this has a significant impact on how a *ReFlowd* might be deployed or installed. All its components can just be grabbed and used as it suits everyone's needs, while respecting their licenses. Furthermore, enforcing *portability* (S.A.02) leads to a more simplified and independent development process that can be organized in a

way so that the primary division into components can be leveraged.

The range of host systems for *server* platforms is highly diverse but the main shares [153] belong to either the UNIX or LINUX family, even though almost every platform is POSIX-compliant.[7] When it comes to *mobile* platforms the market is far less diverse. Native applications are either developed in Java (for Google's Android) or in Swift (for Apple's iOS). Whereas the host systems have nearly no relevance for *web* platforms, other then the screen size and maybe which browser and version the environment system runs. But that's a condition the user can change if necessary.

As a result, being able to use certain components on a *server* platform depends on what *server* environment is available and vice versa. In order to decide what implementation of a component is suitable, it's crucial to know in which environment that component has to run. Either way, it is also important not to forget all the dependencies a component itself might have. Such constraints can be avoided by abstracting the runtime of those components and either embed every required software dependency or provide them in separate runtimes, if possible.
Depending on the used technologies, this concept is commonly known as *virtualization* or *containerization*. It isolates software by putting them into a so called *container*. But since those container-wrapped components still have to interact with each other, they need to be supervised or at least managed. This is done by an *orchestration* software, which not only allocates system resources but is also capable of emulating a whole network infrastructure (e.g. DNS, TCP/IP, routing). Thereby, it is utilized to determine how certain containers (and their contained components) are allowed to communicate and what resource are accessible from the inside (e.g. filesystem). This complete abstraction to the surrounding environment effectively means it's the only dependency the *ReFlowd* would have, regardless of how its components are implemented. They just have to be *'containerizable'* - by satisfying the *container image specification* [118]. This concept can also be utilized for the *Supervised Code Execution* (S.A.01) mentioned before without any restraints.

[7] Portable Operating System Interface; a collection of standards released by the IEEE Computer Society to preserve compatibility between operating systems

Migrating from a server-located *Personal Data Storage* to a *mobile* based version introduces another challenge. The subsequent approach is a first and more general solution to that problem.

NOTICE: it is assumed that a running instance of a ReFlowd* is in place, the operator owns a modern mobile device and on this device a *ReFlowd* mobile application is installed.*

1. After starting the app, the operator needs to establish a connection between server and mobile application. Therefore, the operator has to scan a QR-Code with the help of that app. The QR-Code is presented to the operator within the management tool of the *ReFlowd* running in a browser. Alternatively, the operator inserts her credentials into a form presented by the mobile application.

2. After the connection is established, the operator can trigger a progress that duplicates all her personal data to the device that has just been associated with the *ReFlowd*.

3. At this point, one of two paths can follow, depending on whether a complete write log for the *personal data* (see discussion about backup strategies in *Chapter 5 - Data*) exists or not.

    a) If *log exists*, query by query the whole log is obtained from the existing storage and is then again executed in chronological order by the query language abstraction, starting with the oldest. The only difference here is that the target storage, on which that query is actually performed on, is located on that newly introduced platform.
    b) If *log does not exist*, the situation is more complicated if the database systems are not based on exactly the same technology. This would mean additional migration software is required. If both database systems provide import and export mechanisms that support at least one interoperable data format, the migration software can leverage these features simply by exporting all the data and saving it to the filesystem. The software then transfers the dump to the target environment and triggers the import process.

When this is not the case, the software not only needs to be aware of both database systems and their native query language, it also has to have a comprehensive understanding of how their data structuring concepts work, in order to reliably transform one into the other. So, to be more specific, at first the software has to analyse the structure of the source database. Based on this result it might need to perform some configuration on the target database before actually obtaining the data from there. Received data then needs to be transformed into queries that are supported by the target database system. Those transformed queries are transferred to the target environment, where those incoming queries get executed until all data is migrated.

4. After the duplication process has finished, the operator can decide which PDS the *ReFlowd* should use to serve *access requests* and what should happen with the other storage(s).

To conclude, a migration process like moving personal data from one platform to another can be further simplified and robust if a complete query log would exist. It is also worth mentioning that the migration process described above is not restricted to exactly this source or migration direction. As long as target and source are either a *server* or a *mobile* platform, any variant is imaginable.

Conclusions: Installing a *ReFlowd* should be straightforward with the least possible effort being spent for preparations. Package managers of all popular operating systems should offer (semi-)automated installations. Additionally, components themselves and the project as whole have to provide detailed documentations for various ways of how those parts or the entire system need to be installed. Alternatively, data subjects might be willing to entrust external third parties with hosting a *ReFlowd* instance for them. In that case, the distributed approach involving a *mobile* platform might come in handy, so that the actual data is not stored somewhere beyond their reach. The *ReFlowd* as an open source development encourages anybody who is interested or even wants to contribute to checkout the source code of the various implementations, get it to run, and play around with it. Therefore, at least the components of the *server* platform are required to document

what other software they depend on, so that the target environment can be prepared properly. Aside from hardware, on which the *ReFlowd* needs to run, the only other requirement is owning an internet domain that is registered on a public DNS[8] server and has no subdomains configured yet.

If a component needs to get segregated from its host environment, *containerization* is the recommended technique since it causes less overhead compared to *virtualization* and is generally a lightweight approach. Although additional abstraction might also introduce new problems, in form of complexity, instead of solving them.

5.7 Attack Scenarios

Previous sections have already touched upon a few potential attack types and vulnerabilities of the system, such as man-in-the-middle attacks that are possible during connection establishment and data transfer, and session hijacking, which can be done by stealing the JWT used to authenticate privileged entities to the system. Attack vectors responsible for these and other scenarios as well as possible counter-measures are discussed in detail within this section.

Before that, however, is to work out what motivation(s) would drive such attacks. The first, and probably most dangerous motivation, appears to be *Digital Identity Theft*. Meaning that the attacker impersonates the individual that was originally associated with the system. Then data consumers are misled into believing that everything the intruder does with the *ReFlowd* is happening on behalf of the data owner herself. Such control can be abused for vicious data changes to harm the physical counterpart or the access can simply be exploited to unopposedly extract the individual's personal data. The more data the system contains and the more purposes it serves, the more power that can be gained by controlling it. Those motivations are extrinsically driven by third parties, whereas intrinsic and mostly unintended damage can be caused by system or human failure, which may lead, for example, to data loss or irrecoverable system access.

[8]Domain Name System; decentralized open directory that associates readable (domain) names with IP addresses

As the name states, in a man-in-the-middle attack, the transferred content of a communication is intercepted and possibly tampered with while the participants remain unaware. A common solution here is to encrypt and sign the content before transferring it. For HTTP communication this is implemented by the *Transport Layer Security (TLS)*. The crux here is to move from an unencrypted to an encrypted connection. This upgrade procedure involves the *Diffie-Hellman-Key-Exchange* to agree on a pre-shared key and certification based on *asymmetrical cryptography* for authentication. The procedure starts i.a. with the server-role presenting the client-role with its certificate *while the connection is still insecure*. Therefore, at this critical point, it is vulnerable to man-in-the-middle attacks. In order to prevent this and to ensure the trustworthiness of the certificate, the client has to verify the certificate. This is done by using an installed certificate issued by trusted public CA, which has therefore trusted the server's certificate, to examine its chain of trust.
But since HTTPS connections between data consumers and the system use certificates that are issued by the system itself instead of relying on CA-certified certificates, there is no chain of commonly trusted entities that can verify the presented certificates. So, in order to still trust those certificates despite being sent on a insecure channel during the connection establishment, they have to be handed over in private sometime before. This is defined in the registration process in *Chapter 5 - Access Management*, where the consumer either presents a QR-Code served via publicly certified HTTPS, which he is responsible for but can easily be verified by the data subject, or submits the registration request including the CSR to a URL provided by the data subject that is also reachable only by publicly certified HTTPS. The server then issues the certificate and sends it - alongside with its own certificate - back to the consumer also via publicly certified HTTPS. The consumer is able to trust the server certificate and can even pin it to detect fictitious certificates it might get presented with.
This approach enables not only confidentiality and integrity of the data exchanged between consumer and data subject, but by enforcing *two-way authentication* authenticity of the respective opposite is provides to both parties as well. While other HTTP connections via TLS may still rely on certificates signed by publicly trusted CAs, regular key change and certificate renewal has therefor to be ensured.

Another aspect of the system that could be vulnerable to certain attacks is the authentication mechanism used by the data subject to log into her management tool. A *JSON Web Token (JWT)*, which contains all session information, serves as the key. As mentioned before, any connection between components is forced to be established only by TLS.
So from this perspective the JWT is no less or more secure than any other type of credentials. However, this doesn't prevent the token from being usable to the attacker once it is stolen. A short expiration date, equal to a session timeout, and token invalidation cycle, which is the same as forced logout, can reduce the potential harm an incident like this may cause.
Furthermore the integration of 2-factor authentication hardens the authentication procedure. But in order to not introduce another dependency, namely an external service providing such functionality, 2-factor authentication is only supported when a mobile device is associated to the *ReFlowd*.
Further precautions can be taken by preventing attackers from getting close to such token, referring to *cross-site scripting (XSS)*, to which web-based graphical interfaces are vulnerable. Approaching this issue means to abandon external resources providing parts of the interface and storing all content on the server platform instead, serving it with the system's own web server. Even if that means an increased load time caused by the browser constraints of how content is loaded, this isn't an issue in HTTPS/2 anymore.
Even though connections are based on TLS, a *Replay Attack*[9] is still a possible scenario. Using a unique identifier for a JWT (`jti`) is not able to prevent this attack, but if suspicious behaviour is detected, it can help to identify the related JWT and act accordingly (e.g invalidate token). Depending on the implementation, this strategy may ultimately make the server stateful. Thus, a more detailed analysis of such scenario is required in order to make an adequate decision on this topic.

The approach of running consumer-provided programs locally in order to prevent personal data from leaving the system represents a key part in this work, but it also raises major security concerns. To address those concerns, such a program is executed in an application container, which encapsulates the runtime from the host environment and allows the restriction of resource access, such as network, filesystem, or computing power, to the required

[9]delaying or repeating transmissions, so that the recipient is deluded with wrong assumptions, for example, being securely authenticated

minimum. This concept, previously introduced as *Supervised Code Execution* forms a containment layer preventing security breaches towards the host environment, which ultimately means a great security enhancement.
Nevertheless, it makes involved components and dependencies vulnerable to almost any security flaw that they might carry in. Therefore it is vital to keep all related software up to date, which probably means enabling automated update mechanisms. That again introduces yet another type of attack vector, which can be lessened by only using signed and trusted software and resources.

When it comes to personal data, existing social networks and other large platforms founded on user-generated content have already become de facto data silos, and thus a single point of failure. A more decentralized approach, for example, the concept proposed here, diminishes the impact of potential security breaches those platforms may experience. While, from an global perspective, this can be valued as a step in the right direction, from the perspective of a data subject, a *ReFlowd* instance still represents a single point of failure as well. But in order to provide maximum control over her own personal data, this design choice appears to be the logical consequence. Additionally, several system components are highly portable so that more sensitive parts, like the *Personal Data Store*, can be relocated away from the server platform into the data owner's mobile device. In this case, a JWT issued by the *PSD*, for example, can be used to authenticate the access to personal data.

However, those measures are still no guarantee of comprehensive security protection. The only way is to continuously improve the system and provide mechanisms to securely recover from any type of incident. This is one reason why a complete open (source) development is enforced from the start, allowing vulnerabilities to be easily disclosed and hence addressed immediately.

Conclusions: Securing data transport and storage are the two pivotal aspects to focus on. Choosing self-singed certificates over publicly trusted CAs in order to not only encrypt but also reasonably authenticate the consumer-system communication needs to make sure that the preliminary transfer of the certificates is not compromised, so that both parties can safely rely on them. Settings for any cryptographic procedure regarding their expense have

to be chosen in favour of increasing the level of security instead of reducing resources and cost. The general mindset for this project is to prefer proven libraries and technologies over own implementations for cryptographic concepts and algorithms. Nevertheless, ongoing reviewing and re-evaluation of those dependencies, as well as the software itself, is mandatory.

As a future enhancement it is planned to integrate an *intrusion detection system* into the server platform's host environment, even though all server components are recommended to be encapsulated by containers. Monitoring transactions and events on the application-level is already part of the system architecture, facilitated by the *Tracker* component. Furthermore it is considered to apply full storage encryption to the *Personal Data Storage*, which would result in major security improvements. Therefore, when the personal data is located on the server, it should stay secure even if the system is compromised. As a downside, the system might lose convenience in user interaction. It essentially comes down to an act of balance between security and convenience. It is hardly possible to simplify or abstract security measures without violating other (general) values. The key is to find the right compromise for the right motive.

5.8 User Interfaces

Designing graphical user interfaces is beyond the scope of this work and the *ReFlowd* specification as well. Nevertheless this section shall be understood as a collection of proposed ideas addressing the questions of what types of user interfaces the *ReFlowd* should provide and which features they might need to support.

The most notable characteristic used to distinguish user interfaces from each other are those interfaces that are visible and the ones that arn't. For example a *graphical user interface (GUI)*, composed of visually separated areas with a certain semantic and assembled with meaningful objects on which the user can physically act, for example by touching them. The interface then reacts on those actions by changing its appearance. In this way the user can recognize and comprehend her actions. Whereas non-graphical user interfaces don't provide the user with objects or surfaces to interact with.

Instead, the primary medium is text, regardless if it's human-readable or not. But *command line interfaces (CLI)*, available mainly in command line environments or shells, still provide a certain level of interactivity. A running program can pause in order to prompt the user with an input request. If an input is made and submitted, the program then proceeds. The group of interfaces whose interactions can be programed and thereby fully automated can be called *application programming interfaces (API)*. Depending on the transport technologies, it's not unusual that *API* interactions consist of just one action causing one reaction.
Non-graphical interfaces enable interactions on a lower level. Even though they provide more functionality and can be more time efficient, they are more rudimentary and often less secure. While *GUIs* are normally meant for end users to interact with applications on a more sophisticated level, *CLIs* are used during development, for automation, or for server environment administration; probably remotely since they are typically headless. Whereas *APIs*, documented by their providers, enable software developers to program automated requests against those interfaces.

Table 5.3 provides a list of features and associates the different types of user interfaces mentioned before, which indicates if they should be supported by a certain type. It is notable that the *GUI* provides the operator with a powerful tool. Hence it requires reliable protection mechanisms *(Chapter 5 - Authentication)*. Whereas *API* capabilities are very limited, because it's the one interface that the *ReFlowd* exposes to third parties.

Table 5.3: Features supported by the given user interfaces

Feature	GUI	CLI	API
manage permission profiles (P.VIU.03)	**X**	-	**X**
view access history (P.VIU.04)	**X**	**X**	-
register consumer	**X**	**X**	-
add new front end	**X**	-	-
authenticate operator	**X**	-	-
migrate personal data	**X**	**X**	-
review permission requests (P.I.04)	**X**	-	-
create & maintain templates (P.I.05)	**X**	-	-
adjust precision of data (P.I.06)	**X**	-	**X**

Feature	GUI	CLI	API
introduce new data structs	**X**	-	**X**
configure *ReFlowd*	**X**	-	-
import personal data	**X**	-	**X**
read/access personal data	**X**	**X**	**X**
manipulate personal data	**X**	**X**	-
run supervised code execution	-	**X**	**X**

The architectural design includes *web* and *mobile* platforms. While prioritizing a web-based *GUI*, the management tool for the operator also needs to be natively implemented for common mobile systems (P.VIU.02); in this case Android and iOS. This enables real-time notifications (P.I.03, P.B.02) on mobile platforms, whereas the same feature is added to *web* platforms by providing WebSocket-based connections. Since screen sizes can vary - in particular on *mobile* platforms - the *GUI* is required to be highly responsive and has to adapt (P.VIU.01) to various screen sizes. Given the capabilities of the management tool, an inaccurate or error-prone rendered *GUI* can quickly cause unintended incidents. Thus, the main focus must be to ensure a very robust and low-latency interface rendering.

Known challenges for the *GUI* design are primarily to develop very efficient, but also fun to use interfaces for reviewing *consumer registrations* and *permission requests*. The latter can become especially hard to solve because of the problem of how to display graph-based and nested data structures in such a way that reviewing and also manipulating them is easy, even on the small screen of a mobile phone? One approach could be to utilize the *accordion* pattern [154] for edges and start nesting them in order to represent subsequent data structure. The interaction then might look like tree-structured navigation; moving alongside relations just by expanding and folding in data items.

Since other parts of the system have to provide the mechanisms for increasing or reducing the *precision* of data due to privacy protection, the challenge here is to find the right design concepts for the data subject to facilitate those adjustments. Precision adjustments can be achieved by either changing the sampling rate in a dataset containing a series of data items, or by rounding

values to a certain extent. Examples are cutting fractional digits of the latitude and longitude values in a position, or removing all position information obtained between every quarter from a full day tracking period. Whether data subjects can choose from an abstracted precision grading (e.g. *high*, *mid*, and *low*) or they set specific type or unit related filter mechanisms, configurable defaults on a system-wide level should be provided by *GUIs* in any case. The following data types are supposedly vulnerable to compromising privacy, thus proposed to support precision adjustments: *Date* (time), any kind of absolute measurements, sets containing data series, and position information, as mentioned before.

Conclusions: The most important aspect, when interacting with something or someone, is being provided with feedback. An action typically causes - and is therefore *expected* to cause - a reaction. The result is an interaction, unless no reaction occurred.
The discussion above outlines the relevance of those interactions for the *ReFlowd*. Thus, for users and other software to interact with the *ReFlowd* interfaces are mandatory. Primary characteristics of those interfaces are complete functionality, security precautions and restrictions, as well as comprehensive documentation. Visual user interfaces also need to provide reliable and adaptive rendering, a consistent and encouraging interaction design. *GUIs* need to be provided for all front end platforms, primarily to provide an efficient user experience for the operator. The operator is the only role with permissions to access a *GUI*.
Components on the *server* platform should provide *CLIs*, at least when no other technical option exist to interact with them. Also accessing the database from command line could be appreciated at some point. APIs are mostly meant for data consumers to interact with the *ReFlowd*, and perhaps for automated data contribution (based on the operator role; e.g. browser plugin). *Web* platforms might use those *APIs* as well. In any case, *APIs* must be separated according to the roles.

These are all vital characteristics that need to be outlined in the *specification*. Their implementation details are not the concern of this specification, as long as every stated requirement is being acknowledged.

6

Specification

Based on all preceding discussions, a first draft of the *ReFlowd* Specification has been developed, which can be found as an attachment to this work. Not every conclusion has been applied, but various important aspects are already outlined so that first implementations can start to be built upon them. The draft leaves room for followup work.
It must be recognized, however, that in its current state it has no claim of completeness, continuity or accuracy. Frequent changes are to be expected in the future.

7

Conclusion

The *Right to privacy* is a fundamental right found in numerous national constitutions and confirmed by court decisions, stating that:

> *Given the context of modern data processing, the protection of individuals against unlimited collection, storage, use and transfer of their personal data is subsumed under the general right of personality governed by [...] the Basic Law. [...] this fundamental right guarantees in principle the power of individuals to make their own decisions as regards the disclosure and use of their personal data. [155, p. 3]*

However, this legislation goes largely unrecognized today, as demonstrated in *Chapter 2 - Fundamentals*. Secret agencies simply ignore or reinterpret [156] such laws, and private organisations are usually out of jurisdiction. But since it is still valid law, individuals have not only a reasonable desire but, as stated above, the right to protect their personal data. A web service, controlled only by themselves, can contribute to a move in this direction. It acts as her digital counterpart, encouraging the data subject to effortlessly maintain all her personal data. She can explicitly and selectively make them available to third parties, while carrying and storing them on her mobile

device. This software enables the data owner to keep track of her data flows, while at the same time endeavors to be as trustworthy and secure as possible. The specification for such a service is hereby available as a first *Working Draft* and is attached to this work.

7.1 Ethical & Social Relevance

In *Chapter 2 - Personal Data in the context of the Big Data Movement* and *Personal Data as a Product* it is described what impact personal data has in today's society and, in particular, the economy. Data as a commercial good has led to humans becoming the product. Those, who collect, analyze and interpret personal data using algorithms to uncover hidden information that may help determine people's motives and patterns in order to forecast their behaviour, with the purpose of improving their services and ultimately increasing revenue. What is often overlooked is that correlation is no proof of causation. Furthermore, those algorithms naturally inherit the bias of their creators. Together, this leads to privacy violations, and worse, discrimination.

These 'side effects' are not acceptable in a society where tolerance and solidarity are desirable values. Therefore we, as individuals, have to move away from being the product and start to become self-determined again. The answer to what platform we want for a global community, like the one envisioned by Mr. Zuckerberg [157], must not be *facebook*. Instead the answer should be built on openness and neutrality, and on an infrastructure that is accessible to everyone and owned by everyone - like the origins of the internet, *"an Internet of People"* [158, Sec. 5]. The software proposed here can aid in reclaiming the internet as the platform for a global community - our society.

While trying to prevent our personal data from being read by men in the middle during transfer (HTTPS), we have missed cutting out the middle-men between our data and those who are utilizing them. Thus, every individual must be re-enabled, so that they can fully decide, on their own, what level of privacy they are willing to share and under what conditions other parties are permitted to access their personal data. The software proposed here can

help to counter massive and unimpeded data collection and minimize the possibility of discrimination. However these issues can be addressed only partially by these means. As mentioned before, bias, causing discrimination, is naturally inherited and therefore needs to be acknowledged by raising awareness and teaching involved authors and developers about these issues and how to avoid them or at least address them properly. For example, by reducing questionable data items, considering possible consequences for data subjects of their interpretations, or by keeping indirect correlations in mind. Only a combination of both approaches can remove the greater part of bias, embedded not only in those machines but in all of us.

7.2 Challenges & Solutions

As touched on before, the motivation for this work was to overcome discrimination and privacy violation experienced by data owners and caused through large scale data collection, to prevent data from being collected in the first place. This turns out to be a major challenge, because third parties may still require to possess personal data, at least for basic processes such as ordering or payment (e.g address), and therefore raw data needs to be transmitted, which is, by the nature of information technology, a process of duplicating data. Thus, the definition of ownership, at least on a technical level, cannot be applied anymore *(Chapter 2 - Digital Identity, Personal Data and Ownership)*. Where legislation has already failed to address the issues mentioned above, technology has its own limitations. Certain scenarios, such as creditworthiness checks or calculating recommendation, are able - from a conceptional perspective - to support *Supervised Code Execution*, which allows the data to stay within the *ReFlowd*. However, even when data cannot be completely contained in the *ReFlowd*, being able to control access, or have an overview of who has access to what data, is already a major advantage.

The *ReFlowd* may be valued as the digital representation of a human being. Hence, third parties, who are interacting with the digital self, want to be able to trust, rely upon, and maybe even verify, that the received or accessed data actually relates to the represented individual. Several approaches to ad-

dress this have been discussed in *Chapter 5 - Data Reliability*. The proposed solution was a procedure involving the administration (or another party with similar properties) or an official document *(eID-card)* that signs and certifies the data in question, which can then serve a verifiable proof for the data consumer. Although both require an additional but diverging level of effort for the data owner, the QES-based eID-card version is not widely used and the *Recurring Certification* performed by the administration is a rather theoretical concept. However, the question of an individual's willingness to depend on the government's good intentions may remain, since both solutions require a public key infrastructure to verify the signatures. A more liberal and open approach like the *web-of-trust* could be a valuable alternative solution.

Retaining integrity of the personal data, that's being stored, but also maintaining that data, becomes rather complex when allowing a flexible data structuring. That is, allowing the owner to change underlying data *structs* at any time, results in running a background data migration after every structure change, regardless of where the *Personal Data Storage* is located at that moment. In order to simplify that process, one solution is to create a new database next to the existing one, execute every query in the write history and adjust the affected data items according to the new type structure on the fly in those queries during the mirroring process.

While reviewing the different scenarios described in *Chapter 1* and comparing them with the established capabilities of the *ReFlowd*, it can be observed that the developed system is able to provide a reasonable foundation for all of them. Even though it requires the willingness of all sorts of third parties to integrate with the proposed specification and a rethinking and change of perspective of how personal data has been obtained and utilized until now. How the industry might respond to such fundamental changes is an open question, but this is a secondary issue. First, the focus must be on increasing the adoption rate of data subjects. Problems and benefits, as mentioned in the *Ethical & Social Relevance*, must be outlined. It is vital to create a trustworthy and easy-to-use system, which can be archived through public development and an active and healthy community. Ultimately the result has to provide a great user experience, which primarily refers to installation and setup procedures as well as data management and a seamless mobile integration, which emphasizes the benefit of *'carrying your data with you'*.

If the acceptance has reached a critical minimum, *data subjects* might be able to demand potential *data consumers* to start adopting the specification if they still want to be able to access their personal data.

7.3 Business Models & Monetization

An imaginable opportunity for a viable business model is to provide commercial services around the proposed system. Developing and implementing the specification yields expertise, that may qualify, for example, to offer high-quality hosting of *ReFlowd* instances for end users, or to customize implementations in order to adapt to special scenarios, or to assist with integrations. Even though the underlying technology is open source, history shows various examples of successful businesses based on open source technology, such as Meteor[1], Mozilla Foundation[2], Red Hat[3], or Wordpress[4]. It allows the development to be driven forward and to contribute back to the ecosystem from which the whole community eventually benefits from. Additionally, because there is a specification, interoperability remains preserved on all levels.

Another, very distinct, way of monetization is the possibility and willingness of data subjects to sell their personal data to third parties, on their own terms. This can mean to actually hand over the raw data, but also to simply license the access, which is the business model of data brokers. Of course, this requires further specification, additional infrastructure i.a. for anonymization and payment transfer, and probably a marketplace as well. But let's take a step back for a moment and reevaluate this idea.
A negative scenario may result in people trying to make a living by selling all their data to everybody, or worse, having no other option than doing so. In contrast, other individuals, who don't want to monetize their digital identity could therefore be discriminated. Or data consumers who previously depended on gratis data, for whatever reason and however obtained, may now run dry, because if people are able to charge for their data, they do it.

[1] https://meteor.com
[2] https://www.mozilla.org
[3] https://www.redhat.com
[4] https://wordpress.org

The market may be not stable and data becomes inflationary.
On the other hand, a positive scenario may bring more quality data to various places where they are eagerly needed to improve our social co-existence. Studies and research might be able to reveal otherwise hidden solutions (e.g. health care, public sector, government, etc) that could be commonly relevant. Furthermore this approach would cut out current middle-men whose business model is to collect and monetize personal data, or at least it could result in a fair revenue share between them and data owners. This may also lead to data consumers become more aware of what data they collect and what parts they actually need.
Either way, those are all just excerpts of possible outcomes, and a mixture of both directions is conceivable. In conclusion, certain scenarios may need to be restricted by law. For example, an insurance company should not be allowed to give those members, who permit the company to access their data, an advantage (or discount) over those who don't. This is a simple case of discrimination. In general, if individuals don't want to monetize their personal data, they must not experience any disadvantages. This work originates i.a. in the attempt of preventing this in the first place. It must not create something that encourages discrimination. From that point of view, it is questionable to equip data subjects with such capabilities. On the other hand, though, regardless of this work or such feature, such discount offers as described before are already starting to occur [web_2015_health-insurance-discount-in-return-of-data] [159].

7.4 Future Work

On the one hand, looking back and re-evaluating past research mentioned in *Chapter 2 - Related Work* can give the impression that only minor progress has been made in this field, despite following a reasonable approach that is more relevant today than ever before. Only a few projects have managed to last beyond a prototype or case study, to which the problem of user adoption rate might have contributed. On the other hand, the *openPDS/SafeAnswers* project, for example, whose specification has, in part, followed similar approaches, is fairly new and matches with the latest opinions of experts [160].

The critique, which the specification is partially able to address, is the lack of control of our own personal data while it's being collected. The existence and utilization of data analytics per se is a huge advantage for our society because of the ability to discover information and relationships we would otherwise not be able to uncover. This allows us to draw conclusions and thereby evolve. The *ReFlowd* is a tool that provides a foundation to do exactly that, but on a smaller and individual scope, and only if we choose to do so. However, this requires an additional stack of components to be integrated with the *ReFlowd*, i.a. *Data Mining* and *Machine Learning*.

The concept of *Supervised Code Execution* can be enhanced by migrating the *Personal Data Storage (PDS)* and execution to a *Blockchain*-based approach [161] as proposed in the Enigma[5] Project. Further on, idea of encrypting the *PDS* might be a reasonable step, although it would introduce new challenges. Therefore it requires a reevaluation of its necessity.
During concept development, it appeared necessary to define an additional role. The so called *data contributors* can be a plugin, client, or even just a dedicated account that is authorized by the *operator* to only push data into the *ReFlowd*. This concept was merely touched on and requires additional work invested to outline the details.
Two major, but rather independent, enhancements, that seem natural to include, are an *OpenID Provider*, and support of 2-Factor-Authentication for the individual represented by the *ReFlowd*.

The next steps to push development forward are, finalizing the specification by outlining each component so that the minimum viable set of features discussed in this work are supported, and then starting to develop the first components until a working prototype appears. The latter step typically goes along with finding and addressing flaws in the specification, which is an iterative process. Mobile components have to follow shortly and additional features such as those just mentioned may come afterwards.

In conclusion, the concept proposed here is far from being the ultimate solution, but it is nonetheless an improvement over the current situation, It definitely has its weaknesses and needs further development, but they are worth investing in, in order to reach the overall goals of increasing privacy

[5] http://www.enigma.co

and empowering individuals to control their own data.

ReFlowd *(Working Draft)*

Version 0.1.0

Introduction

This specification describes a system with the purpose of controlling the personal data of a single entity. By 'entity', a person, or an individual, is primarily meant. This individual manages all data relating to her in an online platform operated by her, which exposes a service making those data selectively accessible to third parties that might be interested. Meanwhile, the system tries to ensure that no personal data ever leaves the system.
The overall flexibility and portability of the system enables the individual to store her data on a mobile phone, for example, so that she is able to carry all her personal data along while providing the access to her data at all times.

Notation and Conventions

The requirements notation used in this document originates in the RFC 2119[6] and shall be interpreted as described there. This applies to the following keywords, recognizable by capital letters: MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL.

Values and words presented in a `fixed-width font` need to be understood as names and must therefore be used as is. If one of those starts with a

[6] https://tools.ietf.org/html/rfc2119

$ (Dollar) character, a variable is indicated, which means implementations must replace this variable with an actual value.

Lists that are numbered indicate an order which must be followed. Lists that start with a character (upper or lower case) have no claim to its order. Their items are inclusive (logical *and*) whereas bullet list items are exclusive (logical *or*).

Terminology

(The) System: refers to either the overall concept or the implementation of this specification

Platform: combination of environment capabilities and common role characteristics, which can be inherited from its hardware/device properties; possible values: *web*, *server*, *mobile*

Component: independent piece of software that is part of the system and might be capable of running on different platforms, but not at the same time; serves a unique purpose within the system

Operator: data subject and owner, not necessarily provider or host; individual, represented by the system and whose personal data can be accessed through the system; also referred to as *(data) controller*, *(data) subject*, or *(data) owner*

Third Party: external entity or vendor, who is yet to become registered as a *consumer* to the system

(Data) Consumer: external entity or vendor, who has registered to the system and is therefore permitted to request access to data or even to access data held by the system

(Data) Contributor: external entity, that is authorized by the *operator* to obtain and push personal data of hers into the system

Personal Data: data related to the *operator* that is contained in the system and selectively permitted to be accessed by consumers

Data Broker: entity with commercial interests in collecting, aggregating

and analyzing personal data from any possible resource in order to combine and enrich those and to license the result to other entities

Endpoint: dedicated entry for a specific *data consumer* to communicate with the system (e.g. access personal data)

Permission Profile: set of access rules and configurations tied to an *endpoint* that define, for example, how long, how often, and which personal data are accessible by a related *data consumer*

Access Type: defines the method by which access to personal data is provided

1 Overview

The overall purpose of this specification is to provide detailed instructions for building a web service that, on the one hand, encourages an individual to manage and maintain all the data relating to her in one place, and on the other hand, enables third parties to access such data if they are permitted by the individual to do so, preferably through supervised code execution instead of just handing over the raw data items.
The result is a one-to-many relation between the data owner - the individual - and all those who might require some data to, for example, process a purchase initiated by the operator or make a proper decision on her medical treatment.

The system architecture is designed with flexibility and portability in mind while still preserving simplicity and security. The result supports running some components on different platforms in a distributed way. Therefore the operator is able to operate the system while being on the move and can literally carry all her personal data along.

The design of this specification tries to leverage as many existing standards and open technologies as possible for every aspect and component. By recognizing common practices, its implementation and integration can be made as effortless as possible.

1.1 Key Features

Perspective: *Operator*

- full control over the flow of personal data
- maintain all personal data in one place
- real-time information and notification
- taking her personal data with her
- collect and analyze her own personal data
- trust through open development and open source
- commercialize data access on her own

Perspective: *Consumers*

- reliable single source of latest data about an individual
- access only those data that are actually required and thus reduce 'noise'
- access data that have never been collectible before
- distributed computing

1.2 Scenarios *(Excerpt)*

Online Purchase In order to proceed with checkout, a shop requires some personal data from the user, such as shipping address, email, and payment information. Either the shop is already a data consumer, then it would access the system in the background to check if any data has changed, or the shop has to register as a consumer first. After the registration process has been initiated, if needed, the user is forwarded to complete the checkout. After reviewing the registration and the data that are attempted to be accessed, the user is informed via email that the order has been processed and the shipment has started.

Social Network Instead of storing personal data by itself, a social network, after registering and being permitted to, can request and display data about a user whenever other users try to access them. Created content such as posts, comments, images, or videos can be stored in the system as well. The social network then just holds references to all the content so that it can

obtain and forward information on how to request those data, for example, with a one-time url.

Apply for a Loan The data that credit institutes take into account when deciding on creditworthiness of an individual can be directly accessed through the system; instead of gathering and acquiring them from all sorts of resources, including filled out forms from applicants. The credit institute takes out the part of the computation that is responsible for those calculations and hands it over to the system after it is permitted by the operator to do so. The system invokes that computation with the required data items and sends the result back to the credit institute.

Browser History A browser plugin, which is connected to the service, keeps track of every called URL. Those data, collected by the operator, can not only be reviewed and analyzed by herself, but also be made available to data consumers.

Movement Profile Instead of letting third party apps keep track of an individual's daily movements (e.g. fitness app), an app that is associated with the system, obtains and pushes those location data directly into it. If a third party is now interested in those data, it can apply and eventually get permitted to access those movement data, but at a resolution the data subject is comfortable with.

1.3 Architectural Overview

The architecture design (see Fig. 7.1) defines three different platforms where components of the system could run. While *web* and *mobile* platforms are primarily meant to serve as the front end of the system and to present the operator with GUIs, the *mobile* platform in particular might host the *Personal Data Storage*. However, that data storage can be located on any platform. This is enabled by abstracting the storage through the *Storage Connector* on the *server* platform.

That *server* platform also provides an external API for accessing personal data. Incoming requests from third parties and consumers are then processed by the *Permission Manager*, which i.a. decides if and how data can be accessed. For every consumer a dedicated exposed endpoint is provided that consists of a subdomain (e.g. `CONSUMER_ID.system.tld`). Other components are i.a. a *Code Execution Environment*, a *PKI* that provides and issues certificates and key-pairs to facilitate authentication at the endpoints. The *Persistence Layer* is represented by potentially multiple technologies, such as databases or filesystem. A *Notification Infrastructure* streamlines the different ways and technologies to notify the operator about certain events (e.g. system receives new registration). Probably one of the most important components is the *Operator API*, through which the operator i.a. is able to configure the system or manage permissions and the API is granted read/write permissions at the *Storage Connector*. The *Operator API* MAY partially be designed RESTfully, but accessing data MUST solely consist of the query language provided by *GraphQL*.

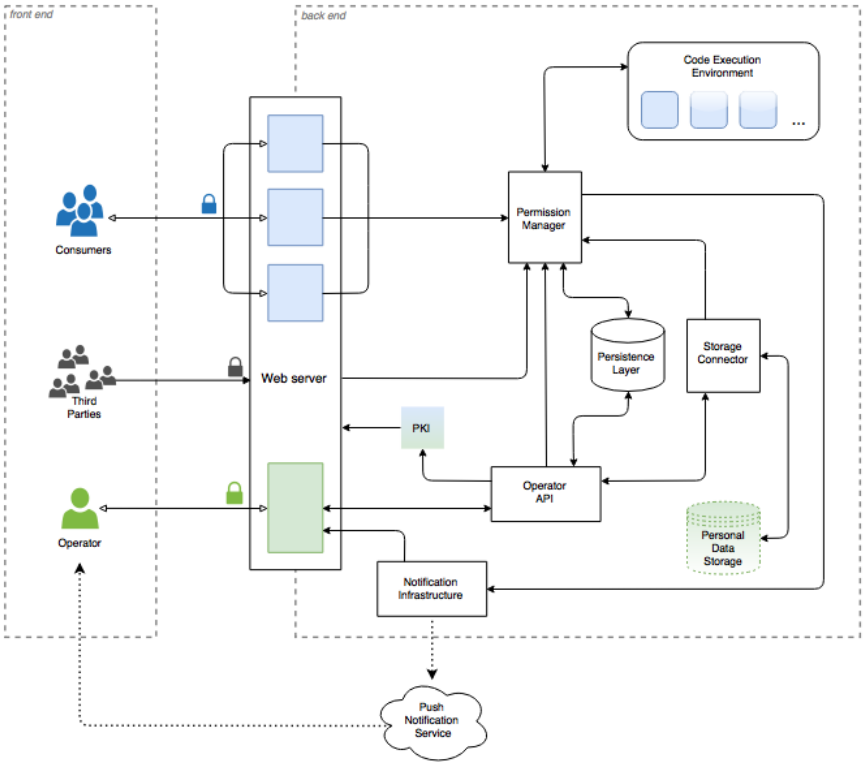

Figure 7.1: System Architecture (simplified)

1.4 Abstract Process Description

0. *Prerequisites: the data subject has an up-and-running system; existing third party aims to access personal data on the system*

1. Third party has to register at the system to become a data consumer, therefore it has to provide certain information to the operator either via QR-Code, which the operator can scans, or by submitting those information to a unique URL that is generated upfront and provided by the operator to the third party.

2. Operator reviews the registration. On a positive outcome a new endpoint gets defined and, depending on the content of the registration request, a permission profile is optionally created and configured by the operator. If the outcome instead is negative, an error message is prepared. In any case, the third party is then informed about the outcome via callback URL and optionally provided with additional information that is required for further interactions.

3. After the consumer has set up a client according to the documentation and the information he has been provided with, and after he successfully authenticates to the system, he then submits a query to the dedicated endpoint.

4. The incoming query is parsed and validated against all associated *permission profiles*. If the query passes, the request is processed according to configurations and depending on the determined *access type* (e.g. supervised code execution or plain forwarding). The response to the consumer either contains information about when and where the result can be obtained, or the actual result is included directly.

5. Finally, the query result is returned to the consumer, either directly or self-obtained later on.

1.5 Relations

consumer:endpoint [1:1] one entity (consumer) relates to one access point (endpoint), exclusively accessible and authenticated by TLS-based *two-*

way authentication

endpoint:permission-profile [1:n] zero or more *permission profiles* are associated to one *endpoint* and are therefore responsible and included in the query validation procedure

storage-connector:personal-data-storage [1:n] the *Storage Connector* MUST be able to access at least one or more *Personal Data Storages*, regardless of their location (platform)

1.6 Depending Technologies and Standards

a) HTTP (RFC 2616[7], RFC 7540[8])
b) TLS (RFC 5246[9]
c) WebSockets (RFC 6455[10])
d) JSON (ECMA-404[11])
e) JWT (RFC 7519[12], RFC 7515[13], RFC 7516[14])
f) GraphQL (Working Draft – October 2016[15])
g) Open Container Specifications (by Open Container Initiative)
   - image[16]
   - runtime[17]
h) X.509 (RFC 5280[18]
i) ACME (Draft 05[19])

- document-based database systems

[7]https://tools.ietf.org/html/rfc2616
[8]https://tools.ietf.org/html/rfc7540
[9]https://tools.ietf.org/html/rfc5246
[10]https://tools.ietf.org/html/rfc6455
[11]http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
[12]https://tools.ietf.org/html/rfc7519
[13]https://tools.ietf.org/html/rfc7515
[14]https://tools.ietf.org/html/rfc7516%7D
[15]https://facebook.github.io/graphql/
[16]https://github.com/opencontainers/image-spec/blob/master/spec.md
[17]https://github.com/opencontainers/runtime-spec/blob/master/spec.md
[18]https://tools.ietf.org/html/rfc5280
[19]https://ietf-wg-acme.github.io/acme/

- browser-supported technologies (HTML, CSS, JavaScript)
- Swift, Java

1.7 Prerequisites

a) DNS-registered Domain
b) publicly reachable server (e.g public IP or dynamic DNS)
c) [OPTIONAL] mobile device

2 Components

Web server an interface to (or proxy for) server components

a) process incoming traffic originating by operator, consumer, and third parties
b) all other server components are reachable only through this component
c) serve web-based front ends
d) perform all TLS-based authentications
e) perform load balancing, if necessary
f) pass through notification for web-based Management Tools that are connected via Web Sockets
g) rule-based traffic management leveraging network characteristics and similar

Permission Manager *(PM)* processes incoming requests from third parties or consumers

a) parse requests and proceed accordingly
b) examine and check queries that are aiming to access personal data
c) queue requests if further processing is blocked
d) notify operator about new registration attempts
e) create and manage permission profiles
f) read personal data through *Storage Connector*
g) delegate supervised code execution and obtain result

Key Infrastructure (PKI) provides the web server with keys and issues certificates

a) issue certificate based on CSR provided by third party
b) create and self-sign a system key-pair
c) create and sign key-pairs for every endpoint
d) maintain certificates and their validity (recurring renewal) issued by trusted public CAs
e) create new Diffie-Hellman groups

Storage Connector abstracts the *Personal Data Storage* and facilitates read, write, changelog and permissions

a) basic access management
b) connect to the PDS
c) validate query according to the abstracted query language
d) query personal data
e) keep track of all changes
f) provide migration and backup possibilities

Personal Data Storage *(PDS)* a database system where the personal data is actually stored

a) can be located on any platform
b) capable of translating the abstract query language into domain-specific query languages required by the supported database systems
c) add, change, remove personal data
d) provide translations for common operations (filter, sort, aggregate)
e) store change history
f) store binary data

Operator API provides all functionalities an operator must be capable of

a) perform all JWT-based authentications
b) read and write personal data through *Storage Connector*
c) configure, maintain, and monitor system

d) provide access history
e) manage data structs
f) can control *Permission Manager*
g) administrate restricted access for *data contributors*

Code Execution Environment *(CEE)* provides isolated runtime (sandbox) for supervised code execution

a) invoke software provided by data consumers with required data items
b) restrict access to the host environment
c) provision runtimes
d) perform test runs and code review
e) monitor runtime during invocation
f) hand results back to *Permission Manager*
g) archive software in *Persistence Layer*

Tracker logs events and transactions occurring in the system

a) track states of ongoing access requests
b) provide data for monitoring and system analytics
c) log access history
d) pattern recognition and anomaly detection

Persistence Layer *(PL)* combines multiple technologies to represent and hold the current system state

a) store:
    - system related data
    - component configuration
    - data provided by the *Tracker*
    - permission profiles
    - tokens
b) filesystem access
c) hold keys and certificates
d) cache certain runtime data

Notification Infrastructure a unified facility for server components to distribute notifications to multiple front ends

a) notify operator about pending approval or review
b) support native mobile notifications through connected Push Notification Service
c) is able to send email

Data Contributor an authorized entity, typically software, that sends personal data into the system

a) MUST be explicitly authorized by the operator
b) data structure and format MUST be known by the system

Management Tool (and other GUIs used by the operator) a graphical user interface, available for mobile and web platforms, accessible only by the operator to control the system

a) based on either web technologies or native technologies that are supported by mobile platforms
b) offer all relevant features provided by the Operator API
c) scan third party registrations via QR-Codes or generate URL to submit registration

3 Security

The following measures are required in order to improve upon and ensure a sufficient level of security. Configurations mentioned below MAY change due to issues or vulnerabilities emerging in the future.

3.1 Transport

All communication from and towards the system as well as internal communication between components located on different platforms MUST be established with *HTTP over TLS*. Thus, external third parties are only allowed to

communicate with the system on port `443`, whereas internal communication runs through port `4223`. Port `80` MUST return HTTP Error `403 Forbidden` and all other ports SHOULD be dropped or blocked.

The key-pair, used in TLS to agree on a mutual symmetric key, MUST be based on either the *RSA* or *DSA* cipher suite, although *RSA* SHOULD be preferred. All created keys MUST have a length of at least `4096` bits. Those ciphers for TLS, that support *Perfect Forward Secrecy* SHOULD be preferred, but ciphers that are not supported by `TLSv1.2` MUST be avoided. All endpoints and other dedicated entry points MUST provide their own generated Diffie-Hellman groups with a minimal length of `4096` bits.

For web-based GUIs *TLS session resumption* SHOULD be activated, but for *endpoints* it MUST be deactivated. Web-based GUIs MUST NOT depend on external resources. All involved assets MUST be stored in the system and thus get served by the *web server*. This required behaviour is enforced by setting the *Content Security Policy (CSP)* in HTTP headers, which also eliminates the risk of Cross-site scripting (XSS) attacks. The *web server* MUST facilitate a web socket connection for web-based GUIs. If a browser does not support this natively, a fallback SHALL be provided by the GUI. Furthermore, those GUIs SHOULD be served with HTTP/2.

The subsequent examples show two *Nginx* configurations for the *web server* component, implementing the specifications from above.

Code 01: Web server configuration for a web-based GUI (excerpt):

```
server {
    server_name gui.system.tld;

    listen 4223 ssl http2;
    listen [::]:4223 ssl http2 ipv6only=on;

    ssl_trusted_certificate /path/to/public-ca.tld.crt.pem;
    ssl_certificate         /path/to/gui.system.tld.crt.pem;
    ssl_certificate_key     /path/to/gui.system.tld.key.pem;
    ssl_dhparam             /path/to/gui.system.tld.dhp.pem;

    ssl_prefer_server_ciphers on;
```

```
    ssl_protocols TLSv1.2;
    ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:
        ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:
        ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:
        ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:
        ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:
        ECDHE-RSA-AES128-SHA256';

    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 5m;

    add_header Strict-Transport-Security "max-age=15768000" always;
    add_header Content-Security-Policy "default-src 'self'";
}
```

Code 02: Web server configuration for a consumer endpoint (excerpt):

```
server {
    server_name CONSUMER_ID.system.tld;

    listen 80;
    listen [::]:80 ipv6only=on;

    return 403;
}

server {
    server_name CONSUMER_ID.system.tld;

    listen 443 ssl http2;
    listen [::]:443 ssl http2 ipv6only=on;

    ssl_trusted_certificate /path/to/system.tld.crt.pem;
    ssl_certificate         /path/to/CONSUMER_ID.system.tld.crt.pem;
    ssl_certificate_key     /path/to/CONSUMER_ID.system.tld.key.pem;
    ssl_dhparam             /path/to/CONSUMER_ID.system.tld.dhp.pem;
```

```
20
21      ssl_verify_client on;
22      ssl_client_certificate  /path/to/CONSUMER_ID.crt.pem;
23
24      ssl_prefer_server_ciphers on;
25      ssl_protocols TLSv1.2;
26      ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:
27          ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:
28          ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:
29          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:
30          ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:
31          ECDHE-RSA-AES128-SHA256';
32
33      ssl_session_cache off;
34
35      add_header Strict-Transport-Security "max-age=15768000" always;
36  }
```

3.2 Authentication

The following two authentication technologies MUST be supported by the system. 2-Factor authentication as an enhancement of the operator authentication procedure is OPTIONAL and can be implemented either by email or with a mobile platform, if it is part of the system.

3.2.1 Transport Layer Security

Before the first consumer tries to register on the system, the system MUST generate a key-pair and sign it by itself. With the resulting certificate, the system becomes a private Certificate Authority primarily responsible for signing certificates that are required for every endpoint and maybe even for connections between mobile and server platforms.
The key-pair for a specific endpoint is then used to issue a certificate based on the *Certificate Signing Request (CSR)* supplied by the consumer who is associated to that very endpoint. The certified certificate and the endpoint's certificate MUST then be transferred back to the consumer on a secure

channel, which the consumer is responsible to provide (e.g. HTTP over TLS certified by trusted public CA, or by a self-signed certificate provided with the registration).

In order to use TLS for bidirectional authentication, not only the client (consumer) MUST be able to verify the server's (endpoint) certificate, but also the server MUST do the same for the consumer. This procedure is known as *two-way authentication*, which is part of the TLS connection establishment. If the connection failed to establish, the authentication has failed. If the connection is successfully established, the consumer is successfully authenticated to the system.

3.2.2 JSON Web Token (JWT)

When creating a JWT, the `"alg"` in the header MUST NOT be set to `"none"`. A JWT MAY be encrypted (JWE). JWTs MUST be created by the *Operator API*. Validation MAY be performed by *Operator API* or *Web server*. Secrets and keys for that purpose are stored in the *Persistence Layer*. As long as the token is not encrypted, every token MUST associate its own secret key for the HMAC computation.

When authenticating, the JWT MUST be provided either as HTTP header (`Authorization: Bearer $JWT`) or as a query parameter indicated by the key `t`. If the operator fails to connect to the system before the token's expiration date has been exceeded, the operator is REQUIRED to login again. The token MUST be renewed, but after at least half of the validity period is reached. The period, in which the token is valid, SHOULD be 24 hours but MUST NOT exceed 48 hours.

The following claims are REQUIRED:

- `"iss"` (Issuer) - domain from which the front end component obtains its data
- `"sub"` (Subject) - front end platform name; MUST be system-wide unique
- `"aud"` (Audience) - `"operator"` or `"contributor"`
- `"exp"` (Expiration Time)
- `"iat"` (Issued At)

- `"jti"` (JWT ID)

One of the following algorithms is REQUIRED (for the `"alg"` header):

- `"HS512"` (key length: 512 bits)
- `"RSA1_5"` (recommended key length: 2048 bits)
- `"RSA-OAEP-256"` (recommended key length: 2048 bits)
- `"A256KW"` (recommended key length: 2048 bits)

3.3 System Architecture

The centralized version of the system architecture places every component on the server platform. Even the web-based management tool, which later is sent to front end platforms, is initially stored there. This means that all the operator's personal data is located somewhere in the 'cloud', probably out of reach, and potentially vulnerable to unauthorized access. Whereas the distributed approach allows, for example, the relocation of the *Personal Data Store* component to a mobile platform, which could also be used by the operator to manage the system. The general approach of a more distributed deployment of components SHOULD reduce the vulnerability for certain scenarios and makes it harder for entities to compromise (parts of) the system. As long as all requirements are met and every component is completely functional, and thus the system as a whole, any component MAY be located on whatever platform is sufficient.

A second approach to gain not only portability, but also to increase the overall level of security is to isolate components [and their process(es)] from the surrounding platform environment. This enables explicit and controlled allocation of resources, such as memory, CPU usage or network and filesystem access. This concept MUST be implemented by either using the process isolation features provided by the host environment, namely *cgroups*, *namespaces*, and *systemd-nspawn*, or by putting components into application containers. The latter is RECOMMENDED and MUST respect all *Open Container* Specifications. An orchestration software MAY be useful to manage all containers.

3.4 Supervised Code Execution *(SCE)*

When running programs on the server platform provided by consumers, it is REQUIRED to execute them only after putting them into an application container. This implies that the container is provisioned first, and then invoked by providing the requested data items as arguments. The container MUST NOT be allowed to access the host's filesystem or network. Before running the container with the actual data, it MUST be executed several times with generated test data. If the program is provided as source code, it MUST be automatically inspected and reviewed. If one of those test layers results are insufficient, processing the access request MUST abort and return with failure information.

3.5 System Monitoring

The *Tracker* component MUST ensure that the following information is being persisted *(required fields for this information is defined in Data and Types)*:

- Access Requests (regardless of its success)
- Failed Access Verifications
- Registrations of consumers (regardless of its success)
- Results of operator Authentications
- Permission Profile creation, manipulation, and deletion
- SCE (regardless of its success)
- Any third party request attempt arriving at the web server
- Server Resources (continually)

To make sure that these data are collected, other components MUST provide the *Tracker* with such information. Therefore, components such as *Operator API*, *Permission Manager* and *Web server*, MUST push information towards the *Tracker*.
By performing pattern recognition & anomaly detection, the *Tracker* is then able to recognize abnormal behaviour or occurrences, for example, by monitoring the IP of an access request origin, which normally should not change very often. Such data MAY also help to prevent spam requests. If the

Tracker finds suspicious patterns, the operator MUST be notified via email and push notification.

4 Protocols

The following protocols reflect core procedures. They have to be understood as detailed instructions on how these procedures MUST be implemented.

Consumer Registration *Before a third party is permitted to request data access, it must first register to the system. As a result, the third party becomes data consumer and is provided with a dedicated endpoint, to which it MUST authenticate by presenting its certificate signed by the system.*

Preconditions:

- secure channel; QR-Code (by *third party*), HTTPS (by *data subject*)
- key-pair and CSR (on the consumer-side)
- unique URL created by *data subject* with the help of the system's management tool

0. [OPTIONAL] *data subject* provides *third party* with unique URL
1. *third party* creates registration request that includes
    - information about itself and reasons for registration attempt
    - X.509 based Certificate Signing Request (CSR)
    - callback URL via HTTPS as feedback channel
    - [OPTIONAL] information about what data items are wanted to be accessible
2. depending on (0), *third party* provides *data subject* with registration request either as QR-Code or via HTTPS by given URL
3. [OPTIONAL] *data subject* gets notified about new registration request (REQUIRED if no mobile platform is associated to the system)
4. *data subject* reviews registration and decides to either accept or refuse the registration

- *Accept*
  1) create new *endpoint*
     - create new entry in `endpoints` (PL) by providing registration information (csr, info, callback URL etc.),
     - generate consumer identifier
     - register new subdomain in *web server* with consumer identifier (e.g. `CONSUMER_ID.system.tld`)
  2) issue certificate
     - generate new key-pair and certificate for registered subdomain (CN: `CONSUMER_ID.system.tld`) and sign it with the system's root certificate
     - process CSR accordingly, sign it with the subdomain's certificate and store it in the related entry within `endpoints`
  3) [OPTIONAL] if registration contains information on what data is requested to get permission to access, that information MUST be processed as described in Permission Request
  4) respond issued third party certificate, endpoint's certificate and [OPTIONAL] result(s) of the Permission Request procedure
- *Refuse*
  1) *data subject* SHOULD provide reason or MUST fall back to default reason
  2) respond with reason
- *Failure*
  - subsequent processes decide on their own what error message and how much information is provided to the third party
  1) form proper error identifier and message
  2) add unique URL to submit a new registration
  3) respond with error

5. assemble and submit response via provided callback URL

6. *third party* is informed about the decision via callback channel and proceeds responded data accordingly

   - *Accept*

1) install certificates and [OPTIONAL] pin provided *endpoint* certificate
2) [OPTIONAL] recognize and process result(s) of the Permission Request

- *Refuse: unspecified*
- *Failure*

1) [OPTIONAL] depending on error, act accordingly (e.g. create a new registration and submit it again)

Result(s):

- *third party* becomes *data consumer* and is now able to request permissions
- *data consumer* MAY now be permitted to access data, if registration has contained information for Permission Request

Permission Request *In order for a data consumer to access data that is provided by the system, he MUST first request those permission(s).*

Preconditions:

- third party must be registered as *data consumer* to the system
- *consumer* must know very precisely what data items he wishes to access

1. *consumer* creates permission request and submits it to his dedicated *endpoint* provided by the system
2. *operator* gets notified about new permission request
3. *operator* reviews requested permission

- *Accept*
  1) *operator* creates new *Permission Profile* by either applying a prepared template/draft, importing configurations from an existing profile, or filling out an empty, or pre-filled with provided information, profile When saving the new profile, it gets associated to the requester's *endpoint*
  2) respond which data items are permitted to be accessed, how

often, and how long this permission lasts

- *Refused*
  1) applying the provided information as is, the system creates and stores a permission profile after everything, which is still associated with the requester's *endpoint* but flagged as `refused`
  2) respond [OPTIONAL] with reason
- *Failure*
  1) form proper error identifier and message based on the error that has occurred
  2) respond with error

4. assemble and submit response to *consumer*
5. *consumer* is informed about the decision and acts accordingly
   - *Accept*
     1) apply provided information to subsequent *access requests*
   - *Refuse: unspecified*
   - *Failure*
     1) [OPTIONAL] depending on error, act accordingly (e.g. review and adjust permission request)

Result(s):

- *consumer* is aware of what data he is allowed to access
- *consumer* is able to access data
- *operator* holds documentation about who has access to which of her data items
- *operator* it able to regulate access permission for the *consumer* at any point in time

Access Request *After requesting permission to access data and having this granted, a consumer actually access data items by either getting them forwarded or by providing a program that is invoked with the data items as arguments. The result of that invocation is then sent back to the originating*

data consumer.

Preconditions:

- successfully requested permission
- [OPTIONAL] program to be invoked

1. after successfully authenticated, *consumer* submits access request, containing
   - query (defines data and its structure)
   - access type (plain forward or program invocation)
   - [OPTIONAL] response method (`keepalive` or `push`)
   - [OPTIONAL] program (source or binary)
2. *Permission Manager* tries to verify the access (see Access Verification) with the following possible results
   - *allowed*
     1) depending on defined response method, either keep session open until timeout limit exceeds, or respond with unique process identifier and [OPTION] estimated duration
   - *denied*
     1) abort processing access request
     2) respond with default denial notice or [OPTIONAL] message from *Access Verification*
3. obtain data from *PDS*
4. [OPTIONAL and if reliable == `true`] verify and indicate data reliability
   1) check if data item(s) in obtained data are in group of data items whose reliability can be verified
      - *yes*, proceed with configured method for checking reliability
        - `null`
          1) abort reliability verification and move on to the next step
        - `'qes'`
          1) notify operator that a signing procedure, involving her

eID card, is required
2) go through signing procedure and return certificate
– `'gov'`
1) check if certificate is not expired
2) compare fingerprint in certificate with calculated fingerprint *) if either fails and `notfyIfNotReliable == true`, pause and await operator's decision, otherwise move on to next step
- *no*, then move on to next step

5. adjust data precision
   - requesting lower precision than approved by *data subject* is always possible, but not the other way around
6. compute result according to provided access type
   - *forward*
     1) add expiration date for the data
     2) move all obtained data to response handler
   - *supervised execution*
     0) copy program into *PL*
     1) inspect and review code, if available
     2) provision container runtime with dependencies and provided program
     3) run multiple tests with generated test data *) if threshold of failed tests is exceeded, abort and pass error message on to response handler
     4) run with real data
     5) forward result to response handler
7. finalize response and submit it back to *consumer*

Result(s):

- in case of *supervised execution*, no actual personal data has left the system
- if *consumer* would have been denied access to data, they would not be accessed

Access Verification *When an access request is received, first it MUST be verified if that access is even permitted according to the permission profiles and the presented data query, before the request is processed and data is obtained from the PDS.*

Preconditions:

- query string
- endpoint name (or consumer identifier)
- options contained in the access request (e.g. access type)

1. gather all *permission profiles* relating to the given endpoint
2. find all profiles that
   a) address at least one requested data item and
   b) have a valid *permission type* at that moment
3. check resulting subset and if …
   - it is empty
     1) return with a negative result (`false`)
   - not all requested data items are addressed among those profiles
     1) pause processing
     2) notify operator and ask for decision on whether access to unregulated data items is allowed or denied
     3) after missing data items are acknowledged by operator, go back to (2.)
   - all requested data items are addressed among those profiles and …
     - at least one is not allowed to access (including `refused` profiles)
       1) return with a negative result (`false`) and [OPTIONAL] include information on which data items are affected
       2) inform operator about this incident
     - all are allowed to access
       1) return with a positive result (`true`)

Result(s):

- only explicitly permitted access requests are able to succeed
- violation of existing access rules is reported

5 Data & Types

The system is aware of two different classes of data - personal data and system data. The latter are stored within the *Persistence Layer* and represent the current state of the system, whereas all the personal data, which reflects the *Digital Identity* of the operator, is maintained and also provided by the *Personal Data Store*.

5.1 Personal Data

The *PDS* is portable and can be placed on any of the platform types supported by the system, whereas data queries typically originate from a server platform. In order to provide such flexibility, and unless editing personal data happens on the same platform instance on which the data is also stored, access to the *PDS* MUST be abstracted. This design approach can be implemented by utilizing GraphQL's native architecture, which is - aside from its generic graph-structured query language - primarily characterized by its separation of validation and execution. The execution layer requires adapters for every database system the PDS has to support, while the validation is agnostic regarding the query origin. In other words, the underlying database system can be swapped, or various database systems can run side by side. A universal query language also has the advantage of being effortlessly reapplicable to a different storage system and being able to revert every data change. Therefore it is REQUIRED to store every write query chronologically, next to the current state of the personal data.

Additional requirement(s) are:

a) precision of data accessed by consumers must be adjustable (e.g. cut fractional digits, decrease sampling rate of time series datasets)

b) store and fetch binary data

Suitable Areas or Scenarios of Application (excerpt):

- core/master/profile data of an individual
- biometric data (e.g. finger print, retina)
- financial record (and history)
- sensor data (e.g. geo-location, motions, IoT)
- payment information
- medical record
- governmental data
- user-generated content (blog posts, comments, pictures, videos, etc.)
- web search or browsing history
- consume behaviors (e.g. watchlist, music playlist)

5.2 System Data

The *PL* consists of multiple storage technologies and MUST be placed on a trusted platform type - a server. At least two technologies MUST be supported: the platform's filesystem, which SHOULD be accessible by other components on the same platform, and a document-oriented storage system that provides high flexibility through a schema-free approach and MUST also be shared among several components. It is OPTIONAL, if the *Tracker* uses the same storage system, multiple ones or a completely different storing mechanism. Regardless, they are all unified under the *Persistence Layer* component.

Additional requirement(s) are:

a) configurations and application data (e.g. permission profile) MUST be reversible

5.3 Structure & Types of Personal Data

The type system, on which all personal data is based, MUST be compatible with the type system provided by GraphQL. The subsequent definitions represent the minimum feature set that a type system for personal data MUST acknowledge in any implementation.

`[primitive]` most basic or atomic data type, consisting of just a single value that is either defined as `String`, `Boolean`, `Integer`, `Float` or `null`, which are `primitives` by themselves; *(corresponding to GraphQL's `Scalar`)*

`[struct]` combines multiple types in order to define more complex types; typically composed of `primitives` and organized as (nested) *Object(s)* or *List(s)*; can consist of other structs as well

The *Storage Connector* MUST support a basic set of `primitives` and `structs` right from the start, in order to reduce the effort when making first steps with the system. The following types are initially REQUIRED:

Primitives:

- `ID`
- `Date`
- `Email`
- `Domain`
- `PhoneNumber`
- `URL`

Structs:

- `Position`
- `File`
- `Address`
- `Unit`
- `Product`
- `Diseases`
- `Treatment`
- `Organisation`
- `Route`
- `CV`
- `Contact`
- `Transaction`

Other *structs* that might be needed in the future, can either be created and

modeled by the operator herself, or are developed and provided by community members and other third parties, so that unsupported types just need to be imported into the system.

5.4 Data Models for System Data

Aside from personal data, whose structure is adjustable by the *data subject* to suit her needs, the system itself REQUIRES some data models too, in order to provide i.a. mechanisms for managing the process of accessing data. Those models are outlined below.

The types that MUST be supported by the *PL* are similar to the *primitives* available in the Query Language, but somewhat more rudimentary: `String`, `Boolean`, `Number`, and `null`. Everything beyond this depends on the technologies that are being used.

NOTICE: Required fields are implicit and therefore not marked as such. Any other notation is explicit, though. The minimum viable types are indicated; if supported, a more precise one SHOULD be used.

5.4.1 Endpoint

- `name: String` - full valid subdomain name
- `consumer: String || Object` - consumer information || name
  - `name: String` - readable and descriptive consumer name
  - `description: String` - [OPTIONAL] consumer description provided during registration
- `crt: String` - endpoint's certificate (filesystem path or file content)
- `key: String` - endpoint's private key (filesystem path or file content)
- `ccert: String` - consumer's certificate (filesystem path or file content)
- `createdAt: Number` - date of creation of this endpoint

5.4.2 Permission Profile

- `data: [String]` - [MUST if `query == undefined`] list of permitted data endpoints
- `query: Object || String` - [MUST if `data == undefined`] query,

representing permitted data
- `endpoint: String` - associating endpoint
- `type: String` - how long the profile is valid
  - `: one-time-only`
  - `: expires-on-date`
  - `: until-further-notice`
- `expiresAt: Number` - [MUST if type == `expires-on-date'`] date of expiration
- `interval: Object` - least amount of time between two access requests
  - `value: Number` - time value
  - `unit: String` - unit of time value
- `createdAt: Number` - date of creation of this profile
- `disbaled: Boolean` - [OPTIONAL] disabled until further notice
- `access: String` - [OPTIONAL] access type: `fwd` (plain forward) or `sce` (supervised code execution)
- `dataExpiration: Number` - [SHOULD if access == `'fwd'`] date of when responded data becomes outdated and thus unreliable

5.4.3 Notification

NOTICE: MUST be encrypted, if is sent via Push Notification Service

- `oid: String` - identifier of what this notification has triggered
- `type: String` - type of procedure/occasion
- `message: String` - information, that is presented to operator
- `payload: String` - anything, that can be serialized into a string (e.g. JSON, containing information about a related procedure)

5.4.4 Tracker

Collection: `registrations`

- `id: String` - identifier of this document
- `csr: String` - certificate signing request
- `callback: String` - callback URL for submitting registration response
- `crt: String` - [OPTIONAL] certificate for callback URL, if not pub-

licly trusted
- `pr: Boolean` - was permission request included?
- `header: String` - third party's HTTP header
- `tsin: Number` - timestamp of occurrence

Collection: `access-requests`

- `id: String` - identifier of this document
- `tsin: Number` - timestamp of occurrence
- `tsout: Number` - timestamp of response
- `type: String` - access type: `fwd` (plain forward) or `sce` (supervised code execution)
- `program: String || Object` - [MUST if `type == 'sce'`] filesystem path or file content that facilitates supervised code execution
- `data: [String]` - [MUST if `query == undefined`] list of permitted data endpoints
- `query: Object || String` - [MUST if `data == undefined`] query representing permitted data
- `state: String` - current state of proceeding
  - : `'handling'` - pre-process access request
  - : `'verifying'` - verify against existing permission profiles
  - : `'obtaining'` - query data from *PDS*
  - : `'adjusting'` - apply precision rules
  - : `'responding'` - hand over generated result back to consumer
- `endpoint: String` - endpoint on which the request came in
- `header: String` - consumer's HTTP header
- `result: String` - [MUST if access == `'sce'`] serialized result of *SCE*
- `responseMethod: String` - how the system should submit he response to the consumer (`keepalive` or `push`)
- `verifyReliability: Boolean` - indicate in response, that obtained data is reliable/authentic

Collection: `failed-access-verifications`

- `id: String` - identifier of this document
- `requestId: String` - reference to `access-request` ID
- `reason: String` - why has the verification failed (e.g. error or refused message)

- `ts: Number` - date of failure

Collection: `tokens`

- `id: String` - identifier of this document
- `type: String` - purpose/audience of the token (values: `operator` or `collector`)
- `secret: String` - secret that is used to encrypt the token
- `algorithm: String` - which algorithm is used to encrypt the token
- `deviceId: String` - unique identifier of the authorized device
- `createdAt: Number` - date of creation
- `renewedAt: Number` - date of last renewal
- `expiresAt: Number` - date of expiration

5.4.5 System Configurations and Defaults

Table 7.1: Global System Configurations and their default values

Key	Primitive	Default	Description
`dataExpiration`	Number	*now + 48 hours*	see *Permission Profile* above
`accessResponseMethod`	String	`'push'`	see *Collection:* `access-requests`
`accessResponseTimeout`	Number	`120`	timeout of access response in sec.
`accessType`	String	`'sce'`	see *Permission Profile* above
`programMaxSize`	Number	`20480`	size of program for SCE (in Kbyte)
`notifyInGui`	Boolean	`true`	notify in management tool
`notifyByMail`	Boolean	`true`	notify via email
`notifyOnRegistration`	Boolean	`true`	notify on incoming registration
`notifyOnPermission`	Boolean	`true`	notify on incoming permission request

Key	Primitive	Default	Description
`notifyOnAccess`	Boolean	`false`	notify on incoming access request
`notifyOnViolation`	Boolean	`true`	notify on violations of access rules
`notifyOnAnomaly`	Boolean	`true`	notify if *Tracker* recognizes anomaly
`notifyOnError`	Boolean	`false`	notify on unexpected errors in system
`notfyIfNotReliable`	Boolean	`false`	notify on failed reliability check
`reliabilityCheckMethod`	String	`null`	method to check data reliability

6 Interfaces

6.1 Graphical User Interfaces (GUIs)

The primary GUI of the system is the operator's *Management Tool*, which is used to administrate the system, to control and manage data access, and to maintain personal data. The tool MAY be available on mobile platforms, but MUST at least be provided for *web* platforms. The following lists show briefly, which features the tool provides to the operator.

6.1.1 Functionality, that MUST be provided:

- Operator authentication
- Managing consumers and their entry point(s)
- Managing permission profiles
- Reviewing registrations and permission requests
- Maintenance of personal data
- Responsive screen adaption
- Low-latency view rendering
- Changing system settings

- Notifications
- Registration of new front end platforms to the system

6.1.2 Functionality, that SHOULD be available:

- Personal data import
- History of access requests
- Reviewing permission requests
- Mechanisms to make backups
- Filter mechanisms for permission profiles, access history
- Creating & Managing templates for permission profiles
- QR-Code scanning (mobile platform only)

6.1.3 Functionality, that MAY be supported:

- Reverting changes in permission profiles
- System monitoring and statistics
- Personal data export
- Utilization of the *accordion* pattern for (re)viewing data structures
- Modeling new data structures
- Migration of the *PDS* to another platform

6.2 Application Programming Interface (APIs)

The lists of GUI features implicitly define what the *Operator API* must be capable of, whereas interactions originated by third parties and data consumers are described below. These descriptions show the behavior of all major API endpoints that are required to access personal data, which is hosted by the system.
The payload MUST be transmitted with HTTP requests, secured by TLS, declared as *POST* method, unless otherwise outlined. It MUST be serialized in JSON and constitute the complete request body. When sending data originating from certain formats (e.g. PEM-formatted file content or binary streams) as string values that are a part of a URL or JSON, *base64url* MUST be used for encoding. Furthermore, it is to be noted, that requests are asynchronous and MAY take several hours up to a few days to be answered.

Registration Request MUST be handed over to the system in order to be acknowledged as a *data consumer*. It can either be submitted via HTTP to the system, if the operator has provided a URL, or encoded as a QR-Code presented on web page, served via HTTPS by the requester, ready to be scanned with the operator's mobile device.

Parameter(s):

- `csr: String` - certificate signing request
- `cb: String` - callback URL (REQUIRES https)
- `cert: String` - [OPTIONAL] certificate for callback URL, if not publicly trusted
- `desires: [String] || String` - [OPTIONAL] list of data item selectors or query string (see Registration Request)

Request: `https://system.tld/register/$uniqueRndValue`

```
{
    "csr": "$base64UrlEncodedCsr",
    "cb": "https://third-party.tld/callbacks/$procedureIdentifier",
    "desires": [
        "profile.lastname",
        "profile.firstname",
        "finance.bankAccounts"
    ]
}
```

Response:

- `endpoint: String` - URL of the consumer-specific endpoint
- `cert: String` - certificate of the consumer-specific endpoint
- `type: String` - permission type (see Permission Profile)
- `expiration: Number` - [MUST if type == `expires-on-date'`] date of expiration
- `interval: Object`
  - `value: Number` - time value
  - `unit: String` - unit of time value
- `grants: [String]` - list of data items allowed to be accessed

```
{
    "endpoint": "https://$endpointId.system.tld",
    "cert": "$base64UrlEncodedCert",
    "permissions": {
        "type": "expires-on-date",
        "expiration": $twoWeeksFormNow,
        "interval": {
            "value": 1,
            "unit": "daily"
        }
        "datapoints": [
            "profile.lastname",
            "profile.firstname"
        ]
    }
}
```

Permission Request MUST create a new *permission profile* and thus MAY enable the data consumer to access personal data. In return, the consumer is provided with all information that is required to request data access based on the permissions requested herewith.

NOTICE: If the representation of the data to which access is requested is defined as a list of strings (`[String]`), then the response MUST show a similar structure; the same applies to the data query for string-representations (`String`).

Parameter(s):

- `desires: [String] || String` - list of data items selectors or query string, representing what data is requested to get access to

Request: `https://$endpointId.system.tld/pr`

```
{
    "desires": "{profile{firstname,lastname},finance{bankAccounts}}"
}
```

Response:

- `type: String` - permission type (see Permission Profile)
- `grants: String` - data query representing the data items allowed to be accessed

```
{
    "type": "one-time-only",
    "grants": "{profile{firstname,lastname}}"
}
```

Access Request leads to the actual data access, if the *Access Verification* does not reject access.

Parameter(s):

- `query: String` - data query representing the data that is requested to be accessed
- `type: String` - [OPTIONAL] access type: `fwd` or `sce`
- `program: Object` - [MUST if `type == 'sce'`]
  - `mimeType: String` - MIME-Type (see RFC 2046[20])
  - `extension: String` - file extension
  - `file: String` - base64url-encoded file content
- `reliable: Boolean` - [OPTIONAL] indicates whether the consumer expects a proof of data reliability or not
- `respond: String` - [OPTIONAL] `push` (returns immediately with information where to obtain the result) or `keepalive` (keep connection open until response is computed)

Request: `https://$endpointId.system.tld/ar`

```
{
    "type": "fwd",
    "reliable": true,
    "respond": "keepalive",
    "query": "{profile{firstname,lastname}}"
}
```

Response: [if `respond == 'push'`]

[20] https://tools.ietf.org/html/rfc2046

- `pickup: String` - URL where to find the actual response
- `duration: Number` - estimated time in seconds until response SHOULD be available

```
{
    "pickup": "https://$endpointId.system.tld/ar/$procedureId",
    "duration": 43200,
}
```

Request: `https://$endpointId.system.tld/ar/$procedureId` *Response:*

- `expiresAt: Number` - date when data become outdated
- `proof: String` - [OPTIONAL] certificate that indicates verified data reliability
- `data: Object || String` - if valid JSON, then Object, otherwise base64url-encoded string

```
{
    "expiresAt": $oneMonthFromNow,
    "proof": "$reliabilityVerificationCertificate",
    "data": {
        "profile": {
            "firstname": "Jane",
            "lastname": "Doe"
        }
    }
}
```

7 Recommendations & Resources

This section provides additional information on hosting and operating such a system, as well as further resources that MAY be worth knowing. Moreover, it contains advice to consider when implementing (parts of) this specification.

Host Environment(s) Recommendations

- Depending on the technologies that are being used for the implementations, the environment specifications MAY vary. Although,

it SHOULD be feasible to expect at least around *2* cores, *4GB* of memory and *60GB* of storage.

- Unless server components are running in containers, it is not recommended to run other applications in the same environment side by side.
- A vendor, who hosts for multiple data subjects, one system instance each, SHOULD NOT locate all of them in one environment, but rather deploy a distinct operating system (virtualization) for every instance, or separate them physically on different machines.
- The host environment (for the system's server components) SHOULD offer measures for detecting security violations and breaches.

Recommendations To Third Parties

- If data consumers are accessing data from more than one system instance, representing different individuals, it MAY be wise to use different key-pairs and certificates for all instances, and probably even unique endpoints.
- Data consumers SHOULD also provide a full-featured *endpoint* based on a subdomain, thereby a bidirectional communication flow can be established, instead of the workaround based on callback URLs.

Software Recommendations

- letsencrypt (ACME implementation)
- nginx (web server and load balancer)
- systemd (process management)
- rkt (OCI implementation)
- Kubernetes (container orchestration
- LibreSSL (cryptographic software library)
- CouchDB (document-oriented database system focusing on real-time)
- RethinkDB (graph-structured database system)
- Prometheus (database system optimized for time series data)
- Container Linux (open-source lightweight operating system, by

CoreOS)

Resources

- SSL and TLS Deployment Best Practices[21]
- Security/Server Side TLS[22]
- Strong SSL Security on nginx[23]
- GraphQL Specification - 3.1 Types[24]

[21]https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices
[22]https://wiki.mozilla.org/Security/Server_Side_TLS
[23]https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html
[24]https://facebook.github.io/graphql/#sec-Types

Source Code

Code 01: Example query in SPARQL:

```
# query 1: obtain the first and last name fof data subject
PREFIX person: <http://reflowd.tld/schemas/person>

SELECT $firstname $lastname
FROM <https://unique-consumer-endpoint.reflowd.tld/sparql/profile>
WHERE {
    $person person:firstname $firstname .
    $person person:lastname $lastname .
}


# query 2: obtain all bank accounts that are available for
# online payment
PREFIX bank-account: <http://reflowd.tld/schemas/bank-account>

SELECT $accountId $bankName $paymentMethod
FROM <https://unique-consumer-endpoint.reflowd.tld/sparql/finance>
WHERE {
    $bank-account bank-account:payment-method "online-service" .
    $bank-account bank-account:payment-method $paymentMethod .
    $bank-account bank-account:account-id $accountId .
    $bank-account bank-account:bank-name $bankName .
}
```

Code 02: Results of Code 01 in JSON:

```
// result 1:
{
    "head": {
        "vars": [
            "firstname",
            "lastname"
        ]
    },
    "results": {
        "bindings": [
            {
                "firstname": {
                    "type": "literal",
                    "value": "Doe"
                },
                "lastname": {
                    "type": "literal",
                    "value": "Jane"
                }
            }
        ]
    }
}

// result 2:
{
    "head": {
        "vars": [
            "accountId",
            "bankName",
            "paymentMethod"
        ]
    },
    "results": {
        "bindings": [
            {
                "accountId": {
```

```
                    "type": "integer",
                    "value": 0905553715
                },
                "bankName": {
                    "type": "literal",
                    "value": "A. W. Fritter Institute"
                },
                "paymentMethod": {
                    "type": "literal",
                    "value": "online-service"
                }
            }
        ]
    }
}
```

Code 03: Example query in GraphQL:

```
# URL: https://unique-consumer-endpoint.reflowd.tld/graphql

query {
    profile {
        firstname
        lastname
    }
    bankAccounts(paymentMethod: 'online-service') {
        accountId
        bankName
        paymentMethod
    }
}
```

Code 04: Result of Code 03 in JSON:

```
{
    "profile": {
        "firstname": "Jane",
        "lastname": "Doe"
    },
    "bankAccounts": [
        {
            "accountId": 0905553715,
            "bankName": "A. W. Fritter Institute",
            "paymentMethod": "online-service"
        }
    ]
}
```

NOTICE: schema notation is based on the rules underpinning the schema definition provided by the SimpleSchema project [162]

Code 05: Struct - Profile (example)

```
{
    firstname: String,
    lastname: String,
    pseudonym: String,
    birth: Date,
    gender: String,
    religion: String,
    motherTongue: Language
    photo: File,
    residence: Address,
    employer: Organisation
}
```

Code 06: Struct - Contact (example)

```
{
    label: String,
    type: String('phone'|'email'|'url'|'name-of-social-network'),
    prio: Integer(0-2),
    uid: String
}
```

Code 07: Struct - Position (example)

```
{
    lat: Float,
    lon: Float,
    radius: {
        value: Float,
        unit: Distance
    },
    description: String
    ts: Date
}
```

References

[1] *Big data privacy international.* URL https://www.privacyinternational.org/node/8. - retrieved 2016-11-15

[2] Pedreshi, Dino ; Ruggieri, Salvatore ; Turini, Franco: Discrimination-aware data mining. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* : ACM, 2008, pp. 560–568

[3] Spiekermann, Sarah: *Ethical IT Innovation: A Value-Based System Design Approach* : CRC Press; Taylor & Francis Group, LLC, 2015 — ISBN 978-1-4822-2635-5

[4] Friedman, Batya ; Nissenbaum, Helen: Bias in computer systems. In: *ACM Transactions on Information Systems (TOIS)* vol. 14 (1996), Nr. 3, pp. 330–347

[5] *Cognitive bias.* URL https://en.wikipedia.org/w/index.php?title=Cognitive_bias&oldid=742803386. - retrieved 2016-11-08. — Wikipedia. — Page Version ID: 742803386

[6] Lemov, Rebecca: *Why big data is actually small, personal and very human. Aeon essays.* URL https://aeon.co/essays/why-big-data-is-actually-small-personal-and-very-human. - retrieved 2016-11-17

[7] Dewes, Andreas: *C3TV - Say hi to your new boss: How algorithms might soon control our lives.* URL https://media.ccc.de/v/32c3-7482-say_hi_to_your_new_boss_how_algorithms_might_soon_control_our_

lives#video&t=1538. - retrieved 2016-11-03

[8] Hong, Jason I.; Landay, James A.: An architecture for privacy-sensitive ubiquitous computing. In: *Proceedings of the 2nd international conference on mobile systems, applications, and services* : ACM, 2004, pp. 177–189

[9] *ProjectVRM - about. ProjectVRM*. URL https://blogs.harvard.edu/vrm/about/. - retrieved 2016-11-09

[10] Tom Kirkham; Sandra Winfield; Serge Ravet; Kellomaki, Sampo: The personal data store approach to personal data security. In: *IEEE Security & Privacy* vol. 11. Los Alamitos, CA, USA, IEEE Computer Society (2013), Nr. 5, pp. 12–19

[11] Poikola, Antti; Kuikkaniemi, Kai; Honko, Harri: MyData – a nordic model for human-centered personal data management and processing, Ministry of Transport; Communications (2015), pp. 1–12 — ISBN 978-952-243-455-5

[12] *Meeco how it works*. URL https://meeco.me/how-it-works.html. - retrieved 2016-11-09

[13] *Open specification of the concept called data reservoir flow control (reflowd). GitHub*. URL https://github.com/lucendio/reflowd_spec. - retrieved 2016-11-11

[14] USA, Federal Trade Commission: *Data brokers*, 2014

[15] Rose, John; Rehse, Olaf; Röber, Björn: The value of our digital identity. In: *Boston Cons. Gr* (2012), p. 36

[16] Regulation (EU) 2016/679 — General data protection regulation, 2016

[17] Wikipedia: *Information privacy law*. URL https://en.wikipedia.org/wiki/Information_privacy_law#United_States. - retrieved 2016-11-20. — Page Version ID: 749338152

[18] Loeb), Ieuan Jolly (Loeb &: *PLC - data protection in the united states: Overview*. URL http://us.practicallaw.com/6-502-0467. - retrieved

2016-11-20

[19] Wilhelm, Alex: *White house drops "consumer privacy bill of rights act" draft. TechCrunch.* URL http://social.techcrunch.com/2015/02/27/white-house-drops-consumer-privacy-bill-of-rights-act-draft/. - retrieved 2016-11-20

[20] Consumer privacy bill of rights act (cpbora) — Administration discussion draft: Consumer privacy bill of rights act of 2015, 2015

[21] FCC 16-148 — Report and order, 2016. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[22] FCC 16-39 — Notice of proposed rulemaking, 2016. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[23] *Privacy policies are mandatory by law.* URL https://termsfeed.com/blog/privacy-policy-mandatory-law/. - retrieved 2016-11-20. — Disclaimer: Legal information is not legal advice

[24] Facebook: *facebook - creating an account.* URL https://www.facebook.com/. - retrieved 2016-11-20

[25] *International privacy standards.* URL https://www.eff.org/issues/international-privacy-standards. - retrieved 2016-11-20

[26] Jan Philipp Albrecht, MdEP: *EU-US "privacy shield" - background and frequently asked questions (faq).* URL https://www.janalbrecht.eu/themen/datenschutz-digitalisierung-netzpolitik/eu-us-privacy-shield-2.html. - retrieved 2017-02-06

[27] Dachwitz, Ingo: *Nationale datenschutzbehörden kritisieren privacy shield und kündigen umfassende prüfung an.* URL https://netzpolitik.org/2016/nationale-datenschutzbehoerden-kritisieren-privacy-shield-und-kuendigen-umfassende-pruefung-an/. - retrieved 2017-02-06

[28] Rosner, Gilad: Who owns your data? In: : ACM Press, 2014

— ISBN 978-1-4503-3047-3, pp. 623–628

[29] Grunebaum, J.O.: *Private ownership*, *Problems of philosophy* : Routledge & Kegan Paul, 1987 — ISBN 9780710207067

[30] Regulation (EU) 2016/679 — General data protection regulation, 2016

[31] FCC 16-148 — Report and order, 2016. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[32] Facebook: *Facebooks's terms of service. Statement of rights and responsibilities*. URL https://www.facebook.com/legal/terms. - retrieved 2016-12-01

[33] Twitter: *Twitters's terms of service. Twitter terms of service*. URL https://twitter.com/tos#intlTerms. - retrieved 2016-12-01

[34] Google: *Google's terms of service. Google terms of service*. URL https://www.google.com/intl/en/policies/terms/regional.html. - retrieved 2016-12-01

[35] Apple: *Apple's iClound terms and conditions. V. content and your conduct*. URL https://www.apple.com/legal/internet-services/icloud/en/terms.html. - retrieved 2016-12-01

[36] *Why metadata matters*. URL https://www.eff.org/deeplinks/2013/06/why-metadata-matters. - retrieved 2016-11-24

[37] Stevens, John P.: *Why you need metadata for big data success*. URL http://www.datasciencecentral.com/profiles/blogs/why-you-need-metadata-for-big-data-success. - retrieved 2016-11-24

[38] *Big data n.* URL http://www.oed.com/view/Entry/18833#eid301162177. - retrieved 2016-11-11

[39] Wikipedia: *Big data*. URL https://en.wikipedia.org/w/index.php?title=Big_data&oldid=748964100. - retrieved 2016-11-11. — Page Version ID: 748964100

[40] Cios, Krzysztof J. ; Swiniarski, Roman W. ; Pedrycz, Witold ; Kurgan, Lukasz A.: The knowledge discovery process. In: *Data mining* :

Springer, 2007, pp. 9–24

[41] Marbán, Óscar ; Mariscal, Gonzalo ; Segovia, Javier: A data mining & knowledge discovery process model. In: *Data mining and knowledge discovery in real life applications*. Madrid, Span : InTech, 2009

[42] Dewey, Caitlin ; Dewey, Caitlin: 98 personal data points that facebook uses to target ads to you. In: *The Washington Post* (2016)

[43] Taie, Mohammed Zuhair Al: *Big data: Types of data used in analytics. Agroknow blog*. URL http://blog.agroknow.com/?p=4690. - retrieved 2017-02-08

[44] Zaïane, Osmar R: Chapter 1: Introduction to data mining. In: *Principles of knowledge discovery in databases* : Department of Computing Science, University of Alberta, 1999. — CMPUT 690, pp. 1–2

[45] *Big data collection collides with privacy concerns, analysts say. PCWorld*. URL http://www.pcworld.com/article/2027789/big-data-collection-collides-with-privacy-concerns-analysts-say.html. - retrieved 2016-11-15

[46] *Answers.io. Answers*. URL https://answers.io/answers. - retrieved 2016-11-14

[47] Burgelman, Author: Luc ; Burgelman, NGDATA Luc ; NGDATA: *Attention, big data enthusiasts: Here's what you shouldn't ignore. WIRED*. URL https://www.wired.com/insights/2013/02/attention-big-data-enthusiasts-heres-what-you-shouldnt-ignore/. - retrieved 2016-11-15. — Partner Content

[48] Laney, Douglas: *3D data management: Controlling data volume, velocity, and variety* : META Group, 2001

[49] Hilbert, Martin: Big data for development: A review of promises and challenges. In: *Development Policy Review* vol. 34 (2015), Nr. 1, pp. 135–174

[50] Davis Kho, Nancy: *The state of big data*. URL http://www.econtentmag.com/Articles/Editorial/Feature/The-State-of-Big-Data-108666.htm. - retrieved 2016-11-18

[51] CEO), Tim Cook (Apple's: *A message to our customers. Customer*

letter. URL http://www.apple.com/customer-letter/. - retrieved 2016-11-18

[52] Green, Matthew: *What is differential privacy? A few thoughts on cryptographic engineering*. URL https://blog.cryptographyengineering.com/2016/06/15/what-is-differential-privacy/. - retrieved 2016-11-18

[53] Budington, Bill: *WhatsApp rolls out end-to-end encryption to its over one billion users*. URL https://www.eff.org/deeplinks/2016/04/whatsapp-rolls-out-end-end-encryption-its-1bn-users. - retrieved 2016-11-18

[54] *The next rembrandt: Blurring the lines between art, technology and emotion. Microsoft news centre europe*. URL https://news.microsoft.com/europe/features/the-next-rembrandt-blurring-the-lines-between-art-technology-and-emotion-2/. - retrieved 2017-02-08

[55] *Stuck in traffic? Insights from googlers into our products, technology, and the google culture*. URL https://googleblog.blogspot.com/2007/02/stuck-in-traffic.html. - retrieved 2016-11-18

[56] Wikipedia: *Google traffic*. URL https://en.wikipedia.org/w/index.php?title=Google_Traffic&oldid=746200591. - retrieved 2016-11-18. — Page Version ID: 746200591

[57] *Global mobile OS market share*. URL https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. - retrieved 2016-11-18

[58] Ao, Ji ; Zhang, Peng ; Cao, Yanan: Estimating the Locations of Emergency Events from Twitter Streams. In: *Procedia Computer Science* vol. 31 (2014), pp. 731–739

[59] Shi, Weiwei ; Zhu, Yongxin ; Zhang, Jinkui ; Tao, Xiang ; Sheng, Gehao ; Lian, Yong ; Wang, Guoxing ; Chen, Yufeng: Improving power grid monitoring data quality: An efficient machine learning framework for missing data prediction. In: : IEEE, 2015 — ISBN 978-1-4799-8937-9, pp. 417–422

[60] Palem, Gopalakrishna: The Practice of Predictive Analytics in Healthcare. In: *ResearchGate* (2013)

[61] Burger, Nicholas ; Ghosh-Dastidar, Bonnie ; Grant, Audra ;

Joseph, George ; Ruder, Teague ; Tchakeva, Olesya ; Wodon, Quentin: Data Collection for the Study on Climate Change and Migration in the MENA Region (2014)

[62] Gaitho, Maryanne: *Applications of big data in 10 industry verticals*. URL https://cfs22.simplicdn.net/ice9/free_resources_article_thumb/Applications_of_big_data_infographic.png. - retrieved 2016-11-19

[63] USA, Federal Trade Commission: *Personal data ecosystem*. URL https://www.ftc.gov/sites/default/files/documents/public_events/exploring-privacy-roundtable-series/personaldataecosystem.pdf. - retrieved 2016-11-17. — Protecting Consumer Privacy in an Era of Rapid Change - Recommendations for Business and Policymakers - FTC Report

[64] Christl, Wolfie: *Corporate surveillance, digital tracking, big data & privacy*, 2016

[65] Moore, Gordon E.: Cramming more components onto integrated circuits. In: *Electronics* vol. 38 (1965), p. 4

[66] Pritlove, Tim ; Schöneberg, Ulf: *Neuronale netze*, 2015

[67] Columbus, Louis: *51% of enterprises intend to invest more in big data*. URL http://www.forbes.com/sites/louiscolumbus/2016/05/22/51-of-enterprises-intend-to-invest-more-in-big-data/. - retrieved 2016-12-07

[68] *ProjectVRM - cDevelopment work. ProjectVRM*. URL https://cyber.harvard.edu/projectvrm/VRM_Development_Work. - retrieved 2016-12-09

[69] *ProjectVRM - principles. ProjectVRM*. URL https://cyber.harvard.edu/projectvrm/Main_Page#VRM_Principles. - retrieved 2016-12-09

[70] *TAS3 - project overview*. URL http://vds1628.sivit.org/tas3/. - retrieved 2017-02-10

[71] The TAS3 Consortium: *TAS3 architecture - figure 2.2: Major components of organization domain.*, 2011. — v 2.24

[72] *Kantara initiative – join. innovate. trust.* URL https://

kantarainitiative.org/. - retrieved 2016-12-14

[73] Kirkham, Tom ; Winfield, Sandra ; Ravet, Serge ; Kellomaki, Sampo: The personal data store approach to personal data security. In: *IEEE Security & Privacy* vol. 11 (2013), Nr. 5, pp. 12–19

[74] Montjoye, Yves-Alexandre de ; Wang, Samuel S. ; Pentland, Alex ; Anh, Dinh Tien Tuan ; Datta, Anwitaman ; others: On the trusted use of large-scale personal data. In: *IEEE Data Eng. Bull.* vol. 35 (2012), Nr. 4, pp. 5–8

[75] *openPDS/SafeAnswers - the privacy-preserving personal data store.* URL http://openpds.media.mit.edu/. - retrieved 2016-12-14

[76] Montjoye, Yves-Alexandre de ; Shmueli, Erez ; Wang, Samuel S. ; Pentland, Alex Sandy: openPDS: Protecting the privacy of metadata through SafeAnswers. In: Preis, T. (ed.) *PLoS ONE* vol. 9 (2014), Nr. 7, p. e98790

[77] *Microsoft HealthVault. Overview.* URL https://www.healthvault.com/de/en/overview. - retrieved 2016-12-14

[78] *How it works meeco.* URL https://meeco.me/how-it-works.html. - retrieved 2016-12-14

[79] Page, Mike: Online adver>sing – booming or broken?, 2015

[80] *The principles. Industrial data space e.V.* URL http://www.industrialdataspace.org/en/the-principles/. - retrieved 2016-12-14

[81] Prof. Dr.-Ing. Otto, Boris ; Prof. Dr. Auer, Sören ; Cirullies, Jan ; Prof. Dr. Jürjens, Jan ; Menz, Nadja ; Schon, Jochen ; Dr. Wenzel, Sven: Industrial data space - digital sovereignity over data.

[82] Diffie, Whitfield ; Hellman, Martin: New directions in cryptography. In: *IEEE transactions on Information Theory* vol. 22 (1976), Nr. 6, pp. 644–654

[83] Stallings, William: 9.1 principles of public-key cryptosystems. In: *Cryptography and network security: Principles and practice.* Seventh edition.

ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, pp. 256–264

[84] K. Moriarty, Ed. ; Kaliski, B. ; Jonsson, J. ; Rusch, A.: *PKCS #1: RSA cryptography specifications version 2.2*. URL https://tools.ietf.org/html/rfc8017. - retrieved 2017-03-04

[85] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *Hypertext transfer protocol – HTTP/1.1*. URL https://tools.ietf.org/html/rfc2616. - retrieved 2016-12-17

[86] OSI model. *Wikipedia*.

[87] Belshe, Mike ; Thomson, Martin ; Peon, Roberto: *Hypertext transfer protocol version 2 (HTTP/2)*. URL https://tools.ietf.org/html/rfc7540. - retrieved 2016-12-17

[88] Dierks, Tim ; Rescorla, E.: *The transport layer security (TLS) protocol version 1.2*. URL https://tools.ietf.org/html/rfc5246. - retrieved 2016-12-17

[89] Stallings, William: 10.5 pseudorandom number generation based on an asymmetric cipher. In: *Cryptography and network security: Principles and practice*. Seventh edition. ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, pp. 443–445

[90] Cooper, Dave ; Santesson, S. ; Farrell, S. ; Boeyen, S. ; Housley, W., R. andPolk: *Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile*. URL https://tools.ietf.org/html/rfc5280. - retrieved 2017-01-11

[91] Fette, Ian ; Melnikov, A.: *The WebSocket protocol*. URL https://tools.ietf.org/html/rfc6455. - retrieved 2016-12-17

[92] Kleinhans, Jan-Peter: *Basisleser weiterhin kritische schwachstelle des elektronischen / neuen personalausweises. Netzpolitik.org*. URL https://netzpolitik.org/2013/basisleser-weiterhin-kritische-schwachstelle-des-elektronischen-neuen-personalausweises/. - retrieved 2017-01-05

[93] Stiemerling, Oliver: *Qualifizierte elektronische signatur mit dem neuen personalausweis – oder: QES mit nPA, ein selbstversuch.*

CR-online.de blog. URL http://www.cr-online.de/blog/2014/08/26/qualifizierte-elektronische-signatur-mit-dem-neuen-personalausweis-oder-qes-mit-npa-ein-selbstversuch/. - retrieved 2017-01-05

[94] Der Bundesbeauftragte der Bundesregierung für Informationstechnik: *De-mail – einfach verschlüsselt und nachweisbar*. URL http://www.cio.bund.de/Web/DE/Innovative-Vorhaben/De-Mail/de_mail_node.html. - retrieved 2017-01-06

[95] Qualified electronic signature. *Wikipedia*.

[96] Crockford, Douglas: The JSON data interchange format.

[97] Bray, T.: *The JavaScript object notation (JSON) data interchange format*. URL https://tools.ietf.org/html/rfc7159. - retrieved 2016-12-17

[98] Bradley, John ; Sakimura, Nat ; Jones, Michael: *JSON web token (JWT)*. URL https://tools.ietf.org/html/rfc7519. - retrieved 2016-12-17

[99] Joseffsson, J.: *The base16, base32, and base64 data encodings*. URL https://tools.ietf.org/html/rfc4648#section-5. - retrieved 2017-02-22

[100] Krawczyk, H. ; Bellare, M. ; Canetti, R.: *HMAC: Keyed-hashing for message authentication*. URL https://tools.ietf.org/html/rfc2104. - retrieved 2017-03-01

[101] Hildebrand, Joe ; Jones, Michael: *JSON web encryption (JWE)*. URL https://tools.ietf.org/html/rfc7516. - retrieved 2016-12-17

[102] Bradley, John ; Sakimura, Nat ; Jones, Michael: *JSON web signature (JWS)*. URL https://tools.ietf.org/html/rfc7515. - retrieved 2016-12-17

[103] Bradley, John: *The problem with OAuth for authentication*. URL http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html. - retrieved 2016-12-17

[104] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/. - retrieved 2016-12-18

[105] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https:

//tools.ietf.org/html/rfc6749. - retrieved 2016-12-18

[106] WG, IETF OAuth: *OAuth 2.0*. URL https://oauth.net/2/. - retrieved 2016-12-16

[107] *Specifications & developer information OpenID*. URL https://openid.net/developers/specs/. - retrieved 2017-02-10

[108] *OpenID connect core 1.0 incorporating errata set 1*. URL https://openid.net/specs/openid-connect-core-1_0.html. - retrieved 2016-12-17

[109] *Facebook login - documentation. Facebook for developers*. URL https://developers.facebook.com/docs/facebook-login. - retrieved 2017-03-05

[110] Facebook connect. *Wikipedia*.

[111] *OpenID - openid vs. pseudo-authentication using oauth*. URL https://en.wikipedia.org/w/index.php?title=OpenID&oldid=763047614#OpenID_vs._pseudo-authentication_using_OAuth. - retrieved 2017-02-10. — Page Version ID: 763047614

[112] Fielding, Thomas: Representational state transfer (REST). In: *Architectural styles and the design of network-based software architectures* : University of California, Irvine, 2000, pp. 76–106

[113] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *HTTP methods*. URL https://tools.ietf.org/html/rfc2616#section-9. - retrieved 2016-12-18

[114] Heo, Tejun: *Control group (v2) documentation*. URL https://www.kernel.org/doc/Documentation/cgroup-v2.txt. - retrieved 2016-12-20

[115] *Overview of linux namespaces*. URL http://man7.org/linux/man-pages/man7/namespaces.7.html. - retrieved 2016-12-20

[116] *Open container initiative*. URL https://www.opencontainers.org/. - retrieved 2016-12-20

[117] *Container runtime specification (v1.0.0-rc3)*. URL https://github.com/opencontainers/runtime-spec/tree/v1.0.0-rc3. - retrieved 2016-12-20

[118] *Container image specification (v1.0.0-rc3)*. URL https://github.com/

opencontainers/image-spec/tree/v1.0.0-rc3. - retrieved 2016-12-20

[119] *GraphQL*. URL https://facebook.github.io/graphql/. - retrieved 2016-12-17

[120] *Semantic web - w3c*. URL https://www.w3.org/standards/semanticweb/. - retrieved 2017-02-10

[121] Beckett, Dave ; McBride, Brian: *RDF/XML syntax specification (revised)*. URL https://www.w3.org/TR/REC-rdf-syntax/. - retrieved 2016-12-19

[122] W3C OWL Working Group: *OWL 2 web ontology language document overview (second edition)*. URL https://www.w3.org/TR/owl2-overview/. - retrieved 2016-12-19

[123] Harris, Steve ; Seaborne, Andy ; Prud'hommeaux, Eric: *SPARQL 1.1 query language*. URL https://www.w3.org/TR/sparql11-query/. - retrieved 2016-12-19

[124] *WebID specifications*. URL https://www.w3.org/2005/Incubator/webid/spec/. - retrieved 2016-12-19

[125] *Solid specification*. URL https://github.com/solid/solid-spec. - retrieved 2016-12-17

[126] *WebAccessControl - w3c wiki*. URL https://www.w3.org/wiki/WebAccessControl. - retrieved 2016-12-19

[127] *Databox.me*. URL https://databox.me/. - retrieved 2016-12-19

[128] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified data processing on large clusters. In: *Proceedings of the 6th conference on symposium on opearting systems design & implementation - volume 6*, *OSDI'04*. Berkeley, CA, USA : USENIX Association, 2004, pp. 10–10

[129] Dierks, Tim: *The transport layer security (TLS) protocol version 1.2*. URL https://tools.ietf.org/html/rfc5246#section-7.4.6. - retrieved 2017-01-09

[130] Reschke, J.: *The 'basic' http authentication scheme*. URL https:

//tools.ietf.org/html/rfc7617. - retrieved 2017-03-05

[131] Barth, A.: *HTTP state management mechanism*. URL https://tools.ietf.org/html/rfc6265. - retrieved 2017-03-05

[132] *Mutual authentication*. URL https://en.wikipedia.org/w/index.php?title=Mutual_authentication&oldid=737409981. - retrieved 2017-01-10. — Page Version ID: 737409981

[133] *Networking 101: Transport layer security (TLS) - high performance browser networking (o'Reilly). High performance browser networking*. URL https://hpbn.co/transport-layer-security-tls/#tls-session-resumption. - retrieved 2017-01-12

[134] Joseph Salowey, P. Eronen, H. Zhou: *Transport layer security (TLS) session resumption without server-side state*. URL https://tools.ietf.org/html/rfc5077. - retrieved 2017-01-12

[135] Stallings, William: 9.1 public-key infrastructure. In: *Cryptography and network security: Principles and practice*. Seventh edition. ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, p. 307

[136] *BSI - technische richtlinien des BSI - BSI TR-03130 eID-server*. URL https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03130/tr-03130.html. - retrieved 2017-01-06

[137] *Personalausweisportal - eID-server*. URL https://personalausweisportal.de/DE/Wirtschaft/Technik/eID-Server/eID-Server_node.html. - retrieved 2017-01-06

[138] Linus Neumann (Press Spokespersion, Chaos Computer Club): *Stellungnahme zum elektronischen rechtsverkehr*. URL https://ccc.de/system/uploads/128/original/demail_april2013.pdf

[139] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *Hypertext transfer protocol – HTTP/1.1*. URL https://tools.ietf.org/html/rfc2616#section-10. - retrieved 2017-01-20

[140] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/#rfc.section.4.2. -

retrieved 2016-11-01

[141] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https://tools.ietf.org/html/rfc6749#section-2. - retrieved 2016-11-01

[142] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/#rfc.section.7. - retrieved 2016-11-01

[143] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https://tools.ietf.org/html/rfc6749#section-7. - retrieved 2016-11-01

[144] Brickley, Dan ; Guha, R.V.: *RDF schema 1.1*. URL https://www.w3.org/TR/rdf-schema/. - retrieved 2017-02-12

[145] Foundation: *OpenEHR - EHR information model*. URL http://www.openehr.org/releases/RM/latest/docs/ehr/ehr.html. - retrieved 2017-01-28

[146] W3C: *Points of interest core*. URL https://www.w3.org/TR/poi-core/. - retrieved 2017-01-28

[147] Authority, ISO 20022 Registration: *ISO 20022 - universal financial industry message scheme*. URL https://www.iso20022.org/. - retrieved 2017-01-28

[148] W3C: *XML schema part 2: Datatypes second edition*. URL https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes. - retrieved 2017-01-29

[149] Facebook, Inc.: *GraphQL Specification*. URL https://facebook.github.io/graphql/#sec-Input-Values. - retrieved 2017-01-29

[150] *Separation of concerns*. URL https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=747272729. - retrieved 2017-01-24. — Page Version ID: 747272729

[151] Kasten, James ; Barnes, Richard ; Hoffman-Andrews, Jacob: *Automatic certificate management environment (ACME)*. URL https://tools.ietf.org/html/draft-ietf-acme-acme-04. - retrieved 2017-01-11

[152] Carlson, Nicholas: *Facebook connect is a huge success – by the numbers*. URL http://www.businessinsider.com/six-months-in-facebook-

connect-is-a-huge-success-2009-7. - retrieved 2016-12-16

[153] *Usage share of operating systems*. URL https://en.wikipedia.org/w/index.php?title=Usage_share_of_operating_systems&oldid=764383522#Public_servers_on_the_Internet. - retrieved 2017-02-12. — Page Version ID: 764383522

[154] *Accordion (GUI)*. URL https://en.wikipedia.org/w/index.php?title=Accordion_(GUI)&oldid=758084292. - retrieved 2017-02-01. — Page Version ID: 758084292

[155] SELF-ORGANIZING ; GROUP): Census act, BVerfGE 65, 1 — English translation of essential parts of the german "volkszählungsurteil" from 15 december 1983, which established in germany the basic right on informational self-determination : German Konrad-Adenauer-Stiftung, 2013. — 1. Headnote

[156] BISELLI, ANNA: *Interne dokumente zur weltraumtheorie: Wie sich BND und kanzleramt vor der öffentlichkeit fürchteten. Netzpolitik.org*. URL https://netzpolitik.org/2016/interne-dokumente-zur-weltraumtheorie-wie-sich-bnd-und-kanzleramt-vor-der-oeffentlichkeit-fuerchteten/. - retrieved 2017-03-03

[157] ZUCKERBERG, MARK: *Building global community*. URL https://www.facebook.com/notes/mark-zuckerberg/building-global-community/10154544292806634. - retrieved 2017-02-27

[158] BALKAN, ARAL: *Encouraging individual sovereignty and a healthy commons*. URL https://ar.al/notes/encouraging-individual-sovereignty-and-a-healthy-commons/. - retrieved 2017-02-27

[159] WILKENS, ANDREAS: *Allianz will autoversicherung mit überwachtem fahrverhalten anbieten. Heise online*. URL http://www.heise.de/newsticker/meldung/Allianz-will-Autoversicherung-mit-ueberwachtem-Fahrverhalten-anbieten-3164069.html. - retrieved 2017-03-02

[160] SEARLS, DOC: *The distributed future is personal. ProjectVRM - blog*. URL https://blogs.harvard.edu/vrm/2017/01/18/the-distributed-future-is-

personal/. - retrieved 2017-03-03

[161] Zyskind, Guy ; Nathan, Oz ; others: Decentralizing privacy: Using blockchain to protect personal data. In: *Security and privacy workshops (SPW), 2015 IEEE* : IEEE, 2015, pp. 180–184

[162] *Aldeed/node-simple-schema. GitHub.* URL https://github.com/aldeed/node-simple-schema. - retrieved 2017-01-29