INSTITUTE FOR COMPUTER SCIENCE
OF THE LUDWIG–MAXIMILIANS–UNIVERSITY MUNICH

Master's Thesis

Conception and implementation of a storage solution for handling documents with legal admissibility

Daniel Steiert

INSTITUTE FOR COMPUTER SCIENCE
OF THE LUDWIG–MAXIMILIANS–UNIVERSITY MUNICH

Master's Thesis

Conception and implementation of a storage solution for handling documents with legal admissibility

Daniel Steiert

Supervisor: Prof. Dr. W. Hommel
Advisors: Felix von Eye
Michael Grabatin

Submission Date: May 10th, 2016

I hereby declare that I have developed and written the enclosed Master's Thesis entirely by myself, and have not used sources or means other than declared in the document. Intellectual property of others and literal quotations are clearly marked. The Master's Thesis has neither been used in this or in another version to achieve an academic grading or to be published elsewhere.

Munich, May 8, 2016

. . . . . .

(Candidate's Signature)

Abstract

People have long dreamt of the paperless office. Saving costs for the storage of many different document versions, finding and managing documents faster and exchanging everything digitally is the wish of every company. Yet, especially when it comes to storing documents that require the written form and whose evidential value must be preserved over long periods of time, little to no solutions can be found, aside from very broad and unpractical ones. In addition to this, there currently is no product on the market to create, edit, approve and manage procedure descriptions on a centrally accessible system, which does not require paper-copies to be stored.

With this thesis an attempt to fill the previously described void is made. The problem is described in detail and a possible solution is proposed for which basic knowledge and a legal base are researched. From this groundwork and scenarios encountered during daily work, requirements are derived that are later used to evaluate current solutions and possible components for the future solution. Considering the requirements, an extensive and secure concept was developed which has also been implemented on a prototype basis. After finishing the implementation, the system was evaluated regarding security aspects and the earlier defined requirements. All mandatory requirements were fulfilled, allowing for the data protection commissioner and the managers of services to create, edit, search, sort and manage procedure descriptions while ensuring the authenticity, confidentiality and integrity of documents. Finally, possible future paths are described which could be taken to further continue with and improve the developed system.

Contents

1 Introduction

Every day we disclose our personal data when we register for an online service, login, use provided support or order from online-stores, allowing others to work with and benefit from what we consider our privacy. It has become second nature we usually don't spend too much time thinking about. This data needs to be kept safe and needs to be handled with care.

Just like with personal data, there is also sensitive data, which is handled for example in finance, accounting and book keeping. This stored information should not to be changeable or modifiable without appropriate permissions or change-history. It needs to be reliably stored in a versioned manner for the people involved, to be able to be accountable by legal standards. This kind of data can be found in all kinds of different organizations including banks, the stock market, insurances, public institutions, companies and more. In almost all cases it is important to keep records; to be able to look up who had access to the information, when it was changed, what was changed or how the data developed over time.

A lot of this has been done in an analog fashion for many years, which involved working with contracts, filling out forms, signing papers, printing out documents and storing them away in filing cabinets. Endless rows of storage shelves needed to be maintained, protected and managed, which made searches and comparisons very time- and cost-consuming.

Then came the digital age. People dreamed of the paperless office, only working with computers, spreadsheets and digital versions of documents. Yet, printers remained, the amount of paper and data created increased. Files were still being printed out and stored in folders and filing cabinets. Especially for processes involving money, people did not fully trust computers and digital storage solutions to securely protect their data from being lost or modified without their knowledge. It's not like there weren't possibilities to encrypt data and archive it on multiple mediums like tape-drives for long-term storage. But there was always a fear of losing or unknowingly changing data in the process somewhere along the way. It also didn't help, that secure long-term storage solutions were very costly. Due to this, companies to this day often keep legally binding documents in printed form in addition to the digital versions. Remains of this fixed mindset can be found even in currently existing laws, requiring documents to be signed by hand or with witnesses around [bgb]. Only in 2001, paragraphs on electronic form [Gar] in Germany have been modified and added to also account for legally binding, digitally signed virtual documents.

In addition to this, data privacy and especially personal data privacy came into special focus after Mr. Snowden's revelations of some of the work of secret services [ER13]. People are becoming more and more aware of how and where they share their personal information and who has access to it. Germany has been painfully aware of the need for data protection guidelines, following World War II's DDR and its massive spying efforts into people's private lives [Schb]. As a result of this awareness, the Bundesdatenschutzgesetz [bds] emerged, which basically defines how data is to be treated, who is allowed to access it and who is responsible

and liable.

Following in its footsteps came the Bavarian State Law for Data Protection and Privacy[1] [bay], which among other things, holds the regulations on how to document and deal with automatic procedures involving personal data in public institutions. These procedure descriptions are part of a so-called public procedure register, which needs to be accessible by anybody interested in the handling of personal data in these institutions. This regulation is there to protect the people disclosing personal data and the company from mistakenly treating this data inappropriately. It also helps to make it possible to examine compliance of automatic data processing with prevailing laws. Since these public institutions are required by law to keep the descriptions, register and its changes up-to-date and sign them properly, it is currently necessary to do most of the management and approval by hand on actual paper.

Until recently there was no reliable and legally accountable way of maintaining this data. Just as with book keeping, it needs to be guaranteed that the data hasn't been tampered with. This is where the storage solution for handling versioned documents with legal certainty of this thesis comes in.

1.1 Problem Statement

It should be straight-forward and unproblematic to create and manage legally relevant documents. People involved should not have to worry about where to store them or how to preserve their evidential value. Computer systems have advanced far enough and are connected in a way that should allow for seamless transmission and handling of these documents without the need to worry about data getting lost or being changed in an unauthorized or undocumented way. The ideal situation would be that when new information becomes available, people enter it or make changes to existing documents, approve proposals by signature, store documents, transfer them to their next step in a workflow or archive them, so that the person that handled the document does not have to give any more thought to this.

However, even with all this technology and the advances that have happened within the computer industry, there is often still a need for real, handwritten signatures or seals and stamps on paper and versioned documents, which in turn are stored in filing cabinets and physical shelves. It has to do with the fact that creating and maintaining legally admissible document management solutions, can be very expensive and complicated. Often one of the easiest ways is to restrict access to the system and use a write once read many (WORM) storage solution. This would ensure that data cannot be overwritten, but must be re-written at a different location. With this type of storage, the amount of data stored can increase dramatically within a short amount of time, since any change means writing the data again, duplicating the data in the procedure.

A specific use-case would be the management of procedure descriptions in a Bavarian public institution like the Leibniz Supercomputing Centre (LRZ). These procedure description documents define how autmomatic handling of personal data takes place within the institution. Figure 1.1 shows a workflow in which such a document is created by a manager before handing it over to the data protection commissioner in step 1, who approves it in step 2.

[1] Bayerische Datenschutzgesetz, translated by Brunner, O. and Ermer, D.[Vet]

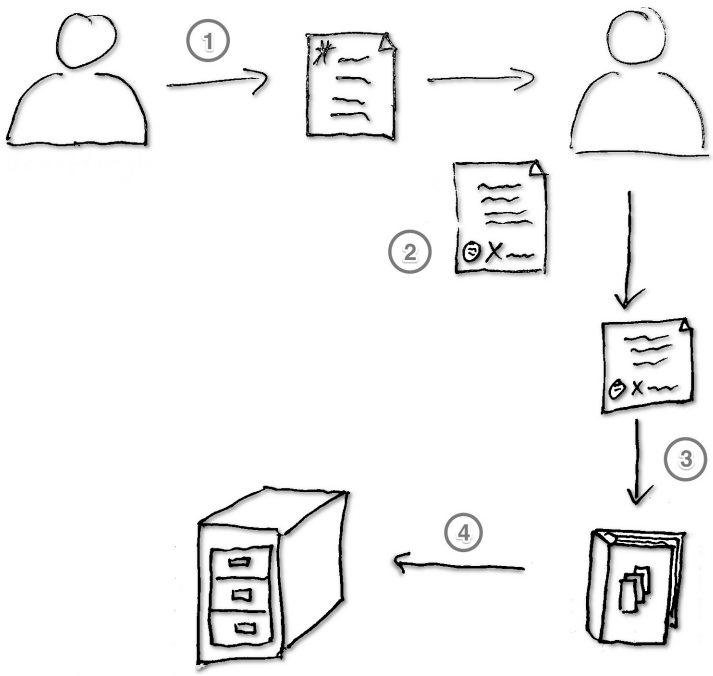

Figure 1.1: Workflow for a new Procedure Description

This approved document then is filed in a folder as depicted with step 3 and stored in a filing cabinet in step 4. Applying changes to existing documents or creating statistics of the last few years for these documents can contain even more steps. These documents need to be stored and managed in a legally admissible way and any change needs to be reconstructable with timestamps and other meta-data. They also need to be signed according to the standards defined in article 26 of the Bavarian State Law for Data Protection and Privacy[bay] and § 126a of the German Civil Code [Gar]. To ensure the integrity of these documents and their authenticity, it would be possible to use in themselves dependent signatures within the system. WORMs alone would only ensure that these documents haven't been modified after they have been written, and don't show if the document was properly authenticated or signed as shown in the workflow depicted in Figure 1.1 or that the approved document hasn't been modified. There is a need for a more complex management solution, that can manage changes and guarantee authenticity and quality of these documents on a legal level, while allowing for them to be compared and revised. Creating this kind of solution in a digital system on software basis could greatly increase productivity and lower costs of management.

Current solutions for this kind of document management either don't fulfil the requirements, are too expensive, or too complicated to manage and set up, which is why there is a need for a software solution to manage and maintain these documents in a cost-effective and effortless manner.

1.2 Task Description

Since current solutions for legally admissible document management with proper signatures are either too expensive or too complicated, while also not fulfilling the requirements for working with electronically signed documents, the task of this Master's Thesis is to conceptualise and implement such a storage and management solution. The concept should account for all the basic data security goals like confidentiality, integrity and availability. This ensures that the documents stored are legally admissible in a court case.

Although the thesis looks at current solutions and develops an internationally acceptable path, it uses the specific needs of the Leibniz Supercomputing Centre (LRZ) for management of procedure descriptions, as described by the Bayerische Datenschutzgesetz [bay] as an example use-case. In addition, this solution should allow for versioned storage of documents with qualified electronic signatures, while protecting data from unauthorized or accidental modifications. It should be possible to use basic workflows in a user permission based authorization schema to create, look up, change, review and approve documents. When approving documents, it is necessary to follow the guidelines stated in article 26 of the Bavarian State Law for Data Protection and Privacy. After a certain amount of time, approved documents should automatically be submitted for re-evaluation. Any concepts used or developed should be platform- and product-neutral.

The implementation, although tailored to the specific needs of the LRZ and its management of procedure descriptions, should be generic and applicable even to other use-cases and be able to run on a Linux web- or application-server. Existing libraries and frameworks can be used to increase performance of the development process. An acceptable prototype solution should at least allow for creation, modification and approval of documents with basic user permission management. It needs to fulfil the defined data security requirements to be compliant with legal admissibility laws and hold up in court when submitted.

1.3 Approach

Chapter 2 deals with the technical background and legal essentials necessary for the understanding and reading of this thesis. It defines how to understand signatures and encryption, how version control works and what authorization and authentication is. Next it also mentions and explains the involved laws, their jurisdiction and what procedure descriptions and the procedure register is.

In Chapter 3 the requirements towards the concept and implementation are defined, while separating between functional requirements and non-functional requirements. They are necessary to express what we are looking for in related work and what needs to be thought of for the concept at a later time.

The Chapter 4 evaluates existing solutions and components related to the topic of this thesis. While doing so it considers the requirements and task description to find possibly usable elements for a proposed solution.

Next, the concept for a solution is developed with a specification of the components and definition of the used architecture in Chapter 5. This concept is on the one hand generic and

should be able to be applied on a wide range of areas, while on the other, be specific in the way that it takes the LRZ's requirements into consideration. Part of those requirements are a qualified electronic signature.

Using the concept of Chapter 5, Chapter 6 describes the prototype that is created from it. It needs to fulfil the basic requirements mentioned in Chapter 1.2.

After the prototype is finished, it is evaluated in Chapter 7 to make sure it complies with the mentioned guidelines and laws by assessing the fulfilment of functional and non-functional requirements and if the application is secure.

Finally, Chapter 8 summarizes the findings and portrays the outlook and further research possibilities in this area.

2 The Basics

To be able to create a secure, reliable and legally sound digital storage solution for documents, we need to first examine the technical background it will be based on and the legal obligations which result from the laws involved. To ensure compliance, not just local, but also international laws and regulations must be considered. This chapter gives a broad overview of these technologies and laws.

2.1 Technical Essentials

This chapter gives a short introduction to the technological basics needed to work with the laws and to understand the inner workings of the later resulting system. It takes a look at how these technologies can be used and how they are to be understood within the context of this thesis.

2.1.1 Version Control

Version control is necessary to ensure all changes made to documents can be securely stored, reviewed and later perhaps reverted to an earlier version. It is also useful to log who made changes and when these changes took place. To achieve this, there are different ways of maintaining changes. One is to create a full copy of each version of the document. Another would be to save the base-version of a document and only save iterative changes between versions. Storage can be saved using the second approach because data is not stored redundantly. Version control can also be achieved within databases by watching changes made to datasets and adding hooks to write out any changes which have been made in form of logging. Meta-data of changes can be stored in its own database or tables by noting down the version number, editor, changes themselves, time of change and the hash of the revision.

2.1.2 Authentication and Authorization

Users of a system often have different responsibilities. Not everyone is allowed to see everything or to perform just any task. Due to this, there needs to be a way to safely identify users and grant them certain rights when using software. Authentication allows for a user's identity to be verified according to the credentials provided. The system can check these credentials against stored values and so confirm the user's identity. This is possible for example through username and password combinations, certificates or digital tokens.

Once authenticated, users can have different roles or obligations, which creates a need to manage the user's or the group's rights. The process of verifying that a user can perform a certain task is called authorization.

2.1.3 Cryptographic Hashing

Hashing algorithms are algorithms which generate a specific string of characters from an original input through a one way function. There is no way to reverse the generation of the hash and with modern and certified hashing algorithms it is highly unlikely that two original strings of characters will have the same, colliding hash-value.

When dealing with digital documents, a hash is generated from the document with a specific, documented procedure. This hash represents the contents of the document and is uniquely related to it. This hash can then be signed with a provided private key. Using the corresponding public key, the validity of the hash value and thus the validity of the document itself can be verified.

Examples of cryptographic hashing algorithms are Secure Hashing Algorithm (SHA) [sha], RACE Integrity Primitives Evaluation Message Digest (RIPEMD-160) [DBP96] or bcrypt [bcr]. Not all algorithms are equally secure and a system containing cryptographic hashes can be attacked for example, by using dictionary attacks, brute-force methods or rainbow tables. The SHA-512 algorithm for example can be used until the end of 2021 in signatures, according to the "Publication to electronic signatures according to the Signature Law and the Signature Regulation"[1] [Reg] of the Federal Network Agency for Electricity, Gas, Telecommunications, Mail and Trains[2]. RIPEMD-160 on the other hand is only certified for use until the end of 2015 [Reg], which makes it unsuitable for further use even during the writing of this thesis.

2.1.4 Signatures and Encryption

When dealing with data in public places such as the internet, it quickly becomes relevant to protect transferred information from unauthorized viewing or modification. To secure against the viewing through third parties, it is possible, for example, to use a public key and specific algorithms to transform readable data into secure and encrypted data which can be transferred to the receiver. The receiver can decrypt the data with his own matching private key and in doing so make the data readable again. This form of encryption is called public-key encryption and is based on asymmetric key algorithms.

Signatures and encryption can also be used to ensure the authenticity of a sent message by the sender. To do this, the sender creates a hash-value of the original data and encrypts it with his private key. Both the encrypted hash-value and the message can then be sent to the receiver. The receiver can now decrypt the hash-value with the sender's public key and generate the hash-value from the message. If the decrypted hash-value and the generated hash-value match, the authenticity of the message is proven and the receiver can safely assume the message was sent by the sender.

In modern computing, encryption and signing are often combined to ensure privacy and integrity of the messages sent at the same time. To do this, the sender would first encrypt the message with the receiver's public key, generate a hash-value of the encrypted message and sign that hash-value with his own private key. Both the signed hash-value of the encrypted

[1] Author's translation of "Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung"

[2] Author's translation of "Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen"

message and the encrypted message itself are sent to the receiver, who can then first validate the authenticity of the message with the sender's public key and then decrypt it with the receiver's private key.

To store keys and assign them to users or objects, certificates can be used which hold information about that person or object and can be verified for authenticity and integrity. There are also attribute-certificates, which describe further detailed information about the entity and refer to the main certificate. A well-known and often used standard for such public key certificates is the X.509 standard [x50]. Generally, these certificates hold information about the owning entity and it's public key, which can be used in a public key infrastructure (PKI).

For some purposes it might be necessary to use qualified electronic certificates, which have been issued by an organization that is supervised by the responsible authority of a country and fulfils the requirements for qualified electronic signatures as stated in the signature guideline 1999/93/EC [199]. It is uniquely attributed to one entity and can be used to sign documents electronically so that they have the same evidential value as handwritten and signed documents. Qualified electronic signatures and their use are explained in more detail in Section 2.2.

2.2 Guidelines and Laws

When creating a legally compliant software application to store documents and preserving their evidential value, it is important to be familiar with the laws and regulations involved. In this chapter the most important guidelines, laws and regulations are listed and explained to give a broad overview of their proposed requirements.

2.2.1 Data Protection Guidelines

To allow for international collaboration on a digital level and fast processing across borders, it is necessary to have data protection and data privacy laws in place that protect basic human rights of citizens when it comes to exchange, storage and abuse of their personal data. Therefore, the Organization for Economic Co-operation and Development (OECD) [oecb] has created guidelines called the OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data [oeca], which are implemented by privacy protection laws in many of the involved OECD Member countries, as stated in the preface of the guidelines themselves [oeca].

The International Standard Organization (ISO) [isoa] defined the standard ISO 15489-1:2001 [Intb] and ISO 15489-2:2001 [Intc] which hold general and broad guidelines on how to handle and manage records, how to conceptualize and create record management systems, and how to treat the data within. It is meant for public and private organizations with internal or external clients and their records, complying with ISO 9001 and ISO 14001, while not giving guidance on how to properly archive records. In it, all needed terms are named and defined for better international understanding of the processes and guidelines involved with document and record management. This international norm has been adopted seamlessly in Germany as a DIN-norm and in doing so also, lays out the groundwork for laws building

on top of it. Some of the things mentioned in "Part 1: General" of the ISO 15489 standard [Intb] are characteristics of records, such as authenticity, reliability, integrity and usability, as well as how to design and implement records management systems, which design and implementation methodology to use and how to ensure proper capturing and management of records.

In particular, it gives specific definitions of authenticity, reliability and integrity. By this definition, a document is authentic if it is possible to verify "that it is what it purports to be", was "created by the person that sent it" and that "it was created or sent at the time which is purported" [Intb]. Reliability means it can be "trusted as a full and accurate representation of transactions, activities of facts" [Intb]. Integrity on the other hand guarantees completeness, that it hasn't been altered and that everything is traceable [Intb]. Along with these definitions there are suggestions on designing and implementing records systems to ensure legal admissibility of records. The following are summaries taken from the ISO 15489-1 standard [Intb]:

According to the suggestions, all records within the scope must be captured, organized in ways of business processes and protected from unauthorized alteration or disposition. All access should be monitored, users verified and destruction authorized, while dividing access into internal and external. It is necessary to be able to manage records through alternative methods, without influencing the authenticity of the documents through viewings, modifications or conversions. The system must allow for efficient and timely access to and retrieval of documents with proper controls for access and audit trails to track any viewings and changes. It must also allow for automatic retention and disposition, which can be activated at any time.

The accompanying technical report, the so called "Part 2: Guidelines" of the ISO 15489 standard [Intc], gives specific expanded instructions on how to understand and make use of the guidelines given in Part 1 [Intc] and which processes are involved.

The European Union in October of 1995 created the directive 95/46/EC of the European Parliament And Of The Council, which holds guidelines for "the protection of individuals with regard to the processing of personal data and on the free movement of such data" [eu9]. In it lie the most basic requirements to ensure proper data protection within the participating states of the European Union, which are to be transferred into state-laws. These requirements lay the groundwork for laws like the German Bundesdatenschutzgesetz, which to this day isn't fully compliant with the directive [Pat].

Moving close to German regulations, the federation Voice Of Information [Scha] has given out rules of thumb on how to implement auditable and evidential value preserving electronic archiving in compliance with the law. At the writing of this thesis it has a list of ten points, ranging from storing documents according to all legal and organisational regulations, when and how to archive, how fast to access them, when to delete, how and what to log, how to ensure compliance through audits and finally what to watch out for when making changes to or migrating the system [VOI].

Next there are regulations to ensure legal admissibility of cryptographically signed documents. The Federal Office for Security in Informationtechnologies[3] (BSI) has recognized a need for standards to ensure evidential value of documents in an increasingly digitalized

[3]Author's translation of Bundesamt für Sicherheit in Informationstechnik (BSI)

world. Therefore the BSI created the technical guideline BSI 03125 TR-ESOR [Feda].

The guideline serves as a means for preservation of evidential value of signed documents. It is to be noted, that the guideline clearly states in the beginning that it is not mandatory to ensure preservation of evidential value by law. It is merely a collection of recommendations to ensure compliance and maximize acceptance of documents as evidence in court or by law. It consists of the main document and various attachments, describing topics more clearly. The main document describes the legal framework, standards, goals, requirements, processes and functions to achieve this preservation in system architectures, which are also described as a reference-implementation. The attached documents illustrate how to specify and define the previously described requirements according to a platform- and product-neutral, functional reference-architecture. It strictly follows the legal requirements for cryptographic preservation of evidential value of signed documents and verifies its conformity at the end of the main document [Feda].

Chapter 6 of these BSI guidelines talks about the resulting technical requirements for a middleware designed in accordance to the standard. The following paragraphs are summaries of the BSI 03125 TR-ESOR guidelines 6.1 [Feda]:

The system must allow for integration with existing and newly developed information systems, for interoperability, availability and marketability of the used exchange-formats of payloads, meta-data, signatures, timestamps and signature-check information for the entire time required by law for archiving documents. [Feda]

The solutions for preservation of evidential value of signed electronic documents must not influence the further usability of electronic documents for different application purposes and in different application systems. Specifically for

- the exchange of documents between application-systems
- the transition of data formats in applications-systems
- the exchange of application-systems or -components

there must not be any obstruction through the method and technical solutions to renew signatures. [Feda] "The TR-ESOR-middleware should be able to manage separated clients. This specifically means there needs to be a strict (logical) separation of the objects to be archived in the Enterprise Content Management (ECM) system or longtimestorage and a separation of data (hash-trees) needed for preservation of evidential value."[4] [Feda]

"Technical solutions of the middleware must make secure administration and configuration possible."[5] The document also describes acceptable formats for long-time storage of electronically signed documents in Chapter 4 of the attachment F [Fedb], mentioning XML as a good format for meta-data, which describes the signed documents and either ASCII, PDF/A, ODF, TIFF, JPEG, PNG or various other multimedia formats for the documents which will be signed. Specifically the format PDF/A-1 has been approved for long-time storage since

[4] Author's translation of A6.1-3: Die TR-ESOR-Middleware soll in der Lage sein, getrennte Mandanten zu verwalten. Dies bedeutet insbesondere eine strikte (logische) Separierung der im ECM/Langzeitspeicher abgelegten Archivdatenobjekte aber auch eine Trennung der für den Beweiserhalt relevanten Daten (Hash-Bäume).

[5] Author's translation of A6.1-4: Technische Lösungen für die Middleware müssen eine sichere Administration und Konfiguration unterstützen. [Feda]

2005 through the ISO standard 19005 [isob]. To ensure platform-independence, the data needs to be Base64 encoded and stored together with the meta-data, in an "XML-formatted archive data object" (XAIP) [Feda], according to the guideline. Further, Chapter 5.2 describes organizational measures to be taken to ensure compliance, while Chapter 7 focuses on describing a referential architecture with the three components ArchiSafe-module, ArchiSig-module and Krypto-module. This described architecture is not a requirement, but gives clear guidelines what a compliant software-system and the interaction within would look like. In this context the ArchiSafe-module is a component which manages information-flow and supplies a sort of bridge between all involved components, which are the business-application, Krypto-module, ArchiSig-Module and the longtimestorage-solution. The ArchiSig-module supplies the proper functionality to ensure preservation of evidential value when signing or re-signing documents. It connects directly to the Krypto-module, the longtimestorage-solution and the ArchiSafe-module. Lastly, there is the Krypto-module, which provides means to create and optionally verify electronic signatures. The interaction of all of the components is described in specific detail with sequence-diagrams and step-by-step instructions. [Feda]

2.2.2 Data Protection Laws

In 2012 the European Commision created a draft for the General Data Protection Regulation (GDPR) [Eur], which is meant to be valid throughout the European Union and would replace the German Bundesdatenschutzgesetz once it takes effect at the proposed release date of 2017 [Ros15]. This law would bring together the many different data protection laws of the various EU countries and make European-wide regulations legally binding, compared to the EU Data Protection Directive 95/46/EC [eu9]. Until this new unified law takes effect, the German Bundesdatenschutzgesetz (BDSG) [bds] is still the law that is to be applied within German institutions. It exists to protect individuals from being limited in their rights through the handling of their personal data and regards public institutions within the federation and public institutions of its states, as far as these don't have their own state-laws. In the case of the Leibniz Supercomputing Center (LRZ) [lrz], the state-law would be the Bayerische Datenschutzgesetz (BayDSG) or Bavarian State Law for Data Protection and Privacy [bay]. This law has jurisdiction only within Bavaria and for its institutions. Anything that isn't covered by this state-law is handled by the BDSG.

There are many different things to watch out for when dealing with data protection within organizations and when handling personal data [bds]: The previously mentioned BDSG in §4 specifically defines when and how it is admissible to gather, process or use personal data and §11 states measures to be taken if this gathering, processing and use happens on contractual basis. It clearly states in §4a that involved people need to consent to their data being gathered, processed and used to protect their privacy. To ensure compliance, §4f requires the organization to elect a Commissioner for Data Protection when automatic processing of personal data is taking place within procedures. The commissioner is in charge of reporting the points mentioned in §4e in case automatic processing of personal data takes place. He also has other responsibilities like making sure that systems comply with existing standards, laws and regulations and that employees are informed of how to properly deal with personal data within the organization. The law states that these public or non-public institutions must ensure that concrete measures are taken to ensure compliance with all mentioned laws and regulations when dealing with personal data as described in §9. §§12-19 specifically define

where and how data should be gathered, stored, processed, used, transferred and what to report to who when being involved in any of these tasks.

When handing over documents to courts or authorities in Germany, it is also specifically necessary to ensure the authenticity and integrity of the electronically transferred documents. According to the Law on Offences[6] (OWiG) in § 110a [owi] it states that documents handed over to authorities or the court must be signed with a "qualified electronic signature"[7]. Alternatively other secure processes can be used, if these ensure the "authenticity and integrity of the electronically transferred document"[8] [owi].

2.2.3 Procedure Descriptions

According to article 26 of the Bavarian Data Protection Law[9] (BayDSG) [bay], it is necessary to approve procedures involving the automatic processing of personal data through written form by the public institution that wants to utilize the procedure. Procedure descriptions are meant to hold the following data according to article 26:

1. "Description of the procedure,
2. Purpose and legal grounds of the inquiry, processing and usage,
3. Type of stored data,
4. Circle of parties concerned,
5. Type of regularly transferred data and its recipients,
6. Statutory deadlines for the deletion of data or for the review of the deletion,
7. Processing- and usage-eligible groups,
8. In case of article 6 passage 1 to 3 the contractors,
9. Recipients of provided data transfers to third Countries." [bay][10]

[6] Author's translation of Gesetz über Ordnungswidrigkeiten
[7] Author's translation of: qualifizierten elektronischen Signatur
[8] Author's translation of a part of §110a OWiG: Authentizität und die Integrität des übermittelten elektronischen Dokuments [owi]
[9] Author's translation of Bayerisches Datenschutzgesetz
[10] Author's translation of article 26, passage 2:

a) Bezeichnung des Verfahrens,
b) Zweck und Rechtsgrundlage der Erhebung, Verarbeitung oder Nutzung,
c) Art der gespeicherten Daten,
d) Kreis der Betroffenen,
e) Art der regelmäßig zu übermittelnden Daten und deren Empfänger,
f) Regelfristen für die Löschung der Daten oder für die Prüfung der Löschung,
g) verarbeitungs- und nutzungsberechtigte Personengruppen,
h) im Fall des Art. 6 Abs. 1 bis 3 die Auftragnehmer,
i) Empfänger vorgesehener Datenübermittlungen in Drittländer. [bay]

These procedure descriptions must be in written form and be filled out before deploying the new service, which then are to be approved by the responsible data protection commissioner of the institution, as article 26, paragraph 3 states [bay].

In addition, article 27 states that it is important to manage a complete procedure registry in which all procedure descriptions are kept and should be made available for free to any interested party. This registry is to hold procedure descriptions for all used automatic procedures that process personal data, with the previously mentioned attributes [bay]

The term "written form", within paragraph 1 of article 26, has a special meaning in this context. It means the documents must be created, approved by hand written signature and stored on paper according to §126 of the German Civil Code (GCC) [Gar].

2.2.4 Digital Signatures

As mentioned in Chapter 2.2.3, the German Civil Code requires a hand written signature on documents of written form for them to be legally admissible when it's stated by the involved law. One of these cases was the case where procedure descriptions need to be approved in written form. Since 2001 there has been an addition to §126, which states:

"The written form can be replaced by the electronic form, if the law does not state anything contrary." [bgb][11]

This "electronic form" is described in §126a of the GCC and requires the issuer to "add his name to the electronic document and sign it with his qualified electronic signature, as stated by the Signature Law"[12]. This alternative to handwritten signatures opens up the possibility to store documents digitally without the need to keep physical copies, but also brings with it new requirements, which need to be fulfilled. Not only are qualified certificates needed, but qualified signature creation units and means to secure the system and documents from manipulation. §2 of the Signature Law [siga] defines the needed technical terms: *Qualified electronic signatures* are advanced electronic signatures, "which have been based on a valid qualified certificate at creation-time and were created with a secure signature creation unit"[13] [siga]. *Advanced electronic signatures* in turn are "electronic signatures, which can only be associated with the signature-key-holder, allow for identification of the signature-key-holder, were created with means only the signature-key-holder has control over and which is related to the data it relates to in a way that allows for retroactive identification of manipulation of the data"[14] [siga]. *Electronic signatures* are "data in electronic form, which is added to other

[11] Author's translation of paragraph 3 of §126: Die schriftliche Form kann durch die elektronische Form ersetzt werden, wenn sich nicht aus dem Gesetz ein anderes ergibt. [bgb]

[12] Author's translation of last part of paragraph 1 of §126a: seinen Namen hinzufügen und das elektronische Dokument mit einer qualifizierten elektronischen Signatur nach dem Signaturgesetz versehen. [Gar]

[13] Author's translation of parts of § 2, paragraph 3:

1. auf einem zum Zeitpunkt ihrer Erzeugung gültigen qualifizierten Zertifikat beruhen und
2. mit einer sicheren Signaturerstellungseinheit erzeugt werden,

[siga]

[14] Author's translation of parts of § 2, paragraph 2:

1. ausschließlich dem Signaturschlüssel-Inhaber zugeordnet sind,
2. die Identifizierung des Signaturschlüssel-Inhabers ermöglichen,
3. mit Mitteln erzeugt werden, die der Signaturschlüssel-Inhaber unter seiner alleinigen Kontrolle halten

electronic data or is logically connected to it and is used for authentication of the data"[15] [siga].

It is to note, that according to §7 qualified certificates are to hold various attributes, such as the name of the signature-key-holder, the assigned signature-verification key, used algorithms, the time frame of it's validity and many more [siga].

There are also mentions in §17, of the products which can be used to create qualified electronic signatures, called Secure Signature Creation Units[16] (SSEE). These should make sure to create unique signature-keys, keep them secure and private and recognize unauthorized manipulation, while not allowing for external storage of keys [siga]. One of these products is the German, state-provided personal identification card. It can be read and used by the Federal Printing Office's sign-me product-line, which requires the most expensive *REINER-SCT cyberJack®RFID komfort* card reader. In case of the need for accreditation, issuing of qualified certificates, recognition of verification- and confirmation-authorities and supervision of various tasks, the responsible agency will raise further charges as defined in §22 [siga]. These requirements for special products and the need for the respective authorities to be involved makes using qualified electronic signatures costly with yearly or one-time fees adding up [siga].

When it comes to long-time storage of documents with qualified electronic signatures, the Signature Regulation[17] [sigb] states in § 17, that qualified electronic signatures "must be re-signed if the signatures are to be stored longer than the involved algorithms and parameters are meant to be suitable". These signatures are to be re-signed before the algorithms and parameters are declared unfit. The re-signing of these documents can be omitted, if qualified eletronic timestamps are applied to the old signature and the timestamp carries its own qualified electronic signature. [sigb]

Since the 17th of September 2014, the regulation No 910/2014 of the European Parliament and of the council on electronic identification and trust services for electronic transactions in the internal market (eiDAS-VO) [eid] is active throughout Europe and needs to be implemented, starting on the first of July 2016 [eid]. This new regulation has the effect, that starting in July 2016, electronic identification units of all member-states must be accepted. In addition the previous European directive 1999/93/EC [199] has been replaced. The directive in addition to qualified electronic signatures introduces qualified electronic seals, which act much like signatures in that they ensure "integrity" and "correctness of origin" as stated in article 35 [eid].

---

kann, und

4. mit den Daten, auf die sie sich beziehen, so verknüpft sind, dass eine nachträgliche Veränderung der Daten erkannt werden kann,

[siga]

[15] Author's translation of § 2, paragraph 1:
Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen, [siga]

[16] Author's translation of Sichere Signatur Erstellungs Einheit

[17] Author's translation of Signatur Verordnung

2.2.5 Qualified Electronic Timestamps

In addition to qualified electronic signatures, there are qualified electronic timestamps, which rely on qualified electronic signatures to ensure their validity. According to the definition number 14 of § 2 in the Signature Law, a qualified electronic timestamp is an electronic certification from a certification service provider, who fulfils the requirements mentioned in the Signature Law, "that certain electronic data has been presented to it at a specific point in time". [siga]

This qualified electronic timestamp can be used to verify the presence of the document at that specific time in the form in which it was presented. Considering § 371a, paragraph 1, sentence 2 of the Code of Civil Procedure[18] (ZPO), this qualified electronic timestamp can be used in court to prove the authenticity and genuineness of the provided time when the hash value and therefore the original document was provided. In combination with § 416, § 416a and § 437 of the ZPO, this supports the assumption of the court, that the signed date of the qualified electronic timestamp can be accepted as proof and therefore the document is genuine. [zpo]

[18] Zivilprozessordnung

3 Requirements

As described in the guidelines and regulations mentioned in Chapter 2.2, it is necessary to not just look at the requirements of the law, but also the requirements of the institution or company in which the software is to be used. To define these organizational requirements, this chapter will look at different scenarios in which the software is to be used and which steps are involved. At the end a summary of all found requirements is given.

3.1 Functional Requirements

When deciding which functional requirements are minimally necessary, it is important to take a look at existing processes to analyse who is involved, how workflows take place and which tasks are performed. The following workflows represent processes which are regularly performed in the context of procedure descriptions and their management.

Chapter 2.2.2 mentions that the commissioner for data protection "is in charge of reporting the points mentioned in § 4e [bds] in case automatic processing of personal data takes place". Part of those points are the name of the company or public institution, names of owners, directors, address of the institution and many more. [bds] In addition Chapter 2.2.3 makes it clear that additional information needs to be gathered and captured in written form in case of automatic processing of personal data. This data then needs to be signed off and approved by the commissioner and stored in a procedure description registry, which is openly accessible by any entity interested in the information. As shown by the workflow seen in Figure 3.1, there are different steps that need to be looked at. The figure describes a possible workflow when creating a new procedure description. Person A, who is in charge of a service in step 1 creates a new procedure description for his new service in which all the relevant data is captured and described, in compliance with the required information mentioned in article 26 of the BayDSG and § 4e of the BDSG. Person A then in step 2 hands it over to the commissioner of data protection, who in step 3 signs and in so doing approves the procedure description. This approved document then is put into an indexed folder in step 4, which stores the documents in an ordered and grouped way so it can be easily retrieved at a later time. Lastly the folder is placed in a filing cabinet of some sort with all the other folders and procedure descriptions of the last years and other important documents in step 5. This last step could also be the process of adding parts of the document to the publicly available procedure description registry, which is mentioned in article 27 of the BayDSG.

The mentioned scenario provides a few insights into which requirements a software-solution for this workflow would need to fulfil. The following points are derived from the illustration.

1. Creation of Documents The software should be able to store all the information and points mentioned by the laws in the form it requires. Therefore a responsible entity needs to

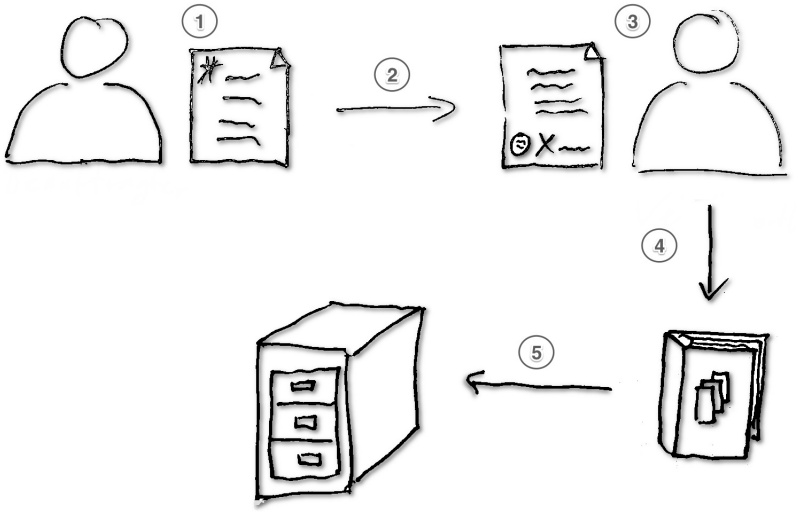

Figure 3.1: Workflow for creating Procedure Descriptions

be able to create new documents which hold the information in compliance with article 26 of BayDSG [bay] and § 4e of the BDSG [bds]. The creation of documents is mandatory.

2. Approval of Documents Next the commissioner needs to be able to approve the document. If it is to be stored entirely in digital form, it needs to carry a qualified electronic signature as stated by the Signature Law [siga]. This is mandatory to comply with the needs of article 26 of the BayDSG [bay]. To comply with the law, procedure descriptions must be approveable, which makes this a mandatory requirement.

3. Authenticity It is necessary to be able to verify a document's authenticity. Not only is this useful to identify the commissioner of data protection after approval, but it is also a requirement of the ISO 15489-1:2001 standard [Intb] and, for documents to be accepted as evidential value when used in court cases as stated in § 110a of the Law on Offences (OWiG) [owi]. Since procedure descriptions include information about processing of personal data by the public institution, it is likely that a case could develop in which this information is relevant. Therefore this is a mandatory requirement.

4. Adding to Procedure Description Registry After the document is approved, it should be added to the procedure description registry, making certain information accessible to interested entities, besides the creator of the document or the commissioner. The registry itself is different from the pure storage of procedure descriptions in that it publishes only specific information and should be accessible by anybody interested. The publication of the specific information is mandatory in accordance with article 27 of the BayDSG [bay]. Providing of this information and making it public means to make it available to specific entities upon request. It does not mean there actually needs to be a registry everyone

can access at all times, or that needs to be accessed by anybody besides the data protection commissioner. When a request is filed, the commissioner can extract the requested procedure descriptions and hand them over to the requesting party. This feature therefore can be seen as nice to have.

5. Archiving of Documents Similar to the manual process that is currently in place, the documents should be able to be archived for retrieval and analysis at a later time. In the context of this thesis archiving means the possibility to connect to an archiving system which fulfils the requirements of or similar to those mentioned in the BSI 03125 TR-ESOR guideline [Feda]. The possibility to connect to an archive server is optional.

6. Provide Archiving on System Providing the possibility to store documents over their entire life-time, especially within the system which is subject of this thesis, is beyond its scope. The requirement to be able to archive the documents without further work necessary on the system itself or through the system itself can therefore be seen as nice to have requirement.

7. Workflows Looking at the last points, it becomes clear that there will need to be some kind of workflow. In all, there are three basic groups of people as described in article 26 and 27 of the BayDSG [bay] with access to the information of the software:

- The contact person of the service to be approved, who creates the procedure description.
- The commissioner for data protection, who reviews the procedure description, approves it and adds it to the procedure description registry
- The interested party, that wants to have access to the registry

Workflows in this context are an optional requirement, because workflows have more to do with the area of usability and they are not mandatory to comply with laws.

8. Permission Handling Since there are different roles in the system and only certain roles in the system have certain tasks they get to perform, permission handling is necessary. Only the commissioner should be able to approve and sign documents. Only the contact person and the commissioner should be able to create and edit documents. Interested parties should not be able to see all information, but only what is published within the registry or what is provided to them. This also reflects requirements proposed by ISO 15489-1:2001 [Intb], the VOI regulations [VOI] and the BSI 03125 TR-ESOR guideline [Feda]. Permission handling is mandatory, since it is required to ensure only approved personnel can see all the data or perform certain actions, which helps prove integrity and accountability for the system.

The previously described scenario is only one of many possible workflows that can occur in normal operations. Another possibility would be one similar to the one shown in Figure 3.2. It depicts the steps needed to make changes to procedure descriptions. Such a workflow can become relevant, when requirements of a service change over the time of operation or a procedure description needs to be re-evaluated. Re-evaluation is already part of normal operations at the Leibniz Supercomputing Center, as well as normal modifications, after the initial description has been approved. Step 1 shows the process of retrieving the procedure

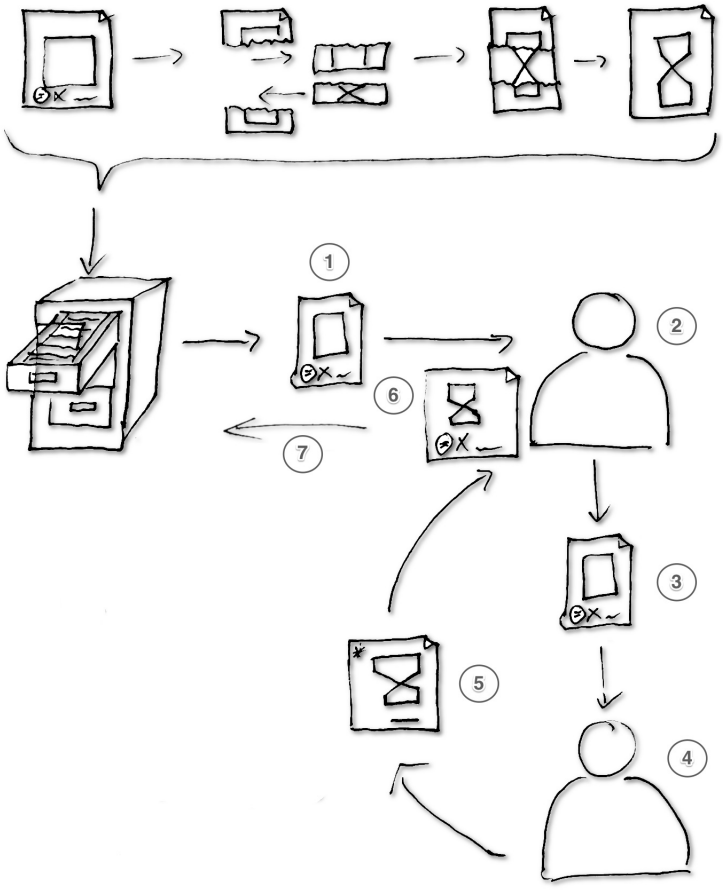

Figure 3.2: Workflow for revising Procedure Descriptions

description from the location where it's stored at. This in most cases is some kind of folder or filing cabinet with an indexing scheme and order to quickly find needed documents. The commissioner of data protection then evaluates the current description in step 2 and hands it over to the person responsible for running the service in step 3. The manager of the service then reviews the document, makes changes as depicted in the top of the illustration in step 4 and hands it back to the commissioner in step 5. The commissioner checks if all the changes are compliant with the law, actually describe the inner workings of the service and hold all the information needed. He then in step 6 signs the changed document, which approves the changes made. Lastly the document is returned to the filing system in step 7, where it gets added at it's correctly indexed position.

9. Retrieval of Documents To quickly review documents and make changes, it is necessary that documents are stored in an indexed form and are quickly retrievable. There needs to be the possibility to retrieve any document that is present in the system in a timely manner. This requirement is mandatory since it needs to be possible to retrieve specific documents in case of a court ruling, to review them regularly or to comply with internal or external audits.

10. Sorting of Documents and Content within Since there might be a need to change documents and re-approve them in an emergency, it should also be possible to sort procedure descriptions by different attributes of their content easily. This way the performance efficiency of the commissioner and other involved entities can be improved. This in some way represents the current setup with folders where procedure descriptions are either sorted by date or by name. Basic sorting is a mandatory requirement, since it otherwise would be less efficient than using paper-based archives.

11. Searching of Documents and Content within Just like with sorting documents, it should also be possible to search procedure descriptions and their contents easily. This has similar benefits as the sorting requirement. Basic searching is a mandatory requirement, since it otherwise would be less efficient than using paper-based archives.

12. Complex Sorting Complex sorting and multiple attribute sorting are not mandatory to comply with laws or regulations and can be seen as optional.

13. Complex Searches Complex searches, multiple attribute searching and content-searches are not mandatory to comply with laws or regulations and can be seen as optional.

14. Version Control and Change Logging When making changes to procedure descriptions it should at all times be possible to track who made the changes, what was changed and when these changes were made. Original documents must not be altered and should stay available. This represents an additional step, which is not depicted in the workflow of Figure 3.2. The original document, which is used to make the changes, is returned to the filing system after it's use or only a copy of the document is handed to managers. This way it is possible to track changes over the course of the document life-cycle. This is a mandatory requirement to ensure the reliability of the system.

15. Re-Approval of Documents Procedure descriptions regularly need to be re-evaluated and re-approved. This could be in yearly, monthly or any other interval form. The possibility to review and re-approve documents is optional, since this use-case is not specifically mentioned in any laws or regulations and can be covered with the approval of documents after changing existing procedure descriptions. In this case the data protection commissioner only needs to regularly check if documents should be evaluated again. There is no need to make this a separate feature.

16. Automatic Regular Re-Submission of Documents To improve the workflow and to support the commissioner, procedure descriptions could regularly be brought up by the system for re-evaluation. There are no mentions of the need to re-evaluate documents after specific times in any of the laws. The complexity the feature would add to the system and the possibility for cluttering of the interface also cannot be justified by the possible benefit it could bring. Therefore automatic re-submission and reminders for procedure descriptions should be seen as a nice to have feature, since it would only make the life of the data protection commissioner easier while not adding a lot of benefit.

17. Export of Documents Documents need to be exportable for outside use when needed. To achieve this, a common, transportable, compatible format should be chosen, which can be accessed and interpreted by standard software and sent over channels without loss of data. Without this functionality, documents and information remains proprietary to the developed system and can not be saved or transferred to new or other systems. To ensure the long-time compliance with the law, exports are a mandatory requirement.

18. Import of Documents Imports of documents are not necessary to comply with laws or regulations and could be performed manually if needed. Therefore this requirement is optional.

19. Backup To allow for fast recovery after a system failure or after an attack, it needs to be possible to restore the system to its original condition. This requirement is different from other forms of archiving, since the system needs to be restore-able into it's productive form with all previously available documents ready for inspection, editing and approving. For this to be successful, a working backup-solution needs to be in place. It ensures protection against data-loss, which is mandatory to remain reliable.

20. Statistical Analysis Building on the versioning requirement, it can be necessary to evaluate how descriptions have developed over time and which points keep returning on a regular basis. To give a good overview, the system should therefore allow for statistical analysis of procedure descriptions and it's contents over time. This is neither required by law, nor is it part of regulations or guidelines. In the context of the developed concept it is a nice to have feature.

3.2 Non-Functional Requirements

Just as the functional requirements must be defined, it is also necessary to evaluate possible non-functional requirements. This section analyses specific scenarios and the previously mentioned laws and guidelines of Chapter 2.2 for non-functional requirements.

In Section 3.1 the two workflows describe situations in which at least two parties are involved. On the one hand the commissioner of data protection and on the other the manager of a service to be described. When looking at more than one service, it can happen that more than one person is working on the system, each with different procedure descriptions. E.g. user A is reviewing a document, while user B is editing a second document. In the meantime, user C is approving a third document. All users are working on the same system, at the same time. Without the system, no procedure descriptions can be accessed and nobody can edit or approve documents. Manual creation with alternative means would be necessary.

21. Availability Since the system might be the only way to access the procedure descriptions in a fast way, the system should be available whenever needed. This means that it should not be blocked by multiple users using it, needs to be monitored for uptime and brought back up in less than the time specified by the data protection commissioner. This time may vary from

company to company. Without this requirement, the efficiency of the commissioner of data protection and the managers can fall back to or become less than when using a paper based solution, since alternatives must be used during the downtime. This is also mentioned in the standard ISO 15489-1:2001, the VOI guidelines and the BSI 03125 TR-ESOR guideline. To comply with later set service level agreements (SLA) the preparation for this is a mandatory requirement.

22. Integrity To ensure that documents are unchanged and can be retrieved in the way they were stored, means to ensure the integrity of the system and of the documents need to be present. This reflects the recommendations and requirements of the standard ISO 15489-1:2001, the VOI guidelines, the German SigV, ZPO and § 110a OWiG. Therefore it is a mandatory requirement.

23. Documentation The system's functionality and composition must be documented in a way that allows for understanding of how it works and which components are involved, to evaluate its value and security. This includes documenting which algorithms and software packages are used, how the system is to be operated and interacted with, how to handle security-relevant situations and which steps are involved when processing the data. To be accountable, this is a mandatory requirement.

24. Usability Users must be able to understand and use the software without extensive training to ensure the solution is not more complicated than using the printed and paper-based version. This means that the software should be built to be self-explaining and support the user in a non-invasive way when performing tasks. This is not required by law, regulations or guidelines. It is an optional requirement, even though it would increase the value and quality of the software overall.

25. Price The developed system must be affordable and should not drastically exceed the costs of maintaining a paper-driven solution. This is a requirement of the public institution and is to be seen as mandatory to achieve the most cost-efficient solution.

3.3 Summary of Requirements

The laws and regulations describe many mandatory and recommended requirements which should be part of a legally compliant document storage solution which preserves the evidential value of the documents. In addition there are user stories, which put these laws and regulations into perspective and add to the requirements. The following table lists all found requirements from guidelines, laws and previously mentioned scenarios for creating a storage solution for procedure descriptions with preservation of evidential value. It lists them with columns holding the name, short descriptions and the priority with which the requirement is to be considered. Priority *M* stands for mandatory requirements which need to be present. *O* represents optional requirements, which increase the quality of the software but aren't mandatory to comply with the law. *N* are nice to have requirements, which are neither mandatory, nor optional to comply with laws or to increase the quality of the software.

	Requirement	Description	Prio.
1	Creation of Documents	Scenario 1, BayDsg - Performing the task of creation of procedure descriptions	M
2	Approval of Documents	Scenario 1, BayDsg - Performing the task of approval of procedure descriptions	M
3	Authenticity	OWiG, ISO 15489-1:2001, VOI, BayDsg - Proof, that people and documents are who and what they are	M
4	Addition to Registry	Scenario 1, BayDsg - Adding new and changed procedure descriptions to the procedure description registry	O
5	Archive Documents externally	Scenario 1, VOI, BSI 03125 TR-ESOR - Save or export documents to archiving server	O
6	Archive Documents internally	Scenario 1, VOI, BSI 03125 TR-ESOR - Provide archiving solutions on server itself	N
7	Workflows	Scenario 1 and 2 - Provide possibility to automatically forward documents within a workflow system	O
8	Permission Management	Scenario 1 and 2, ISO 15489-1:2001, VOI, BayDsg, BSI 03125 TR-ESOR - Manage permissions and only allow approved personnel to perform certain actions	M
9	Retrieval of Documents	Scenario 2, ISO 15489-1:2001, VOI, BSI 03125 TR-ESOR - Allow for retrieval of documents within a reasonable time	M
10	Simple Sorting	Scenario 2, ISO 15489-1:2001, VOI - Allow for simple sorting of documents and their content	M
11	Simple Searching	Scenario 2, ISO 15489-1:2001, VOI - Allow for simple searches over documents and their content	M
12	Complex Sorting	Scenario 2 - Allow for complex sorting of documents and their content	O
13	Complex Searching	Scenario 2 - Allow for complex searches over documents and their content	O
14	Version Control and Change Logging	Scenario 1 and 2, ISO 15489-1:2001, VOI, BSI 03125 TR-ESOR - Track changes and versions to be accountable	M
15	Re-Approval	Scenario 2 - Allow for re-approval of documents	O
16	Automatic Re-Submission	Automatically have the data protection commissioner reminded to review and re-approve documents	N
17	Export	ISO 15489-1:2001, VOI, BSI 03125 TR-ESOR - Export documents or save them externally	M
18	Import	Import documents from external sources	O

19	Backup	VOI, BSI 03125 TR-ESOR - Allow for backups of the system to ensure reliability and availability in the long run	M
20	Statistics	Provide possibilities to extract statistical data from the system	N

Table 3.1: Functional Requirements

21	Availability	Scenario 1 and 2, ISO 15489-1:2001 - Ensure that the system is available and can be used when needed	M
22	Integrity	Scenario 2, OWiG, ISO 15489-1:2001, VOI, BSI 03125 TR-ESOR - Ensure integrity of the documents and the system	M
23	Documentation	ISO 15489-1:2001, VOI - Provide documentation for the system on how information is processed, handled and how the system works overall	M
24	Usability	Allow for easy and self-explanatory usage of system	O
25	Low Price	LRZ - Find the most cost-efficient solution	M

Table 3.2: Non-Functional Requirements

4 State of the Art

In this chapter current solutions for the digital management of procedure descriptions are analysed. First, specific software solutions are mentioned that could possibly be used to manage procedure descriptions. Following that, a current solution actually being used at one selected institution is mentioned. Next, current solutions and providers for creating qualified electronic signatures and timestamps are analysed. Finally, all findings and a summary are given in the evaluation section at the end of the chapter.

4.1 Procedure Description Management Solutions

This subsection analyses existing solutions for management of documents in context of data protection guidelines, what their positive traits are and where they lack functionality.

In an e-mail exchange with Gerhard Kron, CEO of Kronsoft e.K. [Kroa], who is responsible for the creation of the product *opus i* [Krob], a leading data protection software solution, it quickly became clear that basically none of the requirements mentioned in Chapter 3 are being met[1]. Specifically regarding authenticity of documents and people, Mr. Kron's software could not be of any help since there is no qualified signing solution in place. Although according to the website [Kroa] all processes of the data protection laws are supported, it is not possible to approve these and store them in the way specified by the requirements of this thesis. Further research into the software solution *opus i* is therefore not beneficial.

opus i and many other solutions like it originated from the previously widely used GSTool [Fedc], which was provided by the Office for Security in Informationtechnologies[2] (BSI). The distribution of this solution was discontinued on the 31.12.2014 and the software will no longer be supported by the end of 2016 [Fedc].

A study exists which analyses alternatives to the GSTool. This so-called CSC-Study to evaluate GSTool alternatives with the name *GSTOOL QUO VADIS?* [Com], mentions eight alternatives altogether, mainly

- DHC Vision Information Security Manager 5.2 [DHC]
- HiScout GRC Suite 2.3 [HiS]
- iris (Version 15/R2 (Build 15107.1)) [ibi]
- Opus-i (Oktober 2014) [kroc]
- QSEC V4.2 [WMC]
- SAVe V4.4 bzw. V5.0 [INF]
- Sidoc (Oktober 2014) [2ne]
- Verinice 1.9 [Ser]

Similar to the *opus i* solution, none of these solutions provide functionality to approve documents

[1] G. Kron (personal communication, November 21, 2015)
[2] Author's translation of Bundesamt für Sicherheit in Informationstechnik (BSI)

purely in the application itself. They do allow management of documents, even in a versioned manner, together with import and export from and into different formats. But none of them are specifically designed for the management of procedure descriptions or their storage. Due to the full integration with every aspect of the BSI Base Protection[3], all of these tools become very complex to manage, which makes adoption more difficult. All solutions besides Verinice use license models for their basic features, for which either monthly, yearly or one-time fees are charged. When it comes to decentralized management and storage of documents, even Verinice (which is the only open source based solution) charges extra to provide these features. The previously mentioned support for many other uses besides procedure description management also increase the price of licenses, even though these features would not be used. Considering this, it can be derived that these solutions are not suitable for use in the specific case of this thesis and for the purely digital management of procedure descriptions.

When looking into how other institutions manage procedure descriptions, there does not seem to be a common way of managing these purely digitally. For instance Mr. Peter Grimm, faculty administrator at the University Regensburg [Unib] mentions[4] their solution to deal with procedure descriptions is to keep them in a versioned storage-solution called Novell Filr 1.2 [Fit]. This allows for the documents to be available, versioned and secure, since only authorized personnel can edit documents while all other faculty members can look them up [Unia]. But without the use of qualified electronic signatures, documents still need to be stored in physical form on paper and signed by hand and managed externally. Sorting and evaluating of these documents is only slightly easier than keeping them in folders only. This closely represents the situation which is also present at the Leibniz Supercomputing Centre (LRZ) and that could profit from a particular solution for purely digital management of documents of this type.

4.2 Qualified Electronic Certificates for Signatures

There are many providers for qualified electronic signature solutions. On the one hand, there are providers for the qualified electronic certificates that are needed to perform signatures. Next, there are providers for Secure Signature Creation Units[5] (SSEE) to create signatures using the previously mentioned certificates. Lastly, there are producers of software solutions to perform qualified electronic signing of documents. All of these solutions must comply with the Signature Law [siga] and the Signature Regulation [sigb] regarding qualified electronic signing. To ensure that signatures can be verified for 30 years after the signing process, it is necessary to pick a provider who is accredited, since only these certificate providers are required to store this information over that period of time and hand it over to the responsible authority, which makes sure all information and contracts stay available and intact according to the German Signature Law [siga] after they discontinue their services. Without accreditation this form of availability is not guaranteed in case of a termination of the service. The Federal Network Agency[6] supplies a list of all currently available providers for qualified electronic certificates [Wulb], of which only three are accredited, who plan on continuing their business and provide services to institutions like the LRZ or private customers:

- DGN Deutsches Gesundheitsnetz Service GmbH [DGNa]
- Deutsche Telekom AG [T-S]
- D-Trust GmbH [Buna]

The *DGN Deutsches Gesundheitsnetz Service GmbH* provides signature cards and card readers. The cheapest signature cards with qualified electronic certificate cost 47.60 Euros per year [DGNc], while

[3] Author's translation of BSI Grundschutz
[4] P. Grimm (personal communication, November 9, 2015)
[5] Author's translation of Sichere Signatur Erstellungs Einheiten
[6] Bundesnetzagentur

the card-reader *Cherry Smart Terminal ST-2000 UCZ* that can be used with the card costs 47.48 Euros as a one-time expense [DGNb]. The Deutsche Telekom AG provides signature cards called *PKS-ECC-Signaturkarte*, which for two years of validity cost 99.00 Euros [Deu] and can be used with any kind of card reader. The D-Trust GmbH is part of the Bundesdruckerei GmbH and provides signature cards starting at 109.00 Euros for two years of use, while also providing a test-card for only one year at the cost of 39.00 Euros [Gmb]. These cards can also be used with the earlier mentioned *Cherry Smart Terminal ST-2000 UCZ* card reader.

As of 2013, there is an alternative to using a card from the above mentioned suppliers, which uses the newly developed personal identification card that has been introduced within Germany [Thy] in 2010. This new personal identification card can be used to identify the card-holder online and to sign documents with a qualified electronic signature. All that is needed are a card reader and a qualified electronic certificate, which can be purchased from the Reiner SCT Shop [REI]. The certificate cost 9.98 Euros for a one-year license in December 2015, when it was purchased. By February 2016 the same certificate cost 29.90 Euros. The certificate is issued by the D-Trust GmbH, which is accredited and therefore ensures 30 years of availability of verification information after the validity of the certificate ends. To use the certificate on the personal identification card, the online-id functionality must be activated and a *REINERSCT cyberJack®RFID komfort* card reader must be purchased at a one-time price of 124.90 Euros [REI]. It is not possible to purchase a cheaper card reader, since it must be able to read the new personal identification card, which uses the RFID technology to transfer information and therefore must at least be a class 3 comfort card reader. It is possible to purchase the card reader and the certificate together for 149.90 Euros [REI]. To use the card reader, the included driver software must be installed together with the *AusweisApp 2* application [Gov].

4.3 Qualified Electronic Signature Software

To sign documents, not only qualified electronic signatures associated to a specific person and Secure Signature Creation Units[7] (SSEE) are needed, but also software which enables signing of documents. The Bundesdruckerei GmbH provides a list of software compatible with their sign-me certificates used on the new German personal identification cards [Bund]. Of those listed, only the SecCommerce SecSigner [Sec] solution is free to use. This solution is based on the Netscape Plugin Application Programming Interface (NPAPI), which loads a Java applet in a browser down onto the client computer to then communicate natively with the card reader. An alternative to this approach would be the *Sign Live! CC cloud suite bridge* from intarsys [inta]. It allows communication with smart card readers without going through Java Applets, by loading the Java applications through JavaScript and Java-Web-Start (JWS). To use this technology though, the expensive *Sign Live! CC cloud suite* can only be purchased upon special request and must be customized. The exact costs for this solution could not be established before the end of this thesis, since all prices vary upon individual use-cases, their implementation and especially the price for the *Sign Live! CC cloud suite bridge* (bridge) which has not been released yet. To provide some context, a ten-workstation *Sign Live! CC cloud suite* license costs $800 per year. If the bridge has prices close to this it is not feasible for the proposed project.

4.4 Qualified Electronic Timestamping

Just as with the providers of qualified electronic certificates, there are also different providers of qualified electronic timestamps which can be used to ensure the integrity of documents. As before, only accredited providers will be analysed who provide services to institutions like the LRZ and that

[7] Author's translation of Sichere Signatur Erstellungs Einheiten

plan to continue these services. According to the list of the Federal Network Agency[8] [Wulb], this leaves only two suppliers of accredited qualified electronic timestamps:

- D-Trust GmbH [Buna]
- exceet Secure Solutions AG [exc]

D-Trust provides a starter package for 199 Euros, including a soft-token and 500 signatures for one year [Bunb]. Further timestamps can be purchased for 70 Euros for 500 timestamps, up to 3000 Euros for 30000 timestamps while an extension of the service by one year costs 84 Euros [Bunb]. Client software compliant with RFC 3161 and a soft-token are necessary to request timestamps through this service [Bunb]. The exceet Secure Solutions AG does not directly provide the functionality of timestamping through their website. Instead, the company provides this functionality through their subsidary company *signamus*. Their website claims to provide qualified electronic timestamps for as low as 3.9 Euro-Cents as long as requests stay below 500 timestamps per month [SSDG]. A request for further information regarding prices or processes to timestamp documents was not successful.

4.5 Evaluation

After what has been presented so far, it becomes clear that there is no solution that fulfils all our defined requirements satisfactorily. Although there are plenty of solutions to manage documents as such or to create qualified electronic signatures, there has been no development to properly allow for procedure description management and it's storage for later admissibility within court.

Neither *opus i*, nor any of its competitors can provide the possibility to ensure authenticity of the processed documents, nor do they support qualified electronic signatures or the archiving at a later time. They are complex and loaded with features which are not needed in the context of the thesis, making them harder to learn, understand and use.

Qualified electronic signing solutions exist, but need to be implemented or integrated manually after selecting a suitable provider. Due to the need for accredited certificate and timestamp providers, the options of selection are very limited and can be expensive. An alternative could be the use of the new German personal identification card together with its qualified electronic certificate which can be purchased from the Reiner SCT shop.

Because of the aforestated restriction problems, it is necessary to develop a new solution that can fulfil the aforementioned requirements of Chapter 3 and so provide the possibility to ensure the integrity and authenticity of procedure descriptions.

[8] Bundesnetzagentur

5 Conception

This chapter defines the concept developed according to the requirements of Chapter 3. Essentially, the requirements boil down to nine core problems which must be solved:

- What, how and where information is stored in the system
- How to ensure long-term preservation of evidential value
- How to accomplish versioning
- How to ensure authenticity of documents
- How to ensure integrity of documents and the system
- How to handle access rights and permissions
- How to secure the system
- How to ensure availability
- How to accomplish workflows

All mandatory requirements can be answered, while considering and handling the other requirements within the same context but with less detail.

For this, the chapter analyses which steps are part of the concept and describes the organizational and technical guidelines by which the software should be implemented. The solution must be independent from specific products or brands. The chapter will first give an overview of what the finished workflow within the system will look like. It will then define the type and format of information which needs to be stored. The format of the to be stored documents, the format of the container-files and how to make changes to the format and layout of these templates later on. To be able to work with different versions of documents within the system, versioning is brought in to the picture and its setup is explained. Next, approaches to ensure authenticity and integrity are analysed and selected, describing the inner workings of the concept in this regard. Then, user actions and permissions to perform them are discussed. In the seventh step, approaches to secure the system technically and organizationally are defined, to secure the system against internal and external attacks. Workflows are discussed and handled second to last while considering possible actions that could be taken. Finally, the last section mentions possible courses of action to ensure that the application is available when needed.

5.1 Overview

To describe the components of the system seen in Figure 5.3 and define how they interact, this chapter will consider the overall workflow when working with the system and which steps are part of the process. The management tool has many different parts, which take care of version control, logging, permissions, creating, editing, viewing and approving documents, authentication and ensuring integrity within the system. Approval happens through the usage of the *Secure Signature Creation Unit* and the attached *Signature Creation Software* and approved documents are saved to a sync folder, which regularly gets synchronized to the *Archive Server*. This *Archive Server* also holds log files which ensure accountability towards the integrity of the system and its documents.

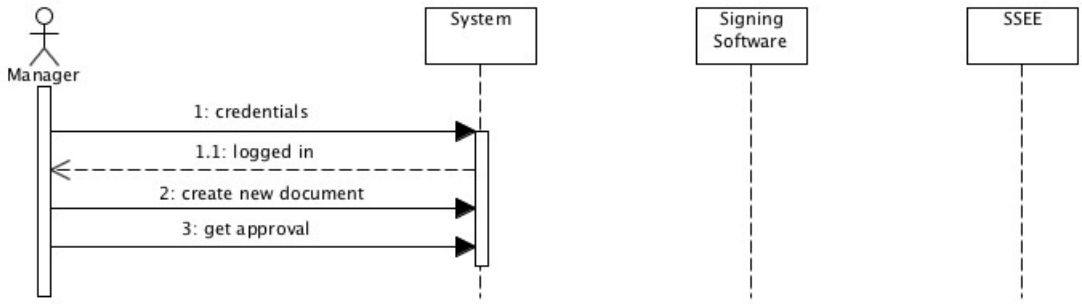

Figure 5.1: Sequence Diagram for creation of new document by Manager of service

The components themselves and the connections between them are explained in more detail in each section.

When managers of a service first log in to the system with their valid and verified credentials as seen in Figure 5.1, there is nothing to see or do besides create a new procedure description. When the *create* button is pressed, a form with a blank procedure description is presented, which can be filled out and modified to the liking of the manager. After making the needed changes to the document, it is saved by pressing an interface element, which afterwards takes the user to a listing or summary of all his existing procedure descriptions. The saved document is the first working copy for that user in the listing and gets the internal version number *1* and can be referenced by its id in the database. After the manager is finished with editing the procedure description and wants to have it approved, an icon or button can be pressed next to the document, setting a flag within it to show that the document is waiting for approval by the data protection commissioner. An e-mail could be sent out to the commissioner to inform him or the document could be added to a special list, which only the commissioner can see upon the next login.

Figure 5.2 shows the commissioner's interaction with the system when approving the document. When the data protection commissioner logs in, a list of all procedure descriptions awaiting approval is shown. After clicking on one of the documents, it can be reviewed and edited to the liking of the commissioner. To approve the procedure description, a button in the document-view is pressed, starting the approval process which opens a connection to the *Secure Signature Creation Unit*[1] (SSEE) through a secure native application and the attached card reader. A dialogue is opened in the application unit which directly interacts with the SSEE, showing the document to be signed and asking for confirmation. The commissioner accepts the presented document and signs it by typing in the needed signature pin. By confirming the signed document, a process is started to send the document with its signature back to the document management system, where it is processed and stored, creating a hash value for it, its signature and all other relevant documents, ultimately saving it to an archive-document. The archive-document, in turn, is written to the disk of the machine together with the information to ensure its integrity and authenticity and then synced to the archive server, where it cannot be manipulated any longer. While this is happening in the background of the system, the data protection commissioner is forwarded to the listing of all procedure descriptions, where the approval status of the document can be seen through a marker next to the document and a timestamp.

Overall there need to be views for all steps mentioned in figures 5.1 and 5.2. Mainly to insert, view, edit and approve procedure descriptions and to browse existing documents with searching and sorting. Additionally there needs to be a view to manage access rights to documents. Managing and

[1] Author's translation of Sichere Signatur Erstellungs Einheit

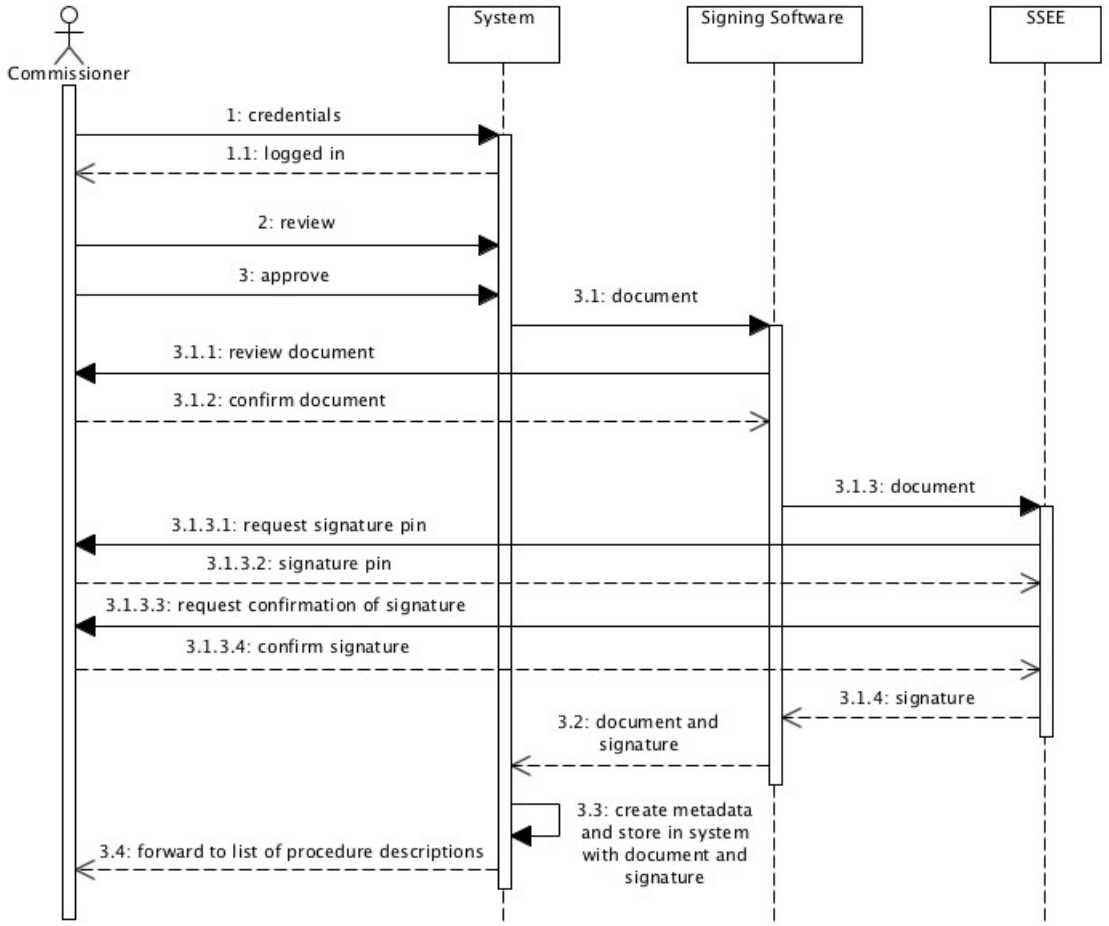

Figure 5.2: Sequence Diagram for approval of document by Data Protection Commissioner

verifying of documents, storing them securely and ensuring the system's integrity will automatically be taken care of on the server side of the system. Access to views and actions must be restricted to specific roles and managed through unique user ids and secure authentication mechanisms. But before the actual system can be designed, the data and its storage inside the system must be defined.

5.2 Integrity of Documents

Chapter 2.2.2 mentions that the Law on Offences[2] (OWiG) [owi] in Germany requires that the integrity and authenticity of documents handed into court is to be provable. How can the integrity for documents and the storing system be ensured? There are a few different possibilities to achieve this. The most simple form would be to timestamp the hash-value of every single document as soon as the document is created. This would ensure that it can be proven at a later point in time, that the document existed exactly in the form in which it was timestamped and hasn't been tampered with since then. With this form of timestamping high costs can develop over time, since qualified electronic timestamps are expensive compared to non-qualified, advanced or simple timestamps, especially if an

[2]Author's translation of Gesetz über Ordnungswidrigkeiten

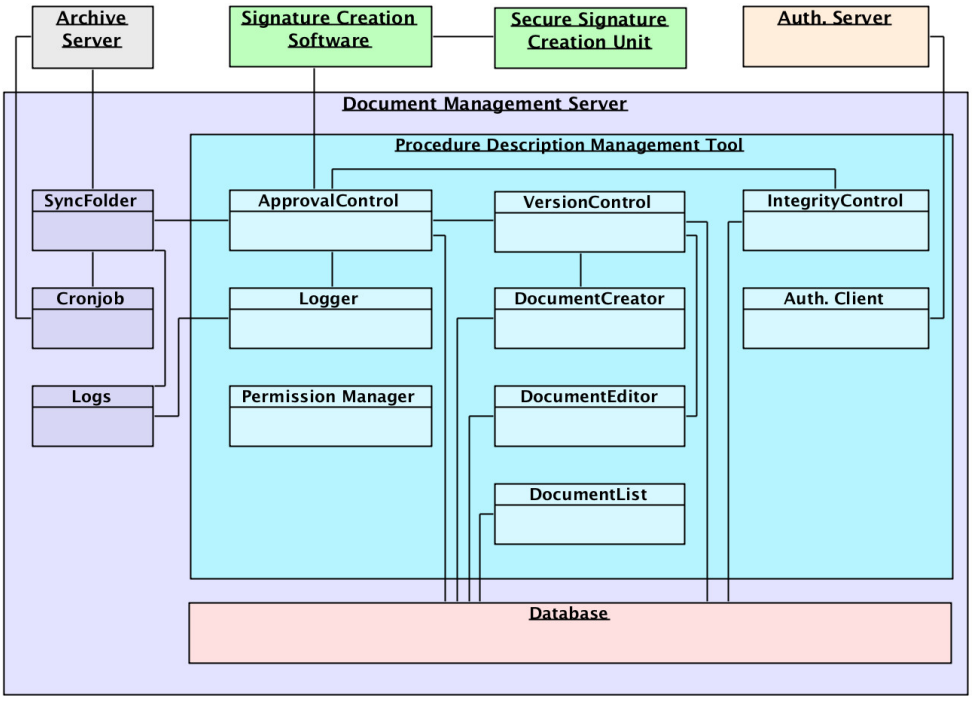

Figure 5.3: Architecture of System with Relations to internal and external Components

accredited provider is selected, as described in Chapter 4.4. Another downside is that the documents have no relation to each other, which would easily allow for a malicious attacker to just delete a document with its timestamp. A possible solution to these problems could be to use a linear hash-chain, in which the hash-value of a new document is hashed together with the head or root of a linear-hash chain. Figure 5.4 shows how such a linear hash-chain would be created. Step 1 creates a hash-value of Document 1, which is hashed together with the previous root-hash of the chain in Step 2, to create the new root hash (3). The next time a document is created, the hash value of that document is created in Step 4, hashed together in Step 5 with the root hash from 3 to create the new root-hash (6). The root hash could then be timestamped, which would in turn ensure the integrity of all files of the chain and the order in which they were added to the chain. To reduce costs of timestamping, timestamps could be applied only once a day, signing only the newest root-hash of the linear hash-chain. Although this takes care of the first two initial problems, it is hard to verify that documents are covered by this form of linked hashing, since every element of the chain must be gone through and the corresponding root hashes must be calculated. Therefore the verification process for element N (as seen from the current root) must calculate the hashes of all N-1 elements that come before it.

An alternative is the Merkle hash tree [Mer80] as seen in Figure 5.5, which consists of a tree structure, holding hashes of documents in its leaves and hashes of the children in the nodes. Its main advantage is, that it allows for verification of hashes in its leaves, without going through the entire tree-structure, but only branches related to the mentioned leaf-hash. This can be demonstrated according to Figure 5.5: Document 1, 2, 3 and 4 are hashed and their hashes make up the leaves of the Merkle tree. To get the next higher level (bottom to top), the hash values of the child-nodes are added together and hashed to create a node-hash for each branch. This goes on until it reaches the root hash, which can be seen as a representative of all hashes under it. If the integrity and validity

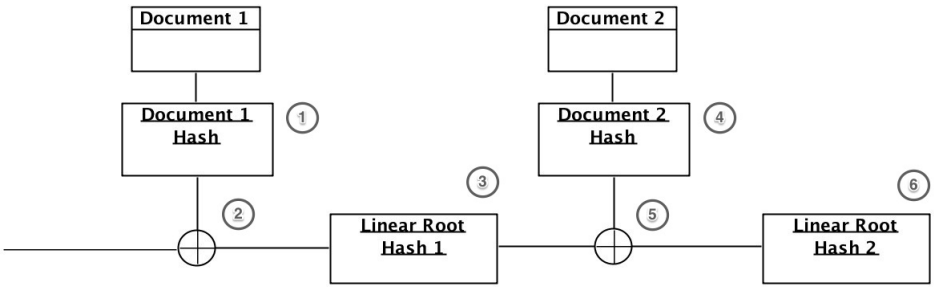

Figure 5.4: Linear Hash Chain

of Document 1 is to be verified under the root-hash, only the sibling-hash from Document 2, their parent-node, its sibling and the root-hash are necessary to calculate all relevant hash-values. For the verification of the hash of Document 1, the hash-values of Document 3 and Document 4 are not necessary. This reduces the computations needed down from N-1 to the height of the tree minus one. In the case of Figure 5.5, that would conclude to two calculations to verify any single one of the leaf-hashes in the tree. If any of the calculated hashes don't match up or cannot be used to rebuild the original root-hash, the document must have been changed, which explains the modified hash of the document and thus the non-matching root-hash.

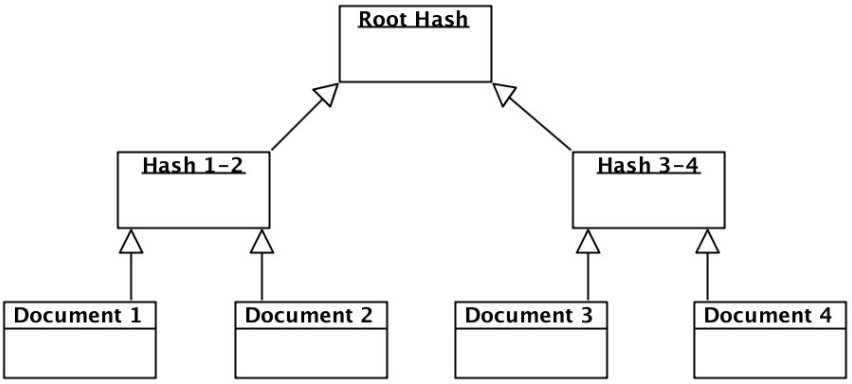

Figure 5.5: Merkle Tree

The component that takes care of integrity can be seen on the far right of Figure 5.3 and is connected to the database, being activated and run upon approval. To ensure integrity within archives and within the system, the files of the archive are added to a Merkle tree, which produces a root hash for the archive. Included in the files are the original document, its signature and all certificates and files needed to verify the authenticity and integrity of the document and signature. The resulting root hash receives an advanced timestamp which is stored together with the other documents in the archive. This root hash is then added to a second, system-wide Merkle tree containing the root hashes of all other archives and the root hash of the previous system-tree. By doing so, the system Merkle trees are linked, creating a history which can be verified. The generated system Merkle tree is then written into a log file together with its root hash, the ids in the order they were added to the Merkle tree and an advanced timestamp. The advanced timestamps would not hold up in court,

but they add a layer of protection while waiting for the daily qualified electronic timestamp that is added at a later point. The archives are stored in a specific directory on the server, together with the log-file and synchronized onto a secured Write Once Read Many (WORM) archiving system, where the hashes can no longer be altered. Once a day that archiving system gets a qualified electronic timestamp for the last root-hash in the log-file, to ensure provability of the existence of all documents that are part of the system Merkle tree. Since the log is eventually written to a WORM storage device, log-rotation must be implemented in a way that uses up-counting id-numbers or the current date within the filename to ensure pure appending.

5.3 Authenticity of Documents

As with integrity, authenticity of electronic documents must also be ensured, according to the Law on Offences[3] (OWiG) [owi] in Germany. In the special case of procedure descriptions the need for authenticity also arises out of article 26 of the Bavarian Data Protection Law[4] (BayDSG) [bay] together with §126a of the German Civil Code (GCC) [Gar] and their implication to sign documents. Authenticity of electronic documents can be achieved and proven with a qualified electronic signature, according to the German Signature Law (SigG) [siga]. To create the qualified electronic signature, a Secure Signature Creation Unit[5] (SSEE) is necessary in addition to a qualified electronic certificate, uniquely associated to the creator of the signature and issued by a qualified certification authority [siga]. To allow for later verification of the signature in accordance with its creator, the verification information must be publicly accessible and published through the certification authority and cannot be restricted in any way. If that is not possible, the verification information must be gathered before or as soon as the document and its signature are archived. The required verification information can be found in Section 5.4.3.

For the data protection commissioner to be able to create qualified electronic signatures for documents within the system, a provider of certificates must be selected who provides the possibility to create signatures for the type of documents stored in the system. This provider should be accredited and also provide the possibility to verify the certificates and signatures through its publicly available registry. The used solution to sign documents should be integrated into the system itself to allow for better usability, access-ability and accountability of the system. To make use of the issued certificate and to actually sign documents, a Secure Signature Creation Unit must be selected additionally, that supports the selected certificate and enables and supports qualified electronic signatures. Since the developed application is a web-application there will have to be a way to connect the user's hardware to the centralized service to exchange documents, sign them locally and forward the signed data back to the system. Only the signing process itself is performed on the user's computer. All of the creation, editing and further processing of documents happens in the background of the web-application on the developed system.

Figure 5.6 shows the process when approving documents with qualified electronic signatures. The component *ApprovalControl* in Figure 5.3 shows how the *Secure Signature Creation Unit* interacts with the *Signature Creation Software* to make this possible. After reviewing the procedure description, the data protection commissioner initiates the approval process in the web-application in Step 1, which sends a request (Step 2) to the main server to prepare the document for signing (Step 3), providing it to the application front-end (Step 4) and establish the connection to the Secure Signature Creation Unit through the provider's solution of connecting the web-application to the user's hardware in Step 5. The document is then transferred to the SSEE in Step 6 and signed through the user's acceptance of the signing process and providing a signature pin in Step 7. After signing, the SSEE sends the data back to the user's computer in Step 8 and provides the server with the signed

[3] Author's translation of Gesetz über Ordnungswidrigkeiten
[4] Author's translation of Bayerisches Datenschutzgesetz
[5] Author's translation of Sichere Signatur Erstellungs Einheit

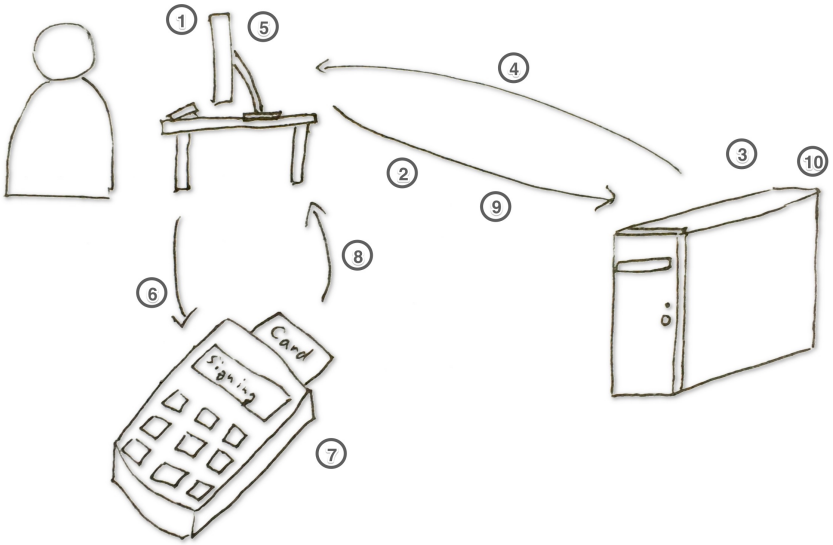

Figure 5.6: The approval process with qualified electronic signatures

document and its signature in Step 9, which verifies, accepts, processes and stores it in the final Step 10.

Interesting for this concept are the non-trivial steps involving the server, mainly steps 3, 9 and 10. In Step 3, the *content* section of the document is taken and together with the document template is transformed into the PDF/A-1 format, which was introduced in Section 5.4.4. This created PDF file is then encoded into Base64 to be transferred from the server to the client application (Step 4). Steps 4 to 8 and the interfaces to be used are strongly influenced by the signature creation software. Creating a signature creation software which supports qualified electronic signing and complies with the German Signature Law [siga] is both out of scope of this thesis and highly difficult since there are very strict requirements and strong restrictions on who gets to and how to interact with Secure Signature Creation Units. This part of the application must be selected from a third party upon implementation. After the provided document has been signed and sent back to the server together with its signature, the sent information must be verified on the server's side to prevent unauthorized injection of information in Step 9. Possible solutions could be to create a hash-value of the original document before handing it over to the signature software and to verify that hash-value by comparing it to the hash-value of the received document and which hash-value the signature is based on. Also security tokens need to be checked to verify the information was sent by an authorized user. Finally, after verification of Step 10, the document, its signature, the OCSP response and the certificate-chain of the public signature certificate are gathered and added to an archive-internal Merkle tree, whose root hash is timestamped. All of these documents and the timestamp's root certificate will be stored as a Base64 encoded strings within the database with a reference placed in the *archive* portion of the version's database entry, together with the accompanied meta data described in Section 5.4.2. Before the gathered data is written to the database and the document can be marked as *approved*, the compressed archive container with the structure mentioned in Section 5.4.4 is created and written to a folder which is regularly synchronized to the archiving server. When the writing process has been confirmed with a successful hash-value match of both the created and the written container, the archive is stored in the database together with the files and the new flags to mark the document as *approved* with an *approvedAt* timestamp. As a last step, the internal System Merkle tree is rebuilt

and stored as a new version of the tree in its own table.

5.4 Data Storage

Now that the overview of how the system should behave has been defined, the next step is to define which information should be stored, as well as where and how to store it. This section describes the involved information objects, how documents are represented, which formats are used, how to prepare documents for archiving and what must be done in case of format changes.

5.4.1 Definition of Data and how it is stored

When looking at storage solutions, it is important to define exactly what needs to be stored where, how long to store it and if the information must be archived. As mentioned in Chapter 3, to archive in the context of this thesis means to connect to an archiving server and provide it with the documents that need to be archived.

In the life-cycle of procedure descriptions, there are different stages for each document. Initially, a draft is created by the manager in charge of a service, which needs to get clearance from the data protection commissioner. This draft can be edited, revised and sent to the data protection commissioner to verify and evaluate. When all requirements for approval are met and everything needed is present, the data protection commissioner can approve the procedure description by signing it and thus making it official. At this point, the procedure description turns into its final form and could possibly be added to the procedure description registry or stored securely to ensure integrity and authenticity for later inspections. Only the final form needs to be stored for later inspection, just as only the final printed out and properly signed and approved procedure description is stored in folders and filing cabinets.

The drafts and representative copies of procedure descriptions can be stored in a database for fast and easy access through the developed application. Final procedure descriptions, which have been approved, should be converted into an archivable format and stored with the related meta-information mentioned in sections 5.4.2 and 5.4.4 on a secure and difficult to manipulate medium. This way it is possible to work with documents and make changes, while also ensuring availability and accountability at a later point in time for approved documents.

5.4.2 Document Content

Drafts and final procedure descriptions both store the same base information. As defined earlier, working copies of approved procedure descriptions are kept in the system's database for further viewing, editing and for fast access. This section describes what exactly and how it is stored in these documents. Chapter 2.2.3 lists the information which needs to be stored. When looking at the aforementioned points, it becomes clear that all of the information can, in its most simple form, be stored in text-form while editing and processing. Initially, there are no images, audio- or video-files involved and no special characters outside of Unicode [Theb] are used. When converting and saving these documents there might be more information involved later, including template-files with colour-schemes and images.

Procedure descriptions at the LRZ are separated into three sections:

- "Sections A.1 and A.2 hold internal information for the data protection commissioner. They will not be made public."[6]

[6]Author's translation from an introductory text in existing procedure descriptions of the LRZ: Die Ab-

- "Sections B.1 to B.8 hold the required information according to article 26 paragraph 2 of the BayDSG and represent the procedure description, which can be examined by anyone."[7]
- "Sections C.1 to C.3 hold information on technical and organisational measures according to article 7 and 8 of the BayDSG; they extend the procedure description and are necessary for the granting data protection law clearance by the data protection commissioner. They will not be made public."[8]

All points within a section can be stored in one table, being linked by references into a table holding the three sections.

Figure 5.7 shows a proposed layout for data of procedure descriptions as derived from printed procedure descriptions in use at the LRZ. It shows that the main *ProcedureDescription* document holds two attributes directly relevant to the procedure description, one for the id of the document and one for the reference to the content itself. This might seem wasteful at first, but allows for storage of further meta information as seen and described in other sections. Inside the *Content*-section there are three attributes, one for the short title of the document, one to mark the document as approved by the commissioner and one for the date of approval. In addition, we find the three mentioned sections, *SectionA*, *SectionB* and *SectionC* as references to other tables. *DocumentPurpose*, *DeletionDeadline* and *PermittedUserGroup* are basically referenced key-value tables for each section to hold the different possible points. *SectionA* also holds the contact information of the institution as required by the Bundesdatenschutzgesetz (BDSG). Approved documents are converted for archival and are connected to the procedure description as a referenced archive entry. The archive contains the signed document, the signature, a change-log, possibly needed certificates and the Online Certificate Status Protocol (OCSP) [MAM+] response as Base64 encoded strings in addition to meta-information needed to evaluate the stored files and a timestamp response for the root hash of the Merkle tree [Mer00] containing hashes of all documents. This layout can be implemented in any kind of database and allows for restrictions to viewing of distinct parts.

The content, which is to be stored permanently and archived as soon as possible, includes final procedure descriptions themselves and documents ensuring its integrity and authenticity in the archive section. Now that the type of data and its format is defined, how can the integrity and authenticity of the documents be ensured and prepared for long-term storage?

5.4.3 Long-Term Preservation of Evidential Value

Although archiving as such is not part of the mandatory requirements, the preparation to preserve the evidential value of documents must still be considered. Documents signed and timestamped with the proposed solutions can be verified for at least the time that the certificate with which they were signed and timestamped is valid and the issuing institution is active. Even though accredited providers must ensure an availability of verification information for 30 years after the certificate becomes invalid, the provider does not have to ensure easy access to that information. Additionally, in some cases access to verification data is restricted due to the certificate owner's request or information is unavailable for other reasons. For these scenarios it is useful to store all necessary information for the verification of signatures and timestamps with the documents they are associated to.

---

schnitte A.1 und A.2 enthalten interne Informationen für den Datenschutzbeauftragten. Sie werden nicht veröffentlicht.

[7] Author's translation from an introductory text in existing procedure descriptions of the LRZ: Die Abschnitte B.1 bis B.8 umfassen die erforderlichen Angaben gemäß Art. 26 Abs. 2 BayDSG und stellen die von jedem einsehbare Verfahrensbeschreibung dar.

[8] Author's translation from an introductory text in existing procedure descriptions of the LRZ: Die Abschnitte C.1 bis C.3 umfassen Angaben zu technischen und organisatorischen Maßnahmen gemäß Art. 7 und 8 BayDSG; sie ergänzen die Verfahrensbeschreibung und werden für die Erteilung der datenschutzrechtlichen Freigabe durch den Datenschutzbeauftragten benötigt. Sie werden nicht veröffentlicht.

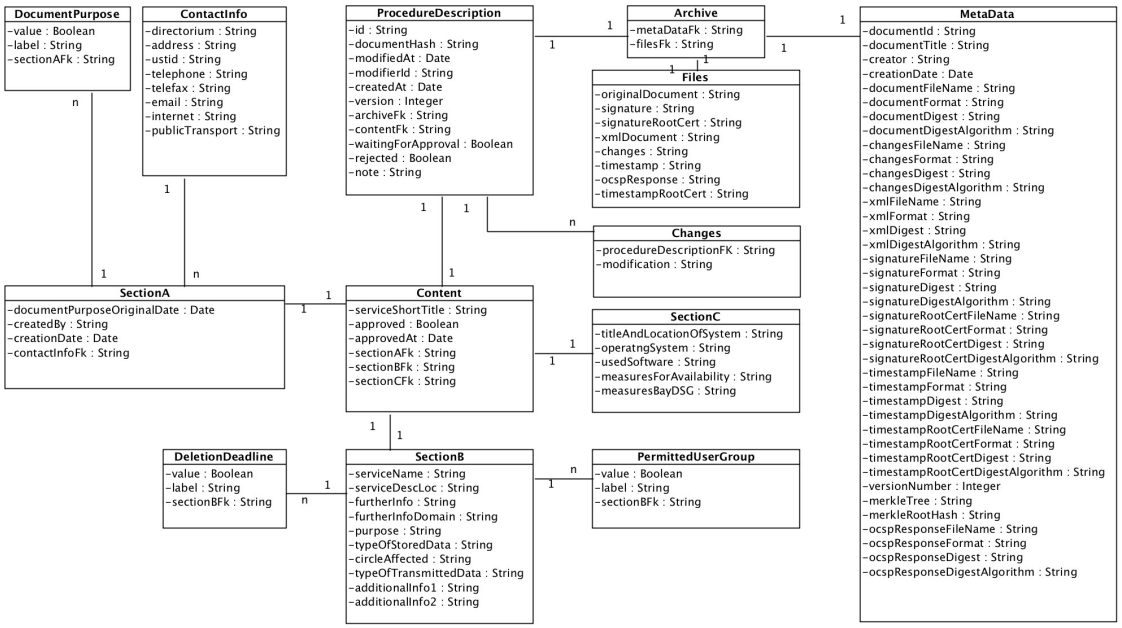

Figure 5.7: Structure for data in the database, derived from procedure descriptions used at the LRZ

To verify a signature successfully after it has been created, five things are necessary:

- original document
- signature
- Online Certificate Status Protocol (OCSP) [MAM+] response
- public certificate corresponding to signature
- certificate-chain

The original document could also be a signed document, in the case that the signature is embedded, as is possible with PDF/A [Ado03] files. In this case the separate signature is not needed. This document is needed to verify what has been signed and that it has not changed since it was signed. With the Online Certificate Status Protocol (OCSP) [MAM+] response it can be verified, that the used certificate to sign the document was valid when the signature was performed. The public certificate related to the signature holds the public key with which to verify the signature and the certificate-chain is needed to verify the relation of the certificate to the issuer and the Federal Network Agency [9] who hands out the issuer's certificates. All of this information should be gathered before archiving the corresponding documents, verified and added to the archive to make later verification easily possible.

The same goes for the archiving of electronic timestamps. To verify a timestamp independently and without outside help, the following files are needed:

- original document or hash-value
- timestamp response

[9] Bundesnetzagentur, translation from http://www.bundesnetzagentur.de/EN/Home/home_node.html

- certificate-chain

Given this information, the timestamp can be verified even after the service it was created with is no longer in operation. To further ensure the preservation of the evidential value, these documents should be timestamped upon receiving them on the archiving server and stored on a storage medium which cannot easily be manipulated, such as a Write Once Read Many (WORM) device. To ensure readability and understandability of all documents inside the archive, the container to store all these files and the metadata file and format must be defined.

5.4.4 Container Format and Layout

All of the previously mentioned files must be stored in a manner which is secure and understandable so they can be successfully evaluated at a later time by third parties. This creates a need for some kind of container and metadata file to put the information gathered into context and ensure the integrity of the file.

To ensure the possibility of long-term storage of these documents, the PDF/A-1 [Ado03] format should be used, since it is recommended by the European Commissions guideline for electronic records management [Eur11], as mentioned in Attachment F of the BSI technical guideline 03125 TR-ESOR [Fedb] and was specifically developed for long-term storage [Ado03].

PDF/A-1 is a standard which is based on PDF version 1.4 [Ado] and was published as the ISO standard 19005-1:2005 [isob]. Requirements to conform with this standard according to ISO 19005-1:2005, include for a basic level to be conclusively, visually, reproducible, to have internal structuring of the content of the document and for an accessible level additionally to allow for mapping of text to Unicode [Theb]. The standard also has some restrictions compared to other PDF standards, e.g. JavaScript and other executable formats cannot be used, device-independent colour-spaces must be used, transparent images are forbidden, encryption of content is not allowed and everything displayed must be embedded. To create the PDF a template is needed to put data into context. This same template should also be used when editing and viewing the information to ensure consistency.

When archiving final procedure descriptions, a container structure based on the open XML standard is used to achieve this. The container's meta data file holds all the information of the document to identify it, e.g. title, version, message-digest, digest-algorithm, timestamp of creation and more. It is needed to archive the procedure description and all other involved documents and perhaps re-sign them or renew their message-digest or digest-algorithm. This container also makes it possible to later verify the document's integrity and authenticity through the attached files. The XML meta data should have the schema described in Listing 5.1 to hold all mentioned files and information.

Listing 5.1 shows the layout of the XML schema for storing the meta information of approved procedure descriptions. The first five elements provide identification of the document with the unique document ID, the title of the document, who created it, when it was created and the version of the document. Next, every file within the archive is listed with its name, format with which it is stored, the associated digest of the document and the algorithm used to create the digest. There is a spot for the actual signed document, a change-tracking file, the xml-representation of the information within the document, the signature of the document, the root-certificates for the signature and timestamp, an OCSP response and the timestamp itself for the root hash of the Merkle tree [Mer00] which connects all files. Lastly, *merkleTree*, represented by an array-string of the included documents holds the structure of the Merkle tree [Mer00] and *merkleTreeRootHash* holds only the top root hash of the Merkle tree [Mer00], to make later signing on the archive-server of all documents at once more efficient. All of these files, together with the meta data file are added to a folder with the name representing the root-hash of the merkle-tree and version-number of the procedure description which is stored. This folder then is compressed into a zip-container and stored within a folder which is named after the document's *documentId* on a volume which gets synchronized to the archiving server. This connection can be seen in Figure 5.3. By storing documents in this structure, versions are connected

through the document's identification number and can easily be read to create a Merkle tree [Mer00] on the archiving server without opening up the containers, since the root-hash of the archive Merkle tree [Mer00] is part of the filename.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="archive">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="documentId" type="xs:string"/>
        <xs:element name="documentTitle" type="xs:string"/>
        <xs:element name="creator" type="xs:string"/>
        <xs:element name="creationDate" type="xs:date"/>
        <xs:element name="versionNumber" type="xs:integer"/>
        <xs:element name="documentFileName" type="xs:string"/>
        <xs:element name="documentFormat" type="xs:string"/>
        <xs:element name="documentDigest" type="xs:string"/>
        <xs:element name="documentDigestAlgorithm" type="xs:string"/>
        <xs:element name="changesFileName" type="xs:string"/>
        <xs:element name="changesFormat" type="xs:string"/>
        <xs:element name="changesDigest" type="xs:string"/>
        <xs:element name="changesDigestAlgorithm" type="xs:string"/>
        <xs:element name="xmlFileName" type="xs:string"/>
        <xs:element name="xmlFormat" type="xs:string"/>
        <xs:element name="xmlDigest" type="xs:string"/>
        <xs:element name="xmlDigestAlgorithm" type="xs:string"/>
        <xs:element name="signatureFileName" type="xs:string"/>
        <xs:element name="signatureFormat" type="xs:string"/>
        <xs:element name="signatureDigest" type="xs:string"/>
        <xs:element name="signatureDigestAlgorithm" type="xs:string"/>
        <xs:element name="signatureRootCertFileName" type="xs:string"/>
        <xs:element name="signatureRootCertFormat" type="xs:string"/>
        <xs:element name="signatureRootCertDigest" type="xs:string"/>
        <xs:element name="signatureRootCertDigestAlgorithm" type="xs:string"/>
        <xs:element name="timestampFileName" type="xs:string"/>
        <xs:element name="timestampFormat" type="xs:string"/>
        <xs:element name="timestampDigest" type="xs:string"/>
        <xs:element name="timestampDigestAlgorithm" type="xs:string"/>
        <xs:element name="timestampRootCertFileName" type="xs:string"/>
        <xs:element name="timestampRootCertFormat" type="xs:string"/>
        <xs:element name="timestampRootCertDigest" type="xs:string"/>
        <xs:element name="timestampRootCertDigestAlgorithm" type="xs:string"/>
        <xs:element name="ocspResponseFileName" type="xs:string"/>
        <xs:element name="ocspResponseFormat" type="xs:string"/>
        <xs:element name="ocspResponseDigest" type="xs:string"/>
        <xs:element name="ocspResponseDigestAlgorithm" type="xs:string"/>
        <xs:element name="merkleTree" type="xs:string"/>
        <xs:element name="merkleTreeRootHash" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Listing 5.1: XML schema structure of the archive metadata file

5.4.5 Procedure Description Format Changes

In the unlikely event that new sections need to be added to procedure descriptions, the content table would need to be modified to include the new section. In addition, the templates which are used for displaying, editing and rendering the procedure descriptions would need to be modified as well, to include the new sections. In case existing procedure descriptions are affected, they would then need to be edited in the front end to hold the new information and then be re-approved by the data protection commissioner. There would need to be checks in the front-end, when rendering and when requesting the information in the system to make sure only existing information is accessed which is needed to ensure system-stability.

The same goes for changes of the format within sections. In this case, the tables for the sections themselves would need to be adjusted together according with template-changes and safety-checks for system-stability within the system when processing information.

It could also be an option to use multiple schemas and accompanying templates, which can be selected when creating new procedure descriptions. If existing documents were to be edited with a different template, there would need to be some form of mapping and change-handling in the background to ensure no data is lost or modified without the user's knowledge.

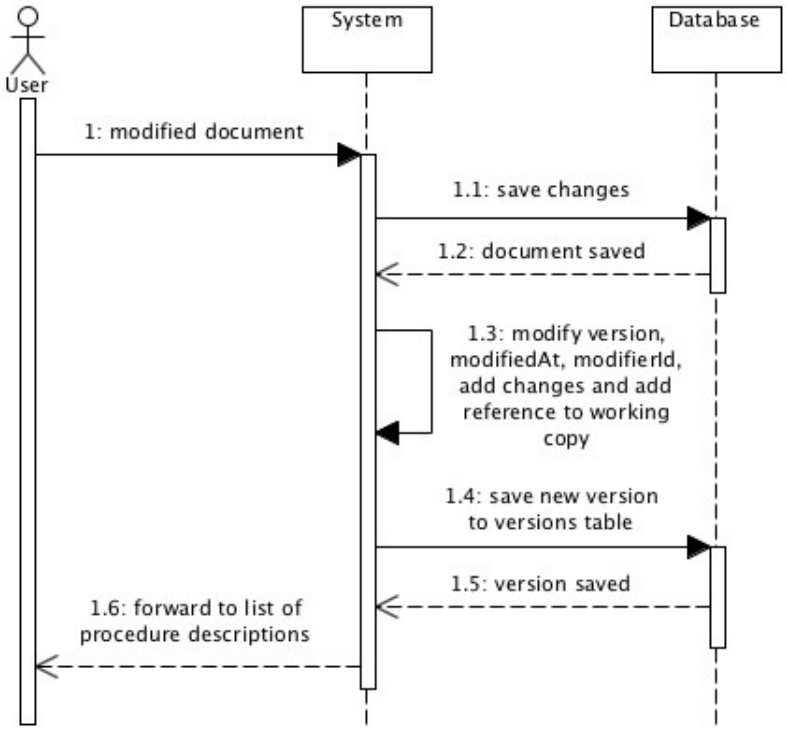

Figure 5.8: Sequence of events when saving an existing procedure description

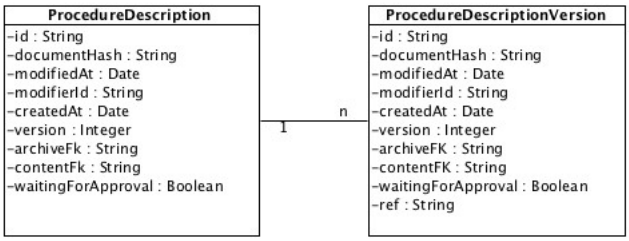

Figure 5.9: Connection between procedure descriptions and their versions

5.5 Versioning

Versioning within the system happens entirely in the used database and in approved archive containers. To achieve this, hooks and methods as seen in Figure 5.8 are used, which run before, after or during actions e.g. when an update happens. Figure 5.8 shows what would happen if an existing document would be modified and then updated within the system. Figure 5.3 demonstrates the connections to these various processes and components at which changes happen. Whenever an action e.g. insert, update or delete is performed on the versioned data, the specified methods are called and fill in necessary information like version information, changes, digest of the current content, creation- or modification-date or the modifier's id. In addition, each version is kept in separate tables with references to the main version. When archiving, a versioned copy is transferred, since the current document gets saved and overwritten any time changes are made or new parts are inserted. The newest version can therefore be regarded as a working copy and the versioned documents as its history up until then. This allows for versioning within the database, without the need to save files externally while still providing a history of all changes and who performed them. Versions cannot be deleted (without direct database access) and every change to the content of procedure descriptions creates a new version.

To demonstrate how this works, Figure 5.9 shows the connection between versions and working copies. The figure shows that both the working copy and the version are the same, apart from the additional attribute called *ref* in the version, which is used to refer to the original or main working copy, which at all times keeps its id. To ensure accountability the procedure description table from Figure 5.7 has dedicated attributes, mainly *modifiedAt*, *modifierId*, *createdAt*, *version* and the referenced *Changes*. The attributes *version*, *modifierId* and *modifiedAt* are updated every time a change is made to the content or the document is deleted. On an insert operation, all key-value pairs of the *content* section are added into the *Changes* table for that procedure description, while only actual changes between versions are added to that table during an update. The *createdAt* attribute only gets updated once when the document is first created. As a *modifierId*, an institution-wide, unique attribute needs to be chosen, by which the modifier can explicitly be identified at a time when needed.

Adding to this, each version has its own entries called *changes*, which hold any modifications done to the actual procedure description within the *content* section of the document. Together with the *modifierId*, the *modifiedAt* timestamp and the *version* number the attributes make up a change. These are grouped and written to a file with the schema seen in Listing 5.2, which can be exported together with approved procedure descriptions. Listing 5.2 shows that there can be multiple *changes* with multiple *modifications* for each procedure description.

As mentioned in Section 5.4.4, saved archives represent a very similar form of versioning when prepared for the archiving server. Each archive-container has a filename representing the Merkle tree root hash of the container together with the internally used version number attached. All versioned

archive-containers are stored in folders representing the original or main document's id within the system, so that versions of documents are grouped together.

The changes to documents are added to the bottom view of a document with its version number, date and modifier's id, which can only be viewed by the data protection commissioner. This makes it possible to switch between versions and see which changes were made.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="changes">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="change">
          <xs:complexType>
            <xs:element name="version" type="xs:integer"/>
            <xs:element name="modifierId" type="xs:string"/>
            <xs:element name="modifiedAt" type="xs:date"/>
            <xs:element name="modifications">
              <xs:complexType>
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="modification" type="xs:string"/>
                </xs:sequence>
              <xs:complexType>
            </xs:element>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Listing 5.2: XML schema structure of the changes file

5.6 Authentication and Permission Handling

Now that document content, layout and how to ensure authenticity and integrity have been defined, it is time to make sure that no unauthorised changes can be made. For this an access strategy is necessary with clear separation of privileges and authentication of each person involved. But what is the ideal way to ensure only those people with the appropriate authorization are able to make changes? It is clear, that there has to be some sort of authentication, since otherwise it is not possible to identify the person that created or edited a document. It would be possible to have the person editing the document just add their name in a separate field, but people cannot be trusted to always enter information truthfully and it would allow for impersonation of others. Pure username and password authentication would solve this problem by giving each user a specific username and a private password, which only that person knows. In this case though, it is not possible to automatically verify a persons identity or to block them from access as soon as they aren't employed at the institution any more. Manual processes by an administrator would be necessary to deactivate the account of the user on the system. A possible solution would be authentication with user certificates, signed by the institution and which can be revoked at any time. This would allow for password-less login and clearly identify the user against the system. The problem with this solution is that the developed system could not be accessed through unprepared devices and the certificate must be installed within the browser to access the service. This is a major drawback, since one colleague for example could not log in to their account on another colleagues computer,

just to make a simple change. They would have to either install their certificate on that computer or they would need to walk back to their computer and access the service from there. Installing the user's certificate on the colleague's computer is not an option, since it would compromise the security of the user's account, while walking back to their own computer is inefficient. It could also possibly allow somebody that finds an unlocked desktop workstation to access the service from that computer without having to be authenticated and would again allow for impersonation of others. Another option would be to log in with a physical token. Although this could allow for portability and effortless logins, simple tokens like USB-sticks with a stored, long password can easily be copied. If easy copying should not be possible, an RSA SecurID [RSA] token could be used, mostly in two-factor authorisation, that generates authentication codes using long keys, called seeds, and the current time for each login attempt [MMPR]. Although this would make the system secure and could allow for better authentication of only one person per account, this solution is quite expensive, since the hardware tokens, seeds and the authentication server must be purchased, which is not feasible for small applications with a limited user-base as is the case with the proposed procedure description management system. To allow for logins from any device within the institution and without making the system unusable, while also making sure that users can be identified correctly, a different approach must be taken. The selected solution for the here developed system is an external authentication service, which gets the user's username together with the password forwarded from the system and verifies the user's identity against its internal user-base. The external authentication service's user base was created by reliable administrators, checking each user's personal information and identity upon account creation. That way each account is directly bound to a specific user and access can be managed centrally. Example solutions for this kind of centralized authentication would be Atlassian Crowd [Atl], OpenID [Fou] or a Lightweight Directory Access Protocol (LDAP) [JS] Server, just to name a few. These solutions provide a database to authenticate against and which applications can use to log in their users.

For the use with the procedure description management software a solution must be picked, which has its authentication server within the institutions control. These solutions are called on-premise and mean that the server running the software is operated within the company's control or their data-centre. In this case the *Authentication Client* of Figure 5.3 can communicate directly with the *Authentication Server*. People not belonging to the institution should not be able to make alterations to the system or be able to administer it. Only authorised personnel can add, delete or modify users after checking their personal information. This way it can be guaranteed, that users are who their credentials make them out to be, as long as their password has not been compromised. In addition, it is possible for administrators of the service to act quick and deactivate accounts in the event of an attack or when a user leaves the company.

To deal with the permissions each user has when working with documents, the system uses roles or so-called permission-groups. In Figure 5.3 the *Permission Manager* symbolically holds this functionality. It has connections to all areas where documents are viewed or modified through user actions and is not specifically connected to all the components to reduce complexity within the diagram.

As described in Chapter 3.1, the system needs to discern between two types of roles when editing documents on the system. The normal user or perhaps manager of a service, who wants to create a procedure description and the data protection commissioner, who can review, edit and approve any procedure description within the system. Additionally, now there needs to be a third role called *admin*, which is allowed to edit rights and roles of other users. Interested parties are left out of the picture since requested procedure descriptions can be made available by the data protection commissioner manually via export upon request. The *admin* cannot perform other or more actions on documents than those of the data protection commissioner. He can also not see documents not belonging to himself and is not allowed to change his own permissions. Instead he can only allow other users to view documents that are not their own and promote or denote the data protection commissioner. The data protection commissioner can and should have the *admin* role in addition to his own role to manage the permissions users have regarding documents. Since the data protection

commissioner by default can view all documents, this does not create a conflict of interest. All of the permissions of the roles can be viewed in Table 5.1. Notable are especially the permission-rules on deletion of own documents. Only the data protection commissioner is allowed to delete documents at all times and owners are only allowed to delete documents as long as these documents are not approved. Any user with an institutional id that uniquely identifies that person on the authentication service can be a normal user with no special access rights and therefore create, view and edit their own procedure descriptions without having access to others documents.

Permission	user	dataProtectionCommissioner	admin
Creation of Documents	X	X	X
Viewing of own Documents	X	X	X
Viewing of Documents of Others	-	X	-
Editing of own Documents	X	X	X
Editing of Documents of Others	-	X	-
Deletion of own unapproved Documents	X	X	X
Deletion of own approved Documents	-	X	-
Deletion of Documents of Others	-	X	-
Approval of Documents	-	X	-
Change Permissions of Others	-	-	X
Change own Permissions	-	-	-
Promote Data Protection Commissioner	-	-	X
Demote Data Protection Commissioner	-	-	X

Table 5.1: Permissions within system

To restrict or grant access to specific documents within the system, the ids of documents are used as roles. When a user creates a new document, the document's id is attached as a role to the user's roles, which gives him read and write permissions. If somebody else is also supposed to have access to the document at a later date, the data protection commissioner can add the id to the new user's roles. The same works when removing read and write permissions to documents from users. The *admin*, *dataProtectionCommissioner* and *documentId* roles are populated and managed with the centralized authentication service. Upon sudden leave or death of the data protection commissioner, it is therefore possible to appoint a new data protection commissioner and to manage access to documents. It is important to keep track of who has which role on the system and when those roles are added or removed to users to ensure accountability. These logs should be managed by the authentication service.

5.7 Securing the System

Securing the system from attacks and malicious behaviour, be it intended or not, should not be an afterthought but part of the conception and realisation of the system. In addition to following best practices when developing the system, there are requirements that should be considered to minimize the impact of potential attacks and misuse of the system. This section describes technical measures

and organizational guidelines to follow before and during development and before taking the system into operation.

5.7.1 System Hardening

First off, it is important to define technical rules and guidelines to protect the system during normal operation or attacks. Securing the system in this way is called system hardening. To secure the database and general system from external attacks, all input which is received should be escaped and validated before further processing it or handing it over to the database. This especially goes for authentication information and data entered into forms, searches or when passing information back to the system from external applications as is done in interaction with the secure signature creation software mentioned in Section 5.3. Since some operations are initiated through the client's browser to perform actions on the server, the access rights to perform these actions should also be validated on the server side. To do this, in addition to id-verification of the user, unique login tokens stored in the browser's local storage should be used, which verify that the user sending the request is logged in and is who he is without constantly logging in again through the authentication service mentioned in Section 5.6. This is especially useful when the requests don't come directly from the user's browser but from a third application such as the secure signature creation software, which as mentioned in Section 5.3 and gets integrated into the system upon implementation.

All communications should also run through secured channels using Transport Layer Security (TLS) encryption to prevent interested third parties from obtaining data which has been sent in plain-text. By using encryption, login-information, authentication-tokens and other confidential data can be protected from unauthorized inspection by malicious third parties on otherwise insecure networks. Another measure which should be followed when implementing the system should be to create hash values of documents and content when passing them between components and verifying their validity upon return. This was also mentioned in Section 5.3 when dealing with the secure signature creation software or when writing documents to the filesystem as described in sections 5.4.4, 5.2 and 5.3. Passwords handled by the system when authenticating users should at no time be stored on the server. If local accounts are used, only salted hash values created with secure hashing algorithms should be stored.

In addition, the components of the system should be secured from tampering. This includes running the system behind a proxy-server redirecting the default port 80 to a secured and TLS encrypted port 443. Not only can transferred data be encrypted and thus protected this way, but a proxy also allows for filtering and mapping routes and input before it reaches the actual application server, adding another layer of security.

To protect the database from unauthorized manipulation, access controls for interaction with it should be implemented. Part of this would be to enable password authentication when interacting with the database, to remove guest-access and to limit the access of the application's database user.

On top of the internal measures for the database, the web-application, the proxy-server and all involved components, a firewall should be configured to only allow access to ports 80 and 443, with the insecure port 80 automatically being redirected to the encrypted port 443 by the mentioned proxy-server. To note when implementing these security measures is, that normal operation of the application should not be influenced by these additions and the hardening of the system.

5.7.2 Organizational Guidelines

Finally, there also need to be organizational guidelines in place to define which actions should be taken in situations of change or failure to ensure accountability and integrity of the system.

The server running the system should be placed in a secure data centre run by the institution with

all components complying with security requirements of the data centre. Access to the underlying system and the archiving server, should be limited, strongly restricted and well documented. All interactions with the underlying system and its attached components as in during updates, exports or upon migration, need to be documented and restricted to only authorized personnel and data confidentiality is to be handled with utmost care. This can be achieved with remote logging of access or in the form of server audit logs that are maintained manually. Further, confidentiality can be achieved with strong encryption before and during transfer and storage to and on other systems. Generally, the web-application should only be reachable from within the institutions internal network and should not be exposed to the entire internet. There is no need for external access since it will be used mostly by managers of services of the institution and the data protection commissioner. This reduces the likelihood of possible attacks in addition to the technical measures taken in the Section 5.7.1. In the case of a compromised password or login information, the account needs to be locked and the password should be reset immediately.

The *admin* role for the system mentioned in Section 5.6 should only be assigned to the data protection commissioner and a limited amount of administrators in charge of the system. Access to procedure descriptions should only be modified by the data protection commissioner upon special request and after evaluating the requesting person's permissions to receive access. In the case of loss of the secure signature creation device such as a chip-card or other forms of compromising of the signature certificate, the certificate should immediately be invalidated by the signature certificate provider and access to the system should be blocked for the data protection commissioner whose certificate is affected until the invalidation process is fulfilled to ensure no documents can be signed with the compromised SSEE. Under no circumstance is it allowed to share secure signature creation devices such as chip-cards since the owner of the signature certificate must be uniquely identifiable at all times to ensure accountability.

Should hash algorithms or parameters used with these algorithms become insecure to use within the life-span of a signed document stored on the archiving server, the affected documents and their accompanying verification information need to be retrieved and their hash values need to be recalculated before signing them with an appropriate and future-proof hashing algorithm and parameters defined in the current "Publication to electronic signatures according to the Signature Law and the Signature Regulation"[10] [Reg]. These newly versioned and signed documents can then be transferred back and stored on the archiving server in the structure described in Section 5.4.4. The system itself and the components involved would also need to be updated in this case to ensure that newly created versions of documents comply with the new guidelines.

5.8 Workflows

Since this system will be a centralized document storage and management solution with different roles interacting, a workflow system, similar to that in real life with physical interaction would be very helpful to increase productivity. As described earlier in Section 5.6, the managers of services create procedure descriptions for their services, which the data protection commissioner can then approve. In the most simple case, the commissioner has to do nothing more than review the document, find out that it meets all standards required and approve it. Anybody who has had to approve documents coming from other sources before knows that this case very rarely applies unless the information which is to be approved is of a very simple nature. The more likely outcome would be that the commissioner finds a small detail or bigger problems of the document to be problematic and needs to reject it. Should the document be rejected, information about why the document was rejected would be useful. This could be accomplished by a post-it note on the document when placing it back on the manager's desk or with a phone call to the manager.

[10] Author's translation of "Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung"

In the here conceptualized document storage and management solution, separate flags and attributes in the container-document which holds the actual procedure description will be introduced to deal with these actions and messages between parties. This includes *waitingForApproval*, *approved* and *rejected* flags in addition to a *note* attribute, which holds a message one editor wants to leave for the next. These can be seen in Figure 5.7

The manager and the data protection commissioner have the possibility while editing the document to press a button to add a note. Each version has its own note, which can be edited as long as the document is not saved yet. If no note is added, no note will be displayed to the next person reviewing the document. If a note was added, the next person to edit the document will be presented with the note in an overlay, which can be removed by clicking an 'x' at one of the edges of the message and can be opened once more when clicking a small icon representing a note at the top of the document. The manager and normal user in addition have a button they can press in case they are satisfied with their changes and want the document approved. In this case, the *waitingForApproval* attribute is set to *true* which the data protection commissioner can filter for in a document list view. The data protection commissioner can then either edit the document, approve it, reject it or just leave a note like the one added by the manager or normal user. If the *reject* button is pressed, a note must be added and the *waitingForApproval* attribute is automatically reset to *false*. When purely editing or approving a document, no note must be added, but can be added. Upon approval, the *approved* attribute is set to *true* and the *waitingForApproval* attribute is set to *false* in addition to adding the current time in the *approvedAt* attribute. The *note* field by default remains set until it is emptied or reset manually by either party.

5.9 Availability of the System

Since the system is supposed to be used to manage procedure descriptions of active services and changes to these services could need to be added quickly, it is important to also ensure availability of documents when needed. This means that documents can be accessed and verified during normal business hours or upon request to the data protection commissioner. In order to be able to quickly restore a broken system regular backups are needed of the underlying database and the application server that has been developed. Since these backups hold confidential information such as passwords, personal information about users and service-specific data, the backups should only be stored in encrypted form and access must be restricted only to recovering administrators in the case of a need to restore the system. All access to the backups should be monitored and logged and backups need to be stored on a secure and managed system.

Aside from the archiving process described in Section 5.2, the data protection commissioner should also have the options to download the PDF document of all documents and the signature and archive of approved procedure descriptions for further processing. This way documents can also be printed out and perhaps archived or reviewed offline if necessary. This can be seen as a form of manual backup functionality.

Additionally the underlying operating system and the relevant applications and services such as the database, proxy server and the application server itself need to be monitored regularly for functionality. One possible way to do this is to send log-entries to a remote logging server and to regularly analyse the logs for system failures. If such a failure is noticed notifications are sent out and the system needs to be fixed or restored from backups.

Another possible approach would be to run both the application server and the database as clusters, allowing for multiple instances to provide the services needed and to take over in case one of the nodes stops responding. Monitoring is still necessary to identify and fix failing nodes, but a failure would not cause downtime in this case, unless all nodes fail simultaneously.

In the most simple scenario the operating system itself with its init system can be used to restart

hanging applications. *systemd* for example should be configured to always restart a stale or hanging system. This is a first measure to ensure availability. Listing 5.3 shows how the service must be configured for this automatic restart to work.

```
[Service]
...
Restart=always # restarts the service in case of failure
...
```

Listing 5.3: systemd Service Configuration for better Uptime

Basic and fast responses to system failures can be managed with new tool approaches such as a combination out of *Graphite* [Dav] graphed monitoring with *Cabot* [Ara] alerting and either *StackStorm* [Sta] or *Hubot* [Git] for taking direct and automatic actions such as cleaning the system or restoring from backup on another system. Virtual machines and containers could be used for short spin-up times and fast responses to failures. The systems and access controls would have to be carefully configured and defined to not leak information or to cause bigger problems during restoration. But if such an approach is cleanly implemented, almost anything could happen to the monitored system and an alternative backup system would be up in minutes to catch any requests coming in.

6 Implementation

This chapter describes the implementation of the concept of chapter 5 with putting additional emphasis on:

- Used Language and Framework
- Document Processing and Storage
- Versioning
- PDF Generation
- Integrity and Authenticity
- Implementation of Electronic Timestamping
- Approval Process with Qualified Electronic Signatures
- Archiving Documents
- Securing of System

6.1 Language and Frameworks

The prototype implementation of the proposed concept of chapter 5 is based on the *Meteor* [Meta] platform, which allows for full stack JavaScript[1] application development. It uses a secured MongoDB [Monb] database to store information and various *Meteor* and *Node.js* [Joy] packages which will be described in more detail when needed. The *Meteor* platform and the web-server it uses run on *Node.js*, which allows for flexible development and fast prototyping. The community around *Node.js* and the package manager *npm* [npm] have grown extensively since their release in 2009. Applications written in JavaScript allow others to quickly understand the source code that has been written, while being compatible with all standard operating systems e.g. Windows, Linux and Mac OS X. *Meteor.js* adds a framework to *Node.js*, which increases the speed of development and provides tools to build, test and deploy applications. It reduces the need to combine multiple programming or scripting languages like PHP, Java, Perl and JavaScript, which makes it easier to keep track of which parts interact with others.

Through the use of the model-view-controller pattern, templating and its ability to act asynchronously out of the box, building applications is straight-forward and allows the developer to concentrate on implementing features instead of the basic structures and functionalities. Even parts of the security aspect are taken care of with *Meteor* out of the box. By default, source code is separated into client and server code with rights management for database access built in. All user input is HTML escaped through the way the templating system works, securing the application against many Cross-Site-Scripting (XSS) attacks. The standard database used by *Meteor* is MongoDB, which by itself is known to be more resistant to database injections than other database solutions [Mona]. *Meteor* also uses a publish-subscribe model to only allow access to specific data by the client. In addition, it uses something called *Meteor Methods* to call and execute code on the server, where it must be verified first and can't be tampered with.

[1] Scripting language based on ECMAScript [ecm]

Meteor has matured a lot within the seven years since its release and has been used in some production environments for a while now. Examples include the research sharing application LIQUID [Liq], the group Twitter-response application respondly [Res] or the website CodersTV [Pug], which allows users to watch videos about programming in various languages.

For faster creation of templates and to set up the initial project structure, the *iron* command line interface (CLI) [Iroa] was used for scaffolding as described in an article by Eschweiler on scotch.io [Esc15]. The *Bootstrap* [OT] framework is used for the layout together with icons from the *Font Awesome* toolkit [Gan] and the *Collection2* [Dobc] and *AutoForm* [Dobb] *Meteor* packages as suggested in Eschweiler's tutorial [Esc15]. Overall the components are ideal for rapid prototyping of the conceptual work described in Chapter 5.

6.2 Architecture

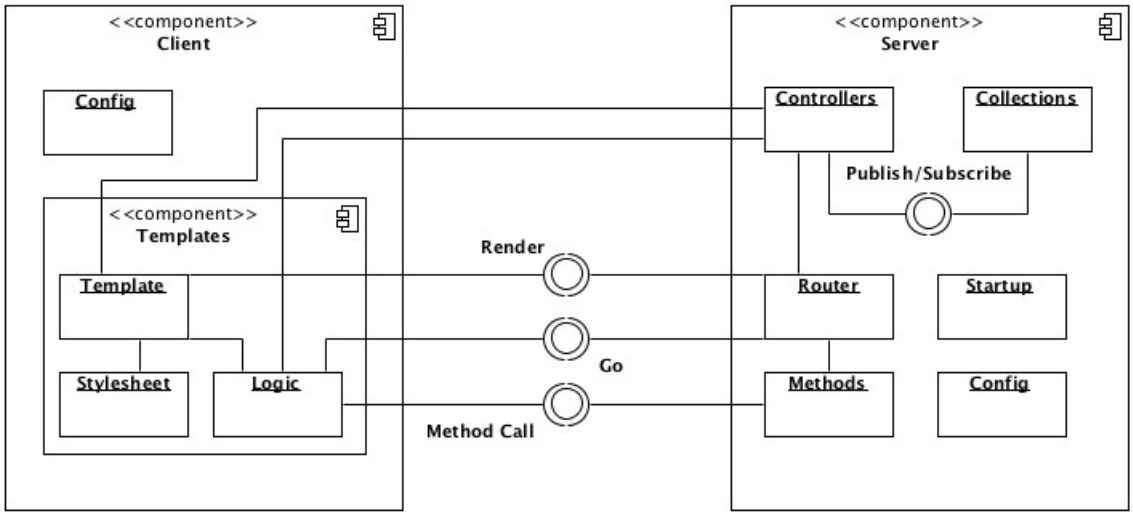

Figure 6.1: Structural Overview

As mentioned in Section 6.1, *Meteor* separates server and client scopes to protect sensitive data and algorithms from inspection through the client. This and the following points can be seen in Figure 6.1 showing the overall structure of the project. The used *Meteor Iron.Router* package [Irob] enables the application to display multiple pages with access restriction. Pages are organized in folders within the *client->templates* scope, holding one template file with the ending *.html*, one style file with the ending *.css* and one logic file with helpers and other methods ending in *.js*. Each file holds the lower case name of the template, which makes it easier to see a connection between the files. There are *config.js* files for each the server and the client scope, configuring each part. Within the server scope there is also a *startup.js* file, which initializes components like error messages, directory mappings, languages or internal collections. *Meteor* methods can be called from anywhere within the system, but are mostly used to securely execute sensitive operations requested by the client on the server. Controllers subscribe to published collections, making the collections and other specifically defined data available through the logic-handlers of templates.

6.3 Data Handling

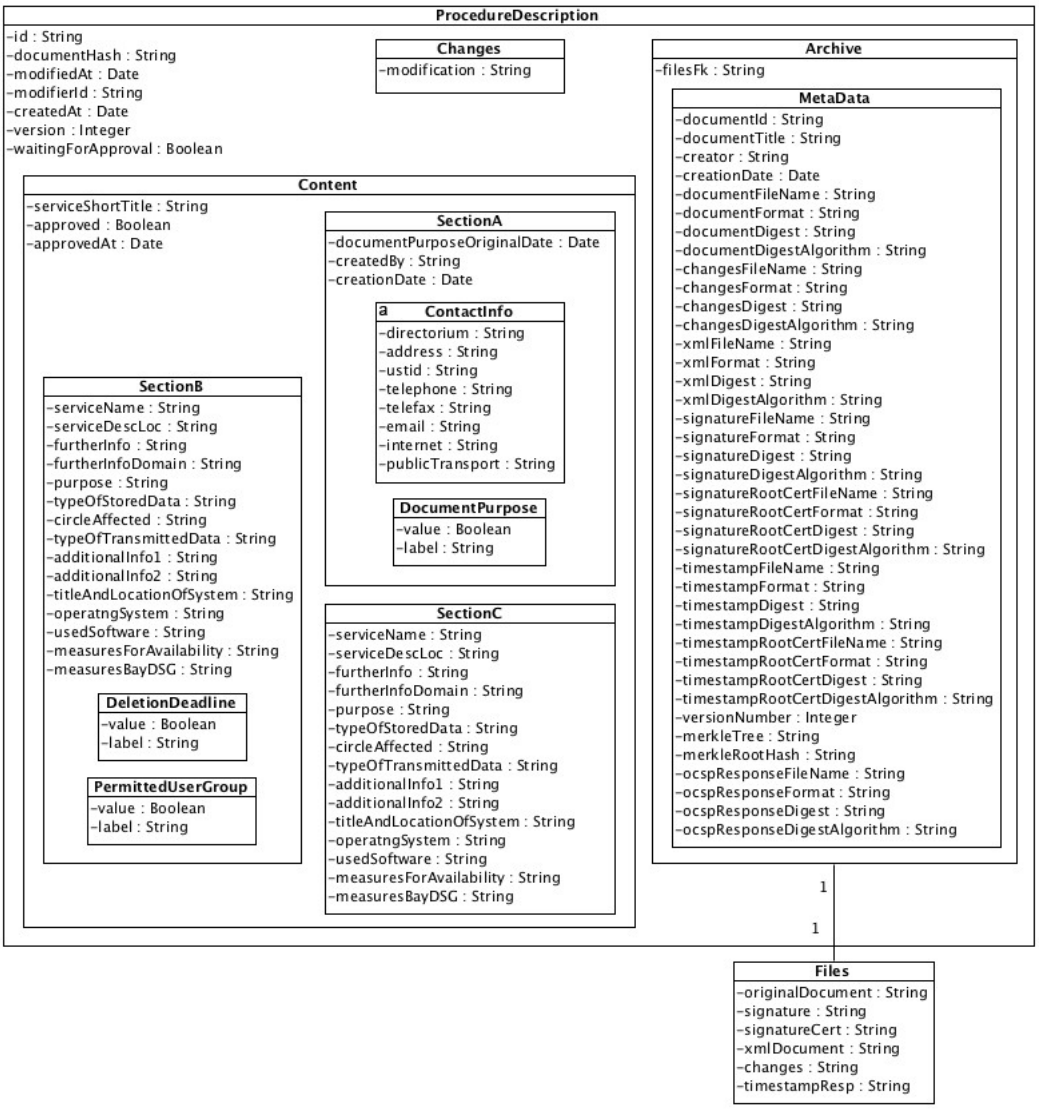

Figure 6.2: Procedure Descriptions in MongoDB

Procedure descriptions are stored in collections within the MongoDB [Monb] database. To ensure correct structure and type-safety, the *Meteor Simple-Schema* package [Doba] is used. It makes the creation of clearly defined tuples possible, which can be validated and dynamically populated. This package is best used with the *Meteor Collection2* package [Dobc], which specifically allows the schemas to be added to MongoDB database collections. Instead of using multiple tables, tuples of schematically structured data are saved within one document. The base64 encoded representations of files are separated out into their own collection to increase performance. Upon creation of a document, the data from within the template is automatically cleaned and validated using the collection's schema and sub-schemas and saved within the *content* section of the document. Before inserting and after updating a document, a hash-value for the *content* section is calculated and saved in the document-

container holding the *content* section. It can later be used to verify that no direct, un-versioned changes have been made to the *content* section. The document hash-value could also be stored on an external server and be requested every time it is needed to add more security. The *aldeed:autoform* [Dobb] package is used to easily access the attributes of collection elements, create new documents, edit these and update the database automatically through secured *Meteor* method calls. Both the working copies and approved documents are stored within the same collection since they are of the same type with applied access restrictions as mentioned earlier in Section 5.6. Searching and sorting is possible via parameters which are stored in session variables and retrieved by helper methods within the template. They are passed to the collection as filtering and sorting parameters. Since *Meteor* by default is reactive, all search and sorting operations initiated through the helper methods have an instant impact on the displayed data.

6.4 Versioning

Inside the system there are two types of versioning. First, all working copies of documents are versioned within the database. Secondly, archive-objects of approved procedure descriptions are versioned when processing them after approval.

The *Meteor Collection Hooks* package [Bou] adds the functionality of database hooks, which are specifically used by the *Meteor Vermongo* [FM] package to accomplish versioning in the system. This package creates a second collection for every activated collection with versions of documents in the MongoDB database. Versions of documents can be accessed from within documents through the helper-method *versions*, which is present in each collection object.

Whenever a document is created, edited or deleted, a new version is created in the versions-collection, with a reference to its working copy. The working copy never changes its id, while each version gets its own id within the versions collection. Before creating the new version in the collection, the modifier's user id, the modification date, the changeset and the version number are updated for the document. In case the document is in its first version and no other version exists, additionally the creation date is added. After these changes have been made the version is sent to the versions collection.

```
// Array.prototype.diff = function(a) {
//   return this.filter(function(i) {
//     return a.indexOf(i) < 0;
//   });
// };
...
// if(fieldNames.diff(options.ignoredFields).equals([])) return;
...
```

Listing 6.1: Vermongo.js modifications to fix incompatibility

```
...
collection.before.insert(function(userId, doc) {
  ...
  if (!doc[options.userId] && options.userId && userId) {
    var user = Meteor.users.findOne(userId);
    doc[options.userId] = user.username;
    doc["changes"] = {'created': 1, 'added': doc.content};
  }

  copyDoc(doc);
});

collection.before.update(function(userId, doc, fieldNames,
        modifier, hook_options) {
  // do nothing if special option is set
  if(offOnce) {
    offOnce = false;
    return;
  }
  // do nothing if only ignored fields are modified
  // if(fieldNames.diff(options.ignoredFields).equals([])) return;

  modifier.$set = modifier.$set || {};
  var originalSet = _.extend({}, modifier.$set);

  // in case of doc not already versionned
  if (!doc._version) doc._version = 1;

  // copyDoc(doc);
  ...
  if(doc[options.userId] && options.userId && userId) {
    var user = Meteor.users.findOne(userId);
    modifier.$set[options.userId] = user.username;

    var changes = new Array();
    for (var key in originalSet) {
      var value = doc[key];
      if (key.indexOf('.') > -1) {
        var subkeys = key.split('.');
        var value = doc;
        for (var i=0; i<subkeys.length; i=i+1) {
           value = value[subkeys[i]];
        }
      }
      if (originalSet[key] != value) {
        changes.push(key + ": " + originalSet[key]);
      }
    }

    modifier.$set.changes = {'modified': 1, 'modifications': changes};
  }
});

collection.after.update(function(userId, doc, fieldNames,
        modifier, hook_options) {
  // do nothing if special option is set
  if(offOnce) {
```

```
    offOnce = false;
    return;
  }

  // in case of doc not already versionned
  if (!doc._version) doc._version = 1;

  copyDoc(doc);
});

collection.before.remove(function(userId, doc) {
  ...
  if(doc[options.userId] && options.userId && userId) {
    var user = Meteor.users.findOne(userId);
    doc[options.userId] = user.username;

    doc["changes"] = {'deleted': 1};
  }

  doc._deleted = true;
  copyDoc(doc);
});

collection.helpers({
  versions: function() {
    return _versions_collection.find({ref: this._id},
      {sort: {_version: -1}});
  },
  getChanges: function() {
    var changes = new Array();
    versions = _versions_collection.find({ref: this._id},
      {sort: {_version: 1}}).fetch();
    for (var version in versions) {
      changes.push({
        'author': versions[version][options.userId],
        'changeset': versions[version].changes,
        'version': versions[version]._version,
        'modifiedAt': versions[version].modifiedAt
      });
    }
    return changes;
  }
});
...
```

Listing 6.2: Vermongo.js versioning modifications

To get the package to work with other packages used in the system, its own implementation of an array difference finder, which collides with another implementation, must be deactivated. The functionality of ignoring fields when inserting or updating was not used within the project so there was no negative impact on how the package works. Changes to added *Meteor* packages can only be made if they are manually added to the *packages* directory of the project. Listing 6.1 shows the function in the *vermongo.js* file of the package that needs to be commented out. Additionally, any mention of the method must be commented out as well in the methods *collection.before.insert(...)* and *collection.before.update(...)*. By default the package also only creates the first version of a document in the versions collection after it has been updated. In order to fix this, the *copyDoc(..)* method was added to the end of the *collection.before.insert(..)* method, removed from the *collec-*

tion.before.update(...) method, added in a new *collection.after.update(...)* method and changed in the *collection.before.remove(...)* method. Additionally, a changeset for each version was introduced, storing differences between versions in each new version as an attribute called *changes*, containing an array of strings with *key: value* pairs. When exporting the changes during the archival process as described in Section 5.5, groups for version changes are added to an array of changes with the version's author, changeset, version number and modification date. Listing 6.2 shows the mentioned changes.

If a document is deleted, a flag is added to the document and is saved with the working copy's id and its own id in the *ref* field in the versions-collection. Versions cannot be deleted from the application. Only working copies can be deleted, which also removes versions of that working copy from views. All versions of that document remain in the database and on the archiving server if an archiving server is in use. When a user modifies an approved procedure description, the approval flag is removed from the working copy of the document while leaving the approved version untouched. The approved document can still be accessed and is still listed with all other approved procedure descriptions. The hook to achieve this automatic modification is added directly beside the collection creation logic, resetting the approval flag whenever changes to the *content* area are made without the approval flag being part of the modifier's set.

6.5 PDF Generation

Documents that have been created and edited within the system need the possibility to be exported, printed or saved for further processing. Also the approval process with qualified electronic signatures needs a file which can be signed and later stored. In Chapter 5.4.2 PDF/A-1 was selected as the document format which should be used to export and store the procedure descriptions. PDF files are generated on the server side in a *Meteor* method after verifying the request's validity and the requesting user's rights. The generated file itself is not saved upon creation and is generated new every time, unless it is already present within the database after an approval process in which it was saved.

There are different libraries to create PDF files with *Meteor*. For the purpose of this prototype implementation of the system, an approach using server-side rendering with the *npm* [npm] package *webshot* [Kok] was pursued, which produces screenshots of web-pages using Apple Inc.'s *WebKit* [App] browser engine. The implementation follows a tutorial created by Swapp [Swa] to pass a template with helper-methods to a server-side rendering engine, which builds an html string to be used with the *webshot* API. The *webshot* API can then be used to create a temporary file on the server, which is read and sent back to the calling function to be converted to Base64 for transfer.

To be able to use the *webshot* API, which is a wrapper for the underlying PhantomJS [Hidb] scripted browser, all operating system package dependencies of the browser must be present on the system. Different from most other *npm* [npm] packages, installing the *webshot* package and letting it gather its dependencies is not enough, since dependencies of the *PhantomJS* package are not pulled automatically. The resulting error messages are not self-explanatory and took a lot of time during the development process to resolve. The dependencies needed to get *PhantomJS* running can be found in the build-section of the developer's website [Hida].

Since all images and used resources must be part of the created PDF file and cannot lie externally, the used LRZ logo at the top of the document was encoded in Base64 and added to the template. There is no way to load the image otherwise, without hard-coding an external address, at which the image must be publicly accessible. The files generated with the *webshot* approach can be reproducibly viewed on all systems. However, they are not fully compliant with the PDF/A-1 standard, since there was no way to add meta data through the used *webshot* API or the underlying *PhantomJS* scripted browser. In addition, the Base64 encoded image of the LRZ logo makes it necessary to

mention in a comment in the file, that the PDF contains parts encoded in 8-bit binary data. Since browsers by default use the RGB colour space with the html-annotation for colours in CSS styling files, complete device-independence is not fulfilled, even though no colours were explicitly specified within the template and no colours besides black are used. Other than that, the generated files are compliant with the PDF/A-1 standard. To allow for law-compliant archiving, the creation-process should be improved to be fully compliant with PDF/A-1 in later versions of the system.

6.6 Approval of Documents

In order to be able to approve documents, it is necessary to create qualified electronic signatures, for which first, a provider of certificates and secure electronic signature creation units must be selected. This section first selects a certificate provider, secure signature creation unit and signing software before describing how documents are approved and therefore signed within the developed system.

6.6.1 Signature Certificate and Card Reader

As described in Chapter 4.2 there are only three viable suppliers of qualified electronic signing certificates to choose from. The by far most economic long-term solution is the option to sign documents using the new German personal identification card and the signature certificate which is issued through the D-Trust GmbH. All that is needed is the new personal identification card with activated online-identification functionality, the signing certificate and the *REINERSCT cyberJack® RFID komfort* card reader. The *REINERSCT cyberJack® RFID komfort* card reader is mentioned in this combination, since it is the only possible option when creating signatures with the new German personal identification card. The creation unit can be seen in Figure 6.3. This is also the only card reader which can be used with the sign-me service [Bunc] of the Bundesdruckerei GmbH, since it alone has an integrated display and secured pin-pad, making it a safety class 3 card reader in addition to providing the ability to read radio-frequency identification (RFID) cards, chip-cards and sign documents.

Figure 6.3: REINERSCT cyberJack® RFID komfort card reader

For testing purposes, the aforementioned D-Trust GmbH certificate for the new German personal identification card was selected together with the mentioned card reader. The developed document storage solution works with any of the in Chapter 4.2 mentioned providers and id-cards as long as the accompanying software and drivers are installed on the client-computer. As long as no attribute certificates or other special features are needed, the national personal identification card with signature feature could also be used to create signatures at the institution with the developed software. In case attribute certificates are needed or there is a need for corporate support, the certificates of the Deutsche Telekom AG are recommended as they are the second most economic solution, costing 99 Euros for two years and allowing the use of class 2 card readers like the *Cherry Smart Terminal*

ST-2000 UCZ. The Deutsche Telekom AG is also the provider with the longest record of service, being accredited since December of 1998 and they therefore have the most experience with providing high-quality solutions [Wula].

To get a certificate, it must be ordered through the portal of the provider, entering payment details and personal or institutional information of the requesting party. In the case of the used D-Trust GmbH certificate this can be done through the third-party reseller *REINER SCT* [REI]. The certificate can then be loaded onto the id-card through the sign-me service of the Bundesdruckerei GmbH [Bunc] by using an access code that is sent by mail to the home-address specified during the ordering process. During the activation process, a signature pin must be defined, which is requested every time a signature is created. This is a safety precaution which is supposed to prevent unauthorized signing with a stolen or lost id-card. Should the id-card be stolen or lost, the certificate should immediately be blocked through the provider's portal. After activating and loading the certificate onto the id-card it can be used with the provided software-package *AusweisApp 2* [Gov] and the software solutions mentioned in the following section.

6.6.2 Signature Creation Software

Figure 6.4: SecCommerce SecSigner signature creation software during signing process

Since all software products for signature creation mentioned in Chapter 4.3, aside from the *SecCommerce SecSigner*, require license fees, the free integration of the *SecSigner* Java applet was chosen. All other current solutions for signature creation through web-services also use Java applets and the Netscape Plugin Application Programming Interface (NPAPI) for applet loading, which makes the *SecSigner* solution very attractive. Future solutions such as the previously mentioned *Sign Live! CC cloud suite bridge* from intarsys [inta] should be evaluated when they become available and affordable. The applet is implemented according to the standards and regulations of the German Signature Law [siga] and Signature Regulation [sigb]. It provides a secure possibility to interact with locally attached hardware like smart-card readers to create signatures and has a self-check mechanism built in to protect it from being tampered with. To use the applet, the *AusweisApp 2* application on the client computer should be configured to automatically launch when the system starts. It is needed to create a bridge to the card reader and national identification card.

The applet can either be integrated into an existing web application or can be activated through an HTTP page. Needed parameters are passed to the applet within the *APPLET*-HTML tags (see Listing lst:secSignerConfig).

```
<APPLET id="applet"
        codebase="{{getUrl}}applet"
        code="seccommerce.secsigner.applet.SecSignerApplet"
        archive="SecSigner.jar,SecSignerExt.jar"
        width="790"
        height="496" MAYSCRIPT>

        <!-- do not send user-certificate to PostURL -->
        <PARAM NAME="Init" VALUE="off">
        <!-- Base64 encoded PDF -->
        <PARAM NAME="DocumentBase64" VALUE="{{getBase64Pdf}}">
        <!-- document is in PDF Format -->
        <PARAM NAME="DocumentType" VALUE="4">
        <!-- signature type is PKCS7 -->
        <PARAM NAME="SignatureFormatType" VALUE="0">
        <!-- create hash with SHA256 algorithm -->
        <PARAM NAME="HashAlg" VALUE="SHA256">
        <!-- send data instead of only signing -->
        <PARAM NAME="PostSignedData" VALUE="on">
        <!-- URL to POST values to after finishing -->
        <PARAM NAME="PostURL" VALUE="{{getUrl}}receive">
        <!-- post parameters to send to PostURL -->
        <PARAM NAME="PostParams" VALUE=
        "userId,userToken,documentId,version,serviceShortTitle,createdBy">
        <!-- post parameter value userId -->
        <PARAM NAME="userId" VALUE="{{currentUser._id}}">
        <!-- post parameter value userToken -->
        <PARAM NAME="userToken" VALUE="{{getUserToken}}">
        <!-- post parameter value documentId -->
        <PARAM NAME="documentId" VALUE="{{_id}}">
        <!-- post parameter value version -->
        <PARAM NAME="version" VALUE="{{_version}}">
        <!-- post parameter value serviceShortTitle -->
        <PARAM NAME="serviceShortTitle" VALUE="{{content.serviceShortTitle}}">
        <!-- post parameter value createdBy -->
        <PARAM NAME="createdBy" VALUE="{{content.sectionA.createdBy}}">
        <!-- URL to redirect to after success -->
        <PARAM NAME="FinishURL" VALUE="/proc_desc_list">
        <!-- load FinishURL in same window -->
        <PARAM NAME="FinishURLTarget" VALUE="">
        <PARAM NAME="CancelURL" VALUE="/proc_desc_list">
        <PARAM NAME="CancelURLTarget" VALUE="">
        <PARAM NAME="ErrorURL" VALUE="/proc_desc_list">
        <PARAM NAME="ErrorURLTarget" VALUE="">

</APPLET>
```

Listing 6.3: SecCommerce SecSigner Java applet configuration

The *codebase* is the public location at which the JAR files can be found and *archive* lists the files holding the applet during loading of the page. The parameter *code* in the *APPLET* tag defines the main-class to load when starting. As Listing 6.3 shows the user's token is passed to the Java applet to prevent just anybody from posting to the mentioned *PostURL*. This token is then checked on the server side to verify that an authorized person is sending the information. Generally, only

the data protection commissioner is allowed to post to the *PostURL*. All parameters have comments explaining their purpose.

Additionally, a properties file which is included with the *SecSigner*, must be configured before using the applet. The configuration for the use with the developed storage solution is modified in the following way:

```
# search for smartcard automatically
seccommerce.secsigner.smartcardautosearch=on
# close applet after finishing
seccommerce.secsigner.autoclose=on
# always show smartcard and reader information
seccommerce.secsigner.autoswitchtosigndlg=off
# do not allow additional signatures after verification
seccommerce.secsigner.allowsignafterverify=off
# embed applet in browser instead of in pop-up
seccommerce.secsigner.displayinbrowser=on
# hide button to display document in external reader
seccommerce.secsigner.showdisplaydocbutton=off
# ocsp checks are mandatory
seccommerce.secsigner.ocspmandatory=on
# do not present the option to create a timestamp
seccommerce.timestamp=off
# do not support readers with insecure pin entry
seccommerce.supportreaderpcscwithoutsecurepin=off
# do not show the selection menu on the left when signing
seccommerce.secsigner.disableselectwhensigning=on
# require qualified certificates
seccommerce.cert.requirequalified=on
# do not present option to store document on computer
seccommerce.secsigner.savedocument=off
```

Listing 6.4: SecCommerce SecSigner Java applet modified properties

Listing 6.4 shows the property-values which must be changed and added. All other values remain in the file as they are by default when first downloading the *SecSigner* package from the supplier's website.

6.6.3 Integration in the System

Now that the certificate, signature creation device, card reader and signature creation application have been selected and configured, the signature creation application needs to be integrated into the document storage solution. For this, the applet tags mentioned in Listing 6.3 are added to an approval template and the helper methods are implemented to gather the needed data in *{{code}}* brackets. The applet files must be accessible in a public location on the server to be loaded by the web-page and sent to the client. All together five files are needed: *SecSigner.jar, SecSignerExt.jar, SecSignerSelfcheckSig.jar, SecSigner.selfchecksig and secsigner.properties*. *secsigner.properties* holds the default properties together with the modifications mentioned in Listing 6.4. *SecSigner.selfchecksig* and *SecSignerSelfcheckSig.jar* are needed so that the application can independently verify that it has not been tampered with by checking its own integrity. *SecSigner.jar* and *SecSignerExt.jar* hold the actual application with all its functionality. These files are placed inside a folder called *applet* in the public folder of the project, which is directly exposed to the client.

To make the template available within the application it also needs to be added to the *Iron.Router* [Irob] *route.js* file in the form of a route as seen in Listing 6.5. The route defines a route, the name for the route, the responsible controller, the action to perform on routing, where the code is to be

executed, what to wait on and what to do before executing the action. At the end of the route, the string *:_id* allows for a parameter to be passed. In this case it is the id of the document that is to be approved. Since a document will be loaded from the Procedure Description collection, the *waitOn* function must return the current status of the subscription to the collection. When the collection is ready, the action will be executed. But before the action can be executed, the *onBeforeAction* is run to validate that the user actually has the permission to approve a document. The checking of permissions is done with the help of the *meteor-roles* package [Lan], by checking if the currently logged in user has the role of *'datenschutzBeauftragter'*. The just mentioned *'datenschutzBeauftragter'* role represents the *'dataProtectionCommissioner'* found in Section 5.6. If not, the user is redirected to the *home* route and otherwise is allowed to proceed by executing the route's action.

The action of the route is a string reference to the function to be called within the defined controller. In addition to providing the data of the document and just rendering the template, the controller must also first generate the PDF file and ensure its availability before the template can load. This is the second of two approaches to ensure availability of the PDF file upon approval. The first is to create a PDF file and store it every time a new version of a document is created. This would mean even drafts which might not be used or were saved by accident would take up space inside the database or on the storage volume, which makes the solution inefficient and unnecessary. Therefore, the PDF file is created and stored only when it is needed. During tests, the waiting time to do this was acceptable on the used server under no load with an average of one second spent on the loading screen. In addition to defining where the PDF file comes from, it is also necessary to select the correct location and time within the application to load the information. If helper methods within the template would be used, the applet would be executed before the PDF file was passed to it, resulting in an error message. Because of this, the generation needs to take place and the information needs to be available before the template is rendered.

```
Router.route('/approve_proc_desc/:_id', {
  name: 'approveProcDesc',
  controller: 'ApproveController',
  waitOn: function() {
    return Meteor.subscribe('proc_descs');
  },
  onBeforeAction: function() {
    var userId = Meteor.user()._id;
    if (!Roles.userIsInRole(userId, ['datenschutzBeauftragter'])) {
      this.redirect('home');
    } else {
       this.next();
    }
  },
  action: 'approve',
  where: 'client'
});
```

Listing 6.5: Iron.Router route for approving a document

Inside the *ApproveController* the action '*approve*' represents the function, which creates the mentioned Base64 encoded PDF file in a *Meteor* method using the *content* section of the document to then store it in a reactive dictionary before rendering the template. Listing 6.6 shows the implemented solution. As proposed in Section 5.7.1, the called *Meteor* method does not just take the method name, callback function and document id as parameters, but also the user's id and login token so the user's permission to perform this action can be verified on the server side. To be able to use the *ReactiveDict* constructor, the *reactive-dict Meteor* package [Metb] must be added to the project manually by running *meteor add reactive-dict* in the project folder even though it is included by default. While the PDF is created and sent back through a callback function, a loading screen is shown, which in Meteor is configured in the basic configuration of the *Iron.Router* by adding a value

for the parameter *loadingTemplate*.

```
ApproveController = RouteController.extend({

  ...
  approve: function() {
    var that = this;
    Meteor.call('proc_desc_pdf', Meteor.userId(),
        localStorage.getItem("Meteor.loginToken"),
        this.params._id, function(err, res) {

      if (res) {
        if (!that.states) {
          that.states = new ReactiveDict(null);
        }
        that.states.set('pdfData', res);
        that.render('ApproveProcDesc', {});
      }

    });
  }

});
```

Listing 6.6: approve action in ApproveController

After the signature has been created and confirmed using the signature creation software, the POST parameters listed in 6.3 are sent to the *PostURL* where they need to be received, verified and further processed by the web-application. This was achieved with a special route */receive* which gets handled by the *ReceiveController* in that it directly works with the *request* object received from the *Iron.Router*. There were some issues with sending and receiving larger amounts of data through routes, which is why *Iron.Router* had to be reconfigured to also allow larger files as seen in Listing 6.7.

To make things clearer, Figure 6.5 shows the steps involved when processing the information that was received. First, the involved user and document objects are retrieved from the database to verify permissions and the digest value. Next, changes for the archive are generated and updated with the approval changes that will be made before creating and retrieving all documents and digests needed to verify and archive the document and its signature. The actual XML representation of the metadata and changes files are created using the *Node.js simplexml* [Man]. From the generated and retrieved files the archive Merkle tree can be built and its root hash timestamped before building the *archive_files* and *archive_metaData* objects. These are added to a modifier set object together with the approval flag, its time of approval and the changeset generated for the version-change. Before updating the procedure description document in the database, the compressed archive is created according to the guidelines of Section 5.4.4 and written to the file system and verified upon creation. Finally, the document is updated as *approved* in the database with the newly attached archive object containing the files and metadata before also updating the system's Merkle tree of approved procedure description versions (see Section 6.7).

```
Router.configureBodyParsers = function() {
  Router.onBeforeAction(Iron.Router.bodyParser.urlencoded({
    extended: true,
    limit: '20mb'
  }));
};
```

Listing 6.7: Iron.Router Configuration for larger File Handling

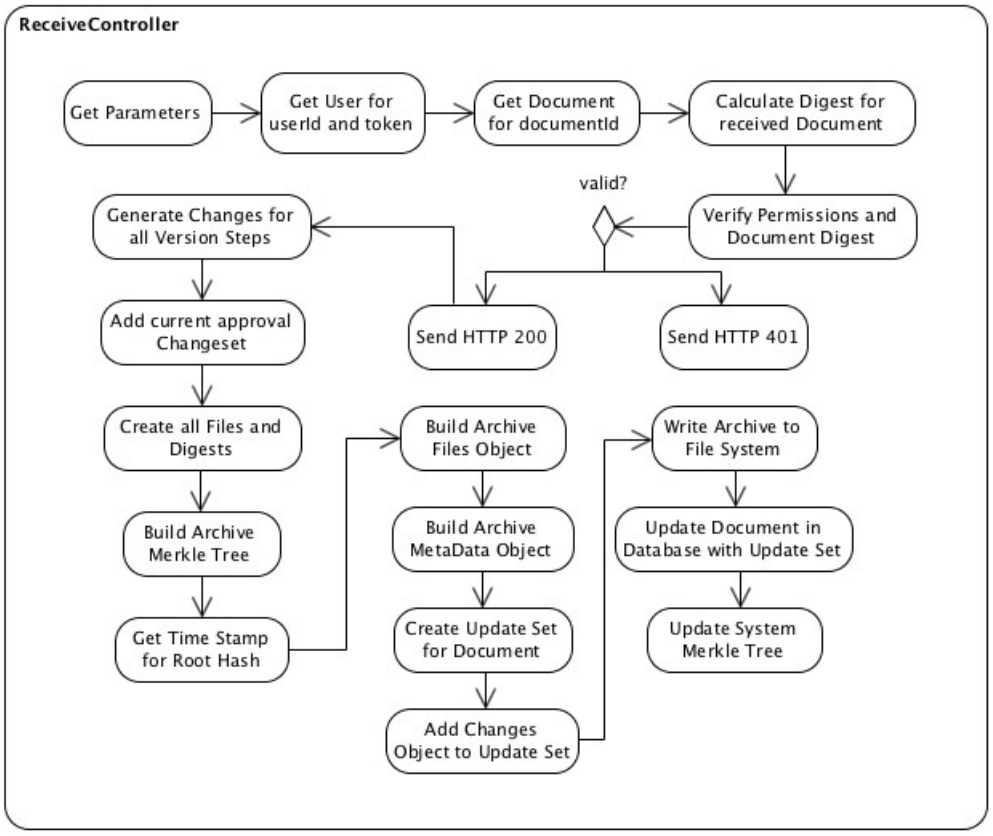

Figure 6.5: Activity Chart of ReceiveController Actions

6.7 Integrity within the System

All current versions of approved documents in the system are stored in a chained Merkle tree as proposed in Section 5.2. Whenever a new approved version of a document is created by signing it, a new archive for that version is created on the file system together with all needed verification information. The actual implementation does not contain the possibility to gather the OCSP response, root certificate of the signer or the timestamp issuer's root certificate to allow for long-time archiving. An implementation meant to be used in an production environment should add this functionality. During development however, complications with the retrieval of all of the mentioned verification information prevented full compliance. For the purpose of the prototypical implementation, the proof-of-concept showing the functionality without all verification information is still sufficient as long as it is fully implemented before production use.

A Merkle tree is created using the *Node.js merkle* package [Mor] with the SHA-256 hashing algorithm. As seen in Listing 6.8, the nodes and leaves of the tree are then extracted and added into a two-dimensional array for further processing. At the end, *output[0][0]* holds the root hash of the tree. This process is the same for the archive Merkle tree and the system-wide Merkle tree with the difference that the system *Merkle* tree holds in addition the root hash of the last tree as its last element. After the tree is created and sent back, the calling method collects an advanced timestamp for the root hash with the method seen in Listing 6.9. The needed timestamps which are based on signed

hash-values with private keys within certificates, qualified or not, are requested through the *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)* [ACPZ]. For testing purposes during the implementation of the system, the free timestamping service of the German Researchnetwork [2] [Ver] was used.

In order to create a timestamp request, two special *Node.js* packages are needed. On the one hand the package *jsrsasign* [Uru] provides the structures and conversion methods to produce a timestamp request purely in *JavaScript*. On the other hand, the package *httpreq* [Dec] makes it possible to send binary HTTP requests and receive the binary answers from within JavaScript, which is not possible with *Meteor's* built-in HTTP package. The result is encoded into Base64 and passed back to the calling method, where it is stored alongside the Merkle tree.

To store these Merkle trees within the system a *vermongo* [FM] versioned MongoDB collection called *MerkleTree* with its own schemata [Doba] is used. The schema has room for the name holding the type of Merkle tree (e.g. *System Tree*), the Merkle tree array, an array of ids with version-indicators of documents or the id *lastTree* (the root hash of the previous system Merkle tree) that were added (e.g. *jnTp5XhHgqqWSf9vL-7*), the timestamp response generated with Listing 6.9, the timestamp provider's certificate and the root hash. In the event of an update to the Merkle tree, the current version of the tree is collected together with all current approved procedure description versions. Next the ids, version numbers and root hashes of their archives are extracted and added together, creating a new system Merkle tree with an id-array and root hash, which is then timestamped. The resulting new *MerkleTree* object is finally used to update the original *MerkleTree* object in the database by referencing the name of the original tree (e.g. *System Tree*). This way there is only one tree for each type of tree, which gets updated and has linked versions attached.

After this process, the new *MerkleTree* JSON object is written out into a log file, generated with the *Node.js bunyan* logger package [Mic] which logs information in the *JSON* format to be easily parse-able. A logrotate [TB] configuration should be used to create a new log file for each day with the current date. The log-files are then to be stored in a directory, that gets synced to the archive server with a WORM storage medium. These syncs should take place every time there is a change to the files in the sync-directory to minimize the amount of time for tampering malicious attackers have to modify data before archiving.

```
create_merkle = function(array) {
  var merkle = Meteor.npmRequire('merkle');

  var tree = merkle('sha256').sync(array);

  var output = [];
  for (var i=0; i<tree.levels(); i++) {
    output.push(tree.level(i));
  }

  return output;
}
```

Listing 6.8: Merkle tree creation

Writing out to the log file and configuring *logrotate* weren't implemented for the prototype but would be possible for a production use-case. The Leibniz Supercomputing Centre (LRZ) does not currently have a secure WORM archiving server. A possible approach to implement such a server would be to create a machine that has a volume which can be written to through an SSH key secured secure shell connection. Once a day this server then would take the root hashes of all documents and the last hash of the Merkle tree in the logs to create a new daily archive server Merkle tree and retrieve a qualified electronic timestamp with all necessary verification information. Once this process has

[2] Author's translation of Deutsches Forschungsnetz

been completed, the archive files, the log and the daily archive server Merkle tree and its timestamped root hash could be written to a secure WORM medium such as a CD or DVD, creating one session per day. All of this could run fully automated and multiple days could be stored on one medium by only adding new files and data on a session base. During the testing phase of the application, archive containers with all required information reached a size of 60 kilobytes, which would indicate that if calculated optimistically, roughly 146.000 versions of approved procedure description archives could be stored on one single DVD+R DL medium with 8.5 GB capacity. Of course a real-world implementation would have to consider the overhead which is added per session. Also DVD+R discs are limited to 157 sessions per disc [Thea]. Taking all this into consideration, such an archiving server would need to have its WORM medium exchanged at least 3 times per year to be able to make daily WORM copies of the system.

```
getTsResp = function(hashValue) {
  var jsrsasign = Meteor.npmRequire('jsrsasign');
  var fut       = new Future();
  var httpreq = Meteor.npmRequire('httpreq');

  var json = {
    mi: { hashAlg: 'sha256',
          hashValue: hashValue }
  };

  json.certreq = true;

  var o = new jsrsasign.asn1.tsp.TimeStampReq(json);
  var hex = o.getEncodedHex();
  var b64 = jsrsasign.hex2b64(hex);
  var pemBody = b64.replace(/(.{64})/g, "$1\r\n");
  pemBody = pemBody.replace(/\r\n$/, '');

  var b = new Buffer(pemBody, 'base64');

  httpreq.post('http://zeitstempel.dfn.de',
    {headers: {'Content-Type': 'application/timestamp-query'},
      binary: true, body: b}, function (err, res) {

    if (err) {
      console.log(err);
    } else {
      fut.return(res);
    }
  });

  var binaryResult = fut.wait();

  return binaryResult.body.toString('base64');
}
```

Listing 6.9: Time Stamp Request for Hashes

6.8 User Access and Permission Management

Aside from the mechanisms for creating and managing documents within the system, user and permission management also is an integral part that needs to be handled properly to ensure documents

cannot be manipulated or extracted without the proper rights. This section goes into detail how to achieve this level of security for users and documents.

6.8.1 User Authentication

Users are authenticated through a Lightweight Directory Access Protocol (LDAP) server which is connected to the system through a secure TLS encrypted connection. The integration of the protocol was achieved by adding the *Meteor accounts-ldap* package [Typ], which extends the platform-integrated *Meteor accounts-password* package. Configuration of the package happens in the *startup.js* file on the server side. The LDAP server's TLS certificate chain can be obtained with the command shown in Listing 6.12. It is to be noted, that the command will retrieve a certificate chain from the LDAP server and must be reordered so that the root-certificate is the first and not the last. Otherwise the used LDAP client package cannot connect to the LDAP server. This goes against best practices since the default and recommended order is from leaf to root as described in the Transport Layer Security (TLS) Protocol specification [DR06] in Section 7.4.2. *Server Certificate* under *certificate_list* and should be fixed. This certificate chain can then be added to the project in the *private* folder and handed over during the configuration process when starting the application as seen in Listing 6.10. In the listing the filename and location of the certificate chain, the secure port and address for the LDAP server are configured, together with a profile map of which attributes to add to the automatically created user-account within the system. By default the first name, second name and display name are added to the user's profile to make it easier for the data protection commissioner to see who created a document or who edited it. A more ideal solution would be to not add the information to the profile, but actually retrieve it from the LDAP server every time a document is reviewed to remove the need to delete the profile information in the case where an employee leaves the institution and wants their personal information removed from the application. In this case though, it would be necessary to store credentials for the application to query the information from the LDAP server when needed. Since all users present on the LDAP server must first officially fill out a form and provide legal identification before receiving access, every person using the system can be uniquely identified with name and surname during usage.

To make sure no local users can be created and be used to login without the LDAP authentication, account creation must be disabled both on the client and on the server. For this the code in Listing 6.11 must be added to the *config.js* files in the *server* and *client* folders.

```
Meteor.startup(function () {
  LDAP_DEFAULTS.ldapsCertificate =
        Assets.getText('ldaps/ldapserver.pem');
  LDAP_DEFAULTS.port = 636;
  LDAP_DEFAULTS.url = "ldaps://auth.sim.lrz.de";

  LDAP_DEFAULTS.searchResultsProfileMap = [{
    resultKey: 'mwnSn',
    profileProperty: 'lastName'
  }, {
    resultKey: 'mwnGivenName',
    profileProperty: 'firstName'
  }, {
    resultKey: 'mwnAnzeigename',
    profileProperty: 'name'
  }];
});
```

Listing 6.10: LDAP configuration when starting the application

```
Accounts.config({
```

```
  forbidClientAccountCreation : true
});
```

Listing 6.11: Disable account creation on server and client

```
openssl s_client -showcerts -servername auth.sim.lrz.de \
  -connect auth.sim.lrz.de:636 < /dev/null | \
  sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' \
  > ldapserver.pem
```

Listing 6.12: Bash command to retrieve LDAPS certificate chain

Since the *accounts-ldap* package cannot be used with the *meteor-accounts-ui-bootstrap-3* package [Mar] out of the box, the *meteor-accounts-ui-bootstrap-3* must be added to the *packages* folder and modified to add LDAP authenticated logins. Listing 6.13 shows the changes that were made to the *login_buttons_dropdown.js* file. As can be seen, the original *loginWithPassword* method call was replaced with a shorter representation and an externalized callback function. In addition, the *loginWithLDAP* method call was added, which uses the new *ldapOptions* object to authenticate users with their common name, their id within the organization. Since the LDAP server's configuration and directory were not modified for the implementation of this prototype system, in the actual project a check was implemented to allow for a local *admin* user to log into the system, ignoring the LDAP server authentication and using only the local user database. This approach is not encouraged and was only used for testing purposes and to set the initial roles described in Section 6.8.2. In a production version of the system all authentication should go through the LDAP server and roles should be managed through the LDAP server's database.

```
...
var login = function() {
  ...

//  Meteor.loginWithPassword(...) {
//    ...
//  });

  var ldapOptions = {
    dn: "cn=" + username +
    ",ou=Kerberos,ou=Kennungen,o=lrz-muenchen,c=de",
    search: "(&(objectclass=Person)(cn=" + username + "))"
  };

  var callback = function(error, result) {
    if (error) {
      ...
    } else {
      loginButtonsSession.closeDropdown();
    }
  };

  Meteor.loginWithLDAP(username, password, ldapOptions, callback);
};
...
```

Listing 6.13: Changes to meteor-accounts-ui-bootstrap-3

6.8.2 Permission Management

Permission management, as mentioned in Section 6.6.3, is handled with the *Meteor meteor-roles* package [Lan]. For this prototype solution no modifications were made to the LDAP user directory regarding group-memberships. Therefore, all groups and access-restrictions are handled on the developed system. For testing purposes an *admin* user was introduced who automatically gets added to the system on its first launch, in case no users are present in the system. This user is given the *admin* role, which allows him to edit other users permissions to see documents and for him to promote and demote users with the *datenschutzBeauftragter* role. When implemented for real production use, this user should not be created but all access rights should be managed through the directory access protocol on the LDAP server. In the case of the use of these external groups, groups would need to be added every time the user logs in to the system, which would mean the *Meteor accounts-ldap* [Typ] package would need to be modified to check for these group-changes and add them to the user-object upon login.

Access controls were added to all locations at which they are required, mainly when publishing collections on the server, approving documents, managing user-rights and deleting documents. As was proposed in Section 5.6, permissions to view or edit documents are granted by adding the user to a role represented by the document's id. Users that create documents are automatically added to the role of the document by collection hooks. Others can be added to that role by the person with the *admin* role. The data protection commissioner with the role *datenschutzBeauftragter* can see, edit, delete and approve all documents. If the *admin* role is added to the data protection commissioner, he can also grant other users permissions to view and edit documents. Approved documents cannot be deleted anymore by normal users, but only by the data protection commissioner.

All permission checks are implemented on the server side and on the client side by hiding buttons and also checking permissions before performing the actual action. To manage roles, the *Meteor Accounts Admin UI* package [Har] was installed and added to the *packages* directory and a button to manage user rights was added to the user-menu for users with the *admin* role. If this solution were to be used in production, there would need to be an audit log for permission changes regarding which roles users have at any given time within the system. This log should ideally be a chained remote log, with intermediate hashes of the chain written to a WORM medium regularly to ensure integrity of the logs. In case the roles are managed through the LDAP server, logging should be implemented on the LDAP server but could be run additionally on the currently developed system for redundancy.

6.9 Hardening the System

In addition to the already mentioned measures in this chapter regarding security during implementation, multiple further steps are taken to protect the system from unauthorized manipulation and misuse. Since *Meteor* uses the publish-subscribe model to make data available to clients, any subscribe request for information is verified to ensure only logged in users and users with specific rights get access to information they are allowed to see. This restriction happens in the *publish.js* file on the server side with the *Meteor.publish(...)* method and is enforced by removing the *Meteor* packages *insecure* and *autopublish*. *insecure* allows for any client to perform any action on the connected database without further verification and *autopublish* automatically gives read-access to all collections by default. Keeping these packages installed would be negligent. Any changes that need to be performed to the database, such as insert, update or remove, can only be made by calling *Meteor* methods from the client side, which verify the user's permission and the validity of the proposed changes before further processing the request. In addition, requests made from the client through *Meteor* methods are also verified with the user's access tokens, which are stored in the browser's *local storage*.

After access rights have been verified, document content validity checks are performed, cleaning,

validating and preprocessing changes with the database schema of the collection described in Section 6.3 and briefly mentioned in Section 6.2. Only after all checks have passed and no unauthorized changes were handed over, are the changes passed to the database for processing. There is no way to work around versioning since even the versioned collections are restricted in regards to client access. Versions can only be changed on the server side and every change sent from the client causes the working copy to be saved anew as a new version. In the case of input processing of fields, that are not managed by the *Meteor autoform* package [Dobb], as with searches, login and others, regular expressions and other filter rules are used to clean dangerous characters and input before passing it to the database for queries and handling.

All routes that are configured within the *Iron.Router* are secured with a global *onBeforeAction*, which checks if the user who made the request is logged in. In special cases or in case of routes that cannot be checked this way, an additional measure was taken, using cookies with the *Meteor* package *meteor-cookies* [Gro] to check the user's access token before proceeding. If a user does not have correct access rights or is not logged in, an error message should be shown or the user should be redirected to the main page of the application.

The *ReceiveController* described in Section 6.6.3, in addition to verifying the user's access rights with id and token also checks the files sent with the request to ensure there cannot be any malicious data injection. The actual implementation does not check the received signature because there were issues with *Node.js futures* [Lav] colliding with the *Node.js spawn* child processes that are necessary to verify them. This should be further investigated to only allow verified signatures to be saved within the system.

By default, *Meteor* users created and maintained by the internal *accounts-password* package remain logged in indefinitely until they choose to actually press the *logout* button. To counteract this behaviour, the *Meteor zuuk:stale-session* package was installed to automatically remove sessions which have been stale for thirty minutes. This is the default behaviour for the package and is achieved by sending *heartbeats* from the client to the server every three minutes and purging the session once it has gone stale and stopped receiving heartbeats for the specified amount of time.

To ensure secure communications between client and server, a TLS certificate was created and added to the *Apache* proxy server handling the routes of the *Meteor* application. To use the server in an production environment, such a certificate should be requested from the institution, so browsers can also successfully verify its origin. All activity on port 80 is redirected to the secure port 443, including web-socket routes. These redirects are also configured in the *Apache* proxy server by using the modules *proxy*, *proxy_ http* and *proxy_ wstunnel*. All other ports of the system are blocked through a firewall configured with the *ferm* [KK] frontend for iptables to only allow ports 80 and 443. Port 80 could have been disabled, but was left activated for compatibility and usability reasons. The *MongoDB* database is installed on the same system as the *Meteor* web-application and is not exposed to the outside. It was not explicitly hardened but should be hardened in an production environment by at least following the security guidelines proposed by the creators themselves [Monc]. If the database were to be exposed to the outside or run in a cluster for higher availability, even stricter approaches to harden the system should be taken.

Both the *MongoDB* database and the *Meteor* web-application are configured as self-restarting *systemd* services with production flags. Before actually running the web-application, the *Meteor* application is also first converted into a pure *Node.js* production application using the *Meteor build* command to bundle dependency, logic and style files and optimize the application for faster loading.

The system can only be accessed through the use of SSH keys and should not allow for pure password-based logins. All shell access should be documented with all changes made and information accessed in a safely kept remote server log.

7 Evaluation

After the conception and implementation of the software solution to store procedure descriptions with legal admissibility, the developed system needs to be evaluated in regards to the initially defined requirements and its ability to keep information secure and confidential while preserving integrity and authenticity of documents.

7.1 Security, Integrity and Authenticity

A document storage solution is only as good at storing information as it can protect that information from being unknowingly tampered with or viewed by unauthorized parties. Therefore the security, integrity and authenticity of the storage solution is of utmost interest. The way information is transferred, processed and stored plays a big role in all of this. Sections 5.7 and 6.9 explicitly touch upon this topic by describing steps which should be and were taken to protect the system and its content from unauthorized viewing and modifications:

There are technical and organizational measures which were proposed and should be followed when creating and running the system in a production environment. The system itself takes care of ensuring authenticity and integrity of documents by only allowing approval of documents with qualified electronic signatures, while also timestamping all verification and archive documents with timestamps before they are transferred to a secure archiving server with WORM capability as mentioned in Section 6.7. In the same way, logs of the internal structure and organisation of approved documents within the system are sent to the archiving server to make it possible to later verify the chained version history and structure of those documents between each other.

Access to the web-application is restricted and protected through many different mechanisms such as route-checking, token-verification, role-management and publication limiting. Tampering is prevented by only allowing database operations after verifying, validating and preprocessing of change-requests and their origin made by the client in *Meteor* methods. All changes to the procedure description content itself are versioned and traceable through changesets added to each version. Communication with the application and the authenticating LDAP server is secured by only allowing access through TLS protected channels.

Although the web-application and the underlying system are hardened and closed in itself, administrators with the right SSH key could gain access and make modifications to the database content. This would invalidate the attached and archived verification information on the server itself, but not the information which is securely stored on the archiving server and on its WORM archiving media. To catch these kinds of modifications and see them in the web-application, it would make sense to integrate verification of documents and their verification information by cross-checking it with the versions stored on the archiving server every time documents are viewed or upon request. Invalidating stored procedure descriptions so they wouldn't be evidentially valid in court any more, would require the destroying of the WORM volumes which hold the approved procedure description archives. Since this is highly unlikely without a high level break-in into the actual data centre or the data centre's own administrators manipulating media, data stored on WORM volumes in the structured format proposed in this thesis can be regarded as safe. In any case, the data protection commissioner, just as with the approval of paper-based procedure descriptions, is ultimately responsible for verifying the information contained in procedure descriptions before approving them. Everything after the

approval is taken care of by the system and its archiving server.

7.2 Analysis of Requirements

Next, the fulfilment of the requirements of Chapter 3 need to be evaluated. For this, each requirement is listed and examined separately. Each requirement can have one of three markers set next to its short title and significance, displaying the fulfilment of the requirement:

Fulfilled ✓

Unfulfilled ×

Partially Fulfilled •

1. Creation of Documents - Mandatory ✓

Section 6.3 describes how documents can be created and edited by using an implementation of the *Meteor autoform* package [Dobb]. These documents are directly stored in the database and saved to the file system upon approval. These created documents together with their verification files and information for integrity and authenticity have the same evidential value as normal paper documents would have in court.

2. Approval of Documents - Mandatory ✓

Documents are approved by pressing a button, which starts the document signing process. Only the data protection commissioner is allowed to approve documents and qualified electronic signatures are a requirement set in the properties of the signature creation software as can be seen in sections 6.6.2 and 6.8.2. Thanks to the qualified electronic signature the approved and saved procedure descriptions have the same value as their paper based counterparts have. Their integrity can also be proven at the latest once they have been timestamped with a qualified electronic timestamp once a day on the archiving server.

3. Authenticity - Mandatory ✓

Similar to point 2, authenticity of approved documents is given by the use of qualified electronic signatures which are used during the approval process. Working copies cannot be verified regarding their authenticity and this was never intended since they were excluded from this requirement in Section 5.4.1.

4. Addition to Registry - Optional ×

Documents are not added to a registry as such that is publicly accessible. All current approved procedure descriptions though can be retrieved by the data protection commissioner with the use of searching and sorting. In the case that a third party should request to see the content of these documents, it would be possible for the data protection commissioner to forward them or make them available to that third party within minutes. Addition to a special registry is therefore not necessarily needed, which is why it was declared *optional* in Section 3.1. For this reason and to favor more important features, explicit addition to a registry was not conceptualized or implemented.

5. Archive Documents externally - Optional •

Approved procedure descriptions are added to archive containers which in the conceptual part hold all the information necessary to verify their authenticity and integrity, while also providing alternative representations of its content. These containers are then stored in a special directory

in a predefined structure so that they can be retrieved easily should the need arise. This directory can be regularly synced to an external archive server where the contained files could be added to a Merkle tree and timestamped with a qualified electronic timestamp. Conceptionally the requirement is met, but since the OCSP response and the root certificates of neither the signature certificate or the advanced electronic timestamp are added in the actual prototype implementation, archiving to an external archive server is not fully compliant due to complications during retrieval of the information, as stated in Section 6.7. Since the implementation is not fully complete and compliant with the requirement it is only met partially.

6. Archive Documents internally - Nice-to-have ×

To build an archiving solution within the document storage solution is beyond the scope of this thesis. As described under point 5 all requirements for archiving are given within the concept. If all verification files and meta data were present within the archives in the implemented prototype, it would be possible to attach a WORM medium within the own system and build the Merkle tree with a timestamped root hash that was created with a qualified electronic signature. But none of this was conceptualized or implemented, since no WORM storage solution is currently present in the LRZ which could be attached and used within the virtual or dedicated hardware machines.

7. Workflows - Optional •

Workflows in a simple form have been developed within the concept but were not implemented. The most simple form of an implemented workflow system could be recognized if the mere fact were taken, that all created documents have a flag showing if the document is waiting for approval or not and the data protection commissioner could, with small modifications to the interface, filter for non-approved documents to approve these whenever they become available. Implementing a full workflow system would require further and more complicated algorithm and database implementations which were abandoned in favour of more important features such as approval of documents and higher security of the system. Since the conceptualized workflow system was not fully implemented, the requirement is only partially fulfilled.

8. Permission Management - Mandatory ✓

As described in Sections 5.6 and 6.8.2, users are authenticated using an LDAP server and their roles are managed via the two *Meteor* packages *meteor-roles* [Lan] and *Accounts Admin UI* [Har]. There are roles called *admin*, *datenschutzBeauftragter* and users without roles, implementing the access rights seen in Table 5.1 in the concept of Section 5.6.

9. Retrieval of Documents - Mandatory ✓

There are many different ways in which documents can be retrieved. Every created document can be retrieved in PDF or XML format. They can also be viewed directly within the application using the dynamically populated templates of the system. In addition, approved documents can be retrieved as entire archive containers with the formatting described in Section 5.4.4. In the case of a system failure, all documents are also available and retrievable from the archive server in case a connection to an archive server was implemented by syncing the archive containers to this server regularly.

10. Simple Sorting - Mandatory ✓

Not directly mentioned in Chapter 5 and only touched in passing in Section 6.3, sorting of documents within the final prototype implementation is possible using the *serviceShortTitle*, *createdBy* and *createdAt* attributes.

11. Simple Searching - Mandatory ✓

Similar to point 10, searching is possible in the final prototype implementation even if it is only shortly mentioned in Section 6.3. Specifically, *serviceShortTitle*, *createdBy*, *serviceName*, *purpose*, *circleAffected*, *typeOfTransmittedData*, *titleAndLocationOfSystem*, *operatingSystem*, and *usedSoftware* can be searched.

12. Complex Sorting - Optional ×

Complex sorting was neither conceptualized, nor implemented, since there would have been a need for bigger database modifications to improve speed with complex sorting, which in turn has an effect on how the interfaces are implemented. Any kind of procedure description can also be found with simple sorting for now.

13. Complex Searching - Optional ×

Complex searching was neither conceptualized, nor implemented, for similar reasons to complex sorting. The database modifications and interface impact were not justified by the possible benefit complex searching would have had on the developed system and its use.

14. Version Control and Change Logging - Mandatory ✓

Both Section 5.5 and Section 6.4 cover this topic extensively. Not only are documents versioned within the database with accompanying change-sets, but approved procedure descriptions are also stored in a versioned manner and hold all modifications that have been made since they were created.

15. Re-Approval - Optional ✓

Although not directly mentioned in Sections 5.3 or 6.6, re-approval is possible due to the fact that any change to the *content* section creates a new version of a procedure description. As long as re-approval is not specifically disabled, it is possible to approve an already approved procedure again by signing it again and thus creating a new archive container that will be stored on the archiving server.

16. Automatic Re-Submission - Nice-to-have ×

Not one single section makes any reference to the possibility to re-submit documents automatically after a given time-period. It was neither conceptualized nor implemented, due to the need for good notification strategies in addition to scheduling, which would have been too time-consuming considering the low priority of the feature compared to more important features.

17. Export - Mandatory ✓

Documents can be manually exported in different formats. All documents can be exported as PDF and in XML notation and approved documents can be exported in form of an archive container. Automatic exports are performed every time a procedure description has been approved by storing the generated archive container in a directory which can be synced to an archive server regularly.

18. Import - Optional ×

There is no mention of import functionality and it was neither conceptualized nor implemented due the fact that it would need to cover duplicate handling and import strategies. Since it is possible to create documents and approve them within a very short time, import functionality was left out in favour of more important features.

19. Backup - Mandatory ✓

There is the possibility to perform backups of the database and the application data with direct access to the underlying system. This access needs to be monitored and logged closely and must follow the guidelines mentioned in Section 5.7. Otherwise a form of backup is performed when archive containers are stored on an archive server in a structured manner. It would be possible to recreate a lost system with some work just from those archive containers. Other than that no explicit backup system or mechanism was implemented.

20. Statistics - Nice-to-have ×

It is not possible to evaluate procedure descriptions statistically. This functionality has not been conceptualized nor implemented, since there was no specific use-case present at the time of development and other features were more important. A working concept would have also taken up too much time to design properly, which the gained benefit wouldn't justify within this thesis.

21. Availability - Mandatory ✓

Measures to ensure availability are mentioned in Section 5.9 but only the measure to run the system with *systemd* was implemented and used in the prototype. To create a clustered *MongoDB* database, further security measures would need to be taken. Monitoring other than *systemd* log-generation which could be sent to an external server for analysis was not implemented.

22. Integrity - Mandatory ✓

How to ensure integrity of documents and within the system is described in detail in Section 5.2 and fully implemented in the prototype system as mentioned in Section 6.7. Archive and chained system Merkle trees were used with logging of the tree structure and timestamps. All of this information can then be stored on a WORM medium within an archive server in case the sync-process to that server has been set up.

23. Documentation - Mandatory ✓

This thesis and especially Chapter 5 and Chapter 6 are extensive documentation of how the system is set up and how it works. In addition, important source code sections have been commented for better understanding.

24. Usability - Optional •

Usability as such is hard to measure. Although the interface is not explicitly mentioned in neither Chapter 5 nor in Chapter 6, it is created in an intuitive and simple way, splitting tasks up into separate actions. Each action also has its own view. There isn't hardly any error handling in place which could give cause for a bad user-experience in case of misuse. Further and more detailed evaluation, perhaps with user-studies, would be needed to better measure the implementation of this requirement. Due to its simple interface this requirement is considered at least partially fulfilled, with room for further investigations and improvements.

25. Low Price - Mandatory ✓

Since no software components were used that require one-time, monthly or yearly fees, only the hardware and signature certificate define the price. These were selected in Section 6.6.1 in a way that assures the lowest long-term price.

8 Conclusion and Outlook

Paper-based document management has been around as long as offices exist. With it come immense amounts of work and overhead to keep documents organized, retrievable and up-to-date. Not only timely and organizational costs develop from this, but storage space, printers, paper, folders and filing cabinets add to the price of maintaining documents in this form. Ever since the computer age started, companies and individuals have therefore dreamt of the paperless office, keeping everything entirely digital, which not only is supposed to save costs but also improve productivity. Implementation of such a solution though, with legal admissibility, especially considering documents which need to have the written form, is still considered non-trivial and costly. This thesis is meant to analyse the requirements for such a system, conceptualize a purely digital storage solution and implement it for the specific purpose of storing procedure descriptions which require the written form at the Leibniz Supercomputing Centre (LRZ). This chapter gives an overview of what has been developed and found in this thesis and concludes with an outlook and a view into which fields further research could go regarding the findings of this thesis.

8.1 Summary

To begin with, Chapter 1 details the current situation regarding document storage and procedure description management at the LRZ, the problem with it and a proposed solution and approach path. Once the path has been set, Chapter 2 takes over by introducing the reader to the technical and legal basics on which to base a possible solution. Taking the found basics, laws and regulations into consideration, the scenarios described in Chapter 3 are converted into requirements the system must, should and could fulfil. These requirements are then used to evaluate current solutions in the industry and components on which the solution could be built. Since no solution existed at the time of writing the thesis to fulfil all mandatory requirements, an individual and new solution can be justified. Considering the requirements of Chapter 3, Chapter 5 describes and conceptualizes a possible solution to the proposed problem based on the approach of Chapter 1. Key areas of such a solution are analysed and answers to the arising questions are proposed, which deal with ensuring authenticity, confidentiality and integrity of documents and how to make the system and its functionality safe. Based on the described concept, Chapter 6 describes the implementation of a prototype that can fulfil the requirements of Chapter 3. Language-, Framework- and Software-specific problems during the process of handling individual areas are highlighted and solutions proposed. Chapter 7 next takes the developed system and evaluates its security, integrity, handling of authenticity and functionality in contrast of the initial requirements of Chapter 3 to determine if the proposed solution is within the defined boundaries and which areas could or should be improved.

Due to financial and timely constraints, it was not possible to consider all requirements of Chapter 3 during the development process. To preserve resources and make it possible to develop and implement a solution within the frame of this Master's Thesis, there was a need to filter out the most important requirements and describe and implement these in detail.

Overall, all thirteen mandatory requirements were fully implemented, three optional requirements were implemented partially, while none of the nice-to-have requirements were implemented. This means the solution is not quite production ready, but with a few modifications could be used to create, edit and store procedure descriptions at the LRZ in the near future, while ensuring that documents

retain their integrity and authenticity even in the long run. Since all mandatory requirements set in Chapter 3 have been met, the prototype can be seen as a success and could be used as a reference to build a more sophisticated production-ready solution which could in the end meet all requirements.

The concept has been developed and the prototype works for what it was intended for. For limited use and the special use case at the LRZ this could be a viable solution.

8.2 Outlook and Further Research

The developed system, although lacking in some optional requirements, is a good step in the right direction towards a document management and storage solution for procedure descriptions at the Leibniz Supercomputing Centre (LRZ). Deficits of the system, mentioned in chapters 6 and 7 should be sorted out before using the solution productively. This includes but is not limited to adding all data needed for verification of signatures and timestamps to archive containers before archiving them, so that they can be considered long-term preserving archives. The secure WORM archiving server solution must be implemented to store documents in a way in which they and their resulting hash-values can no longer be altered. As already proposed in the technical guideline BSI 03125 TR-ESOR [Feda] on preservation of evidential value of electronically signed documents, verification of all documents and their verification information before creating the archive and sending it to the archiving server should be implemented as well. Verifying signatures or timestamps from within the web-application should also be considered. A self-check mechanism could be added on top of that, which checks all currently stored documents, archives, verification information and database entries regarding their authenticity and integrity and that can detect unauthorized modifications to the underlying code-base. Database entries could be encrypted with externally stored keys which are assigned based on groups within the LDAP directory server. This way it would no longer be possible to just read any entry from the database and pure backups of the database would be useless for an attacker.

Aside from the mentioned improvement possibilities of the developed application, work on this thesis brings up some questions regarding the legal and technical implications when storing documents with legal admissibility. For one, are there better and more feasible options to ensure and prove the authenticity and integrity of electronic documents, for which perhaps laws need to be changed or extended? Could integrity, for example, be ensured through the posting of hash-values into a well known cryptographic currency block-chain and authenticity with a person's public and private PGP keys and would it hold up in court? Can cryptographic and distributed software be created which makes using a single or dedicated archive server or WORM mediums unnecessary? Is there a way to make creating qualified electronic signatures as easy and fast as downloading an application to a smartphone? Creating such solutions would greatly increase adoption of these new methods of signing and storing documents and would ensure higher exposure within non-technical user groups. All of this should be further explored to improve on the current state of the art and to make it easier for less tech-savvy users and companies to start using these solutions and technologies.

On the 8th of April 2016, after the theoretical and practical part of the thesis had been finished, SecCommerce released their SecSigner 5 update [Sec]. With this update, SecSigner 4's old NPAPI integration of the applet is superseded by the possibility to start applets with Java-Web-Start. This new solution could enable the developed application to be viable even in the future after integrating and configuring the new SecSigner version.

Considering the aforementioned topics of improvement and the potential answers to the questions asked, further work in this direction seems worth pursuing and future development stemming from this initial approach and its implications would be worth following.

List of Figures

Listings

Bibliography

[199] *DIRECTIVE 1999/93/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 December 1999 on a Community framework for electronic signatures*. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31999L0093, . – Accessed: 2016-01-20

[2ne] 2NET® Carsten Lang: *2net Carsten Lang*. http://www.sidoc.info, . – Accessed: 2016-03-24

[ACPZ] Adams, C. ; Cain, P. ; Pinkas, D. ; Zuccherato, R.: *Internet X.509 Public Key Infrastructure - Time-Stamp Protocol (TSP)*. https://www.ietf.org/rfc/rfc3161.txt, . – Accessed: 2016-03-18

[Ado] Adobe Systems Incorporated: *PDF Reference - third edition - Adobe Portable Document Format - Version 1.4*. http://partners.adobe.com/public/developer/en/pdf/PDFReference.pdf, . – Accessed: 2016-01-20

[Ado03] Adobe Systems Incorporated: *PDF as a Standard for Archiving*. http://www.adobe.com/enterprise/pdfs/pdfarchiving.pdf, 2003. – Accessed: 2016-01-20

[App] Apple Inc.: *WebKit*. https://webkit.org, . – Accessed: 2016-03-09

[Ara] Arachnys Information Services Ltd: *Cabot - monitor and alert*. http://cabotapp.com, . – Accessed: 2016-03-16

[Atl] Atlassian: *Identity management for web apps*. https://www.atlassian.com/software/crowd/overview, . – Accessed: 2016-03-05

[bay] *Juris - Bayerisches Datenschutzgesetz*. http://byds.juris.de/byds/009_1.1_DSG_BY_1993_rahmen.html, . – Accessed: 2015-10-18

[bcr] *A Future-Adaptable Password Scheme*. http://usenix.org/legacy/publications/library/proceedings/usenix99/provos/provos.pdf, . – Accessed: 2016-01-14

[bds] *Bundesdatenschutzgesetz*. http://www.gesetze-im-internet.de/bdsg_1990/index.html, . – Accessed: 2015-10-10

[bgb] *BGB - Schriftform*. http://www.gesetze-im-internet.de/bgb/__126.html, . – Accessed: 2015-10-25

[Bou] Bouchard, Mathieu: *GitHub - Meteor Collection Hooks*. https://github.com/matb33/meteor-collection-hooks, . – Accessed: 2016-02-19

[Buna] Bundesdruckerei GmbH: *Bundesdruckerei - eServices*. https://www.bundesdruckerei.de/de/162-e-services, . – Accessed: 2016-03-19

[Bunb] Bundesdruckerei GmbH: *D-TRUST-ZeitStempel*. https://www.bundesdruckerei.de/sites/default/files/documents/2014/03/produktblatt_zeitstempel_de.pdf, . – Accessed: 2016-02-19

[Bunc] Bundesdruckerei GmbH: *sign-me - ein Service der Bundesdruckerei*. https://live.esign-service.de/esign/Home, . – Accessed: 2016-03-07

[Bund] Bundesdruckerei GmbH: *Signatursoftware für sign-me*. https://www.bundesdruckerei.de/de/3173-software, . – Accessed: 2016-02-19

[Com] Computer Sciences Corp: *CSC-Studie: GSTOOL Quo Vadis?* https://www.csc.com/de/insights/133725-csc_cybersecurity_studie_gstool_quo_vadis, . – Accessed: 2016-02-18

[Dav] Davis, Chris: *Graphite Documentation - Graphite 0.10.0 documentation.* http://graphite.readthedocs.org/en/latest/, . – Accessed: 2016-03-16

[DBP96] Dobbertin, Hans ; Bosselaers, Antoon ; Preneel, Bart: *RIPEMD-160: A Strengthened Version of RIPEMD*, Katholieke Universiteit Leuven, ESAT-COSIC, Diss., April 1996. – Accessed: 2016-01-14

[Dec] Decrock, Sam: *httpreq.* https://www.npmjs.com/package/httpreq, . – Accessed: 2016-03-18

[Deu] Deutsche Telekom AG: *Auftrag Signaturkarte.* https://www.telesec.de/de/signaturkarte/pks-auftrag, . – Accessed: 2016-02-18

[DGNa] DGN Deutsches Gesundheitsnetz GmbH: *DGN | Deutsches Gesundheitsnetz.* http://www.dgn.de, . – Accessed: 2016-03-19

[DGNb] DGN Deutsches Gesundheitsnetz Service GmbH: *CHERRY ST-2000: Der kleine Alleskönner.* http://www.dgn.de/cherry-st-2000-der-kleine-alleskoenner, . – Accessed: 2016-02-18

[DGNc] DGN Deutsches Gesundheitsnetz Service GmbH: *DGN sprintCard: Die Karte für Starter.* http://www.dgn.de/produkte/fuer-den-rechtsgueltigen-elektronischen-datenverkehr-signaturkarten-zubehoer/dgn-sprintcard-die-karte-fuer-starter, . – Accessed: 2016-02-18

[DHC] DHC Business Solutions GmbH & Co. KG: *Enterprise Management System DHC Vision.* http://www.dhc-vision.com, . – Accessed: 2016-03-24

[Doba] Dobbertin, Eric: *GitHub — Simple-Schema.* https://github.com/aldeed/meteor-simple-schema/, . – Accessed: 2016-01-20

[Dobb] Dobbertin, Eric: *GitHub - Meteor AutoForm.* https://github.com/aldeed/meteor-autoform, . – Accessed: 2016-02-19

[Dobc] Dobbertin, Eric: *GitHub - Meteor Collection2.* https://github.com/aldeed/meteor-collection2, . – Accessed: 2016-02-19

[DR06] Dierks, T. ; Rescorla, E.: *RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1.* http://tools.ietf.org/html/rfc4346#section-7.4.2, April 2006. – Accessed: 2016-03-18

[ecm] *Standard ECMA-262 - ECMAScript® 2015 Language Specification.* http://www.ecma-international.org/publications/standards/Ecma-262.htm, . – Accessed: 2016-01-20

[eid] *REGULATION (EU) No 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC.* http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910, . – Accessed: 2016-01-20

[ER13] Elliott, Kennedy ; Rupar, Terri: *Six months of revelations on NSA.* http://www.washingtonpost.com/wp-srv/special/national/nsa-timeline, Dezember 2013. – Accessed: 2015-10-25

[Esc15] Eschweiler, Sebastian: *Github — iron.* https://scotch.io/tutorials/how-to-speed-up-meteor-development-with-scaffolding-and-automatic-form-generation, Juni 2015. – Accessed: 2016-03-09

[eu9] *EU Data Protection Directive 95/46/EC*. Accessed: 2015-12-04,

[Eur] European Commission: *General Data Protection Regulation*. http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf, . – Accessed: 2015-12-04

[Eur11] European Commission: *MoReq2010, modular requirements for records systems - Core services and plug-in modules*. MoReq, 2011. – ISBN 978–92–79–18519–9

[exc] exceet Secure Solutions AG: *exceet Secure Solutions: M2M, IoT & IT Security*. http://www.exceet-secure-solutions.de, . – Accessed: 2016-03-19

[Feda] Federal Office for Information Security (BSI): *BSI 03125 TR-ESOR*. https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/TR03125/BSITR03125.html, . – Accessed: 2015-12-11

[Fedb] Federal Office for Information Security (BSI): *BSI 03125 TR-ESOR Anlage F*. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG03125/BSI_TR_03125_TR-ESOR-F_V1_2_EN.pdf, . – Accessed: 2015-12-28

[Fedc] Federal Office for Information Security (BSI): *IT-Grundschutz - GSTool*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/GSTOOL/gstool_node.html, . – Accessed: 2016-02-18

[Fit] Fitzgerald, Bret: *Novell, Inc.* https://www.novell.com/de-de/products/filr/, . – Accessed: 2015-12-11

[FM] Faivre-Maçon, Mickael: *GitHub - Meteor Vermongo*. https://github.com/micktaiwan/meteor-vermongo, . – Accessed: 2016-02-19

[Fou] Foundation, OpenID: *OpenID Foundation Website*. http://openid.net, . – Accessed: 2016-03-05

[Gan] Gandy, Dave: *Font Awesome, the iconic font and CSS toolkit*. https://fortawesome.github.io/Font-Awesome/, . – Accessed: 2016-03-09

[Gar] García, Oliver: *BGB - Elektronische Form*. http://dejure.org/gesetze/BGB/126a.html, . – Accessed: 2015-10-25

[Git] GitHub Inc.: *HUBOT*. https://hubot.github.com, . – Accessed: 2016-03-16

[Gmb] GmbH, Bundesdruckerei: *D-Trust card*. https://www.bundesdruckerei.de/de/164-d-trust-card, . – Accessed: 2016-02-18

[Gov] Governikus GmbH & Co. KG: *AusweisApp 2*. https://www.ausweisapp.bund.de/startseite, . – Accessed: 2016-02-18

[Gro] Grochowski, Dave: *GitHub - ThePumpingLemma/meteor-cookies: Meteor package that wrap the cookies package from NPM*. https://github.com/ThePumpingLemma/meteor-cookies, . – Accessed: 2016-03-19

[Har] Harnisch, Harrison: *GitHub - hharnisc/meteor-accounts-admin-ui-bootstrap-3: A roles based account management system using bootstrap 3 for Meteor*. https://github.com/hharnisc/meteor-accounts-admin-ui-bootstrap-3/, . – Accessed: 2016-03-19

[Hida] Hidayat, Ariya: *Build | PhantomJS*. http://phantomjs.org/build.html, . – Accessed: 2016-03-09

[Hidb] Hidayat, Ariya: *PhantomJS | PhantomJS*. http://phantomjs.org, . – Accessed: 2016-03-09

[HiS] HiScout GmbH: *HiScout GRC Suite Solutions Plattform für Management von IT-*

Governance, Rist und Compliance. https://www.hiscout.com, . – Accessed: 2016-03-24

[ibi] IBI SYSTEMS GMBH: *ibi systems*. http://www.ibi-systems.de, . – Accessed: 2016-03-24

[INF] INFODAS GESELLSCHAFT FÜR SYSTEMENTWICKLUNG UND INFORMATIONSVERARBEITUNG MBH: *IT-Sicherheitsdatenbank SAVe®*. https://www.infodas.de/produkte/it-sicherheitsdatenbank-save/, . – Accessed: 2016-03-24

[inta] INTARSYS CONSULTING GMBH: *Digitale Signatur mit Smartcards in Web-Anwendungen ohne Java-Plugin*. https://www.intarsys.de/produkte/sign-live/cloud-bridge, . – Accessed: 2016-02-19

[Intb] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 15489-1:2001(E)*. http://www.wgarm.net/ccarm/docs-repository/doc/doc402817.PDF, . – Accessed: 2015-12-01

[Intc] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 15489-2:2001(E)*. http://www.taoiseach.gov.ie/attached_files/Pdf%20files/30%20iso.15489-2%20-%20IRISH%20VERSION.pdf, . – Accessed: 2015-12-01

[Iroa] IRON METEOR: *Github - iron*. https://github.com/iron-meteor/iron-cli, . – Accessed: 2016-03-09

[Irob] IRON METEOR: *GitHub - Meteor Package Iron.Router*. https://github.com/iron-meteor/iron-router, . – Accessed: 2016-03-08

[isoa] *International Organization for Standardization*. http://www.iso.org/iso/home/about.htm, . – Accessed: 2015-12-01

[isob] *ISO 19005-1:2005*. http://www.iso.org/iso/catalogue_detail?csnumber=38920, . – Accessed: 2016-01-16

[Joy] JOYENT, INC.: *Node.js*. https://nodejs.org/en, . – Accessed: 2016-01-20

[JS] J. SERMERSHEIM, Ed: *RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol*. https://tools.ietf.org/html/rfc4511, . – Accessed: 2016-03-05

[KK] KELLERMANN, Max ; KOK, Auke: *ferm - for Easy Rule Making*. http://ferm.foo-projects.org, . – Accessed: 2016-03-19

[Kok] KOKOSZKA, Brenden: *GitHub - node-webshot*. https://github.com/brenden/node-webshot, . – Accessed: 2016-03-09

[Kroa] KRON, Gerhard: *Kronsoft e.K.* https://www.kronsoft.de, . – Accessed: 2015-12-11

[Krob] KRON, Gerhard: *opus i.* https://www.kronsoft.de/datenschutz/datenschutz.html, . – Accessed: 2015-12-11

[kroc] KRONSOFT E.K.: *Software und Tools für Datenschutz und ISO 27001/27002*. https://www.kronsoft.de/, . – Accessed: 2016-03-24

[Lan] LANNING, Adrian: *GitHub - alanning/meteor-roles ... with built in accounts-package*. https://github.com/alanning/meteor-roles, . – Accessed: 2016-03-09

[Lav] LAVERDET, Marcel: *GitHub - laverdet/node-fibers: Fiber/coroutine support for v8 and node*. https://github.com/laverdet/node-fibers, . – Accessed: 2016-03-19

[Liq] LIQUID, LLC: *LIQUID - Easily Collect, Analyze and Share Data*. https://getliquid.io, . – Accessed: 2016-02-19

[lrz] *Leibniz Supercomputing Center*. https://www.lrz.de, . – Accessed: 2015-12-05

[MAM+] MYERS, M. ; ANKNEY, R. ; MALPANI, A. ; GALPERIN, S. ; ADAMS, C.: *RFC 2560 - X.509 Internet Public Key Infrastructure - Online Certificate Status Protocol - OCSP*. https://tools.ietf.org/html/rfc2560, . – Accessed: 2016-03-05

[Man] Manjunath, G.: *GitHub - infindex/node-simplexml*. https://github.com/infindex/node-simplexml, . – Accessed: 2016-03-19

[Mar] Martorell, Ian: *GitHub - ianmartorell/meteor-accounts-ui-bootstrap-3 - accounts-ui package with Bootstrap 3 and localization support*. https://github.com/ianmartorell/meteor-accounts-ui-bootstrap-3, . – Accessed: 2016-03-18

[Mer80] Merkle, Ralph C.: Protocols for public key cryptosystems. In: *Proceedings of the 1980 Symposium on Security and Privacy* (1980)

[Mer00] Merkle, Ralph C.: A Digital Signature Based on a Conventional Encryption Function. In: *Advances in Cryptology — CRYPTO '87* Volume 293 of the series Lecture Notes in Computer Science (2000), S. 369–378

[Meta] Meteor Development Group: *Meteor: The JavaScript App Platform*. https://www.meteor.com, . – Accessed: 2016-01-20

[Metb] Meteor Development Group: *reactive-dict package | Atmosphere*. https://atmospherejs.com/meteor/reactive-dict, . – Accessed: 2016-03-18

[Mic] Mick, Trent: *GitHub - trentm/node-bunyan: a simple and fast JSON logging module for node.js services*. https://github.com/trentm/node-bunyan, . – Accessed: 2016-03-18

[MMPR] M'Raihi, D. ; Machani, S. ; Pei, M. ; Rydell, J.: *TOTP: Time-based One-time Password Algorithm - draft-mraihi-totp-timebased-00.txt*. http://tools.ietf.org/html/draft-mraihi-totp-timebased-00, . – Accessed: 2016-03-05

[Mona] MongoDB, Inc: *FAQ: MongoDB Fundamentals - MongoDB Manual 3.2: How does MongoDB address SQL or Query injection?* https://docs.mongodb.org/manual/faq/fundamentals/#how-does-mongodb-address-sql-or-query-injection, . – Accessed: 2016-03-24

[Monb] MongoDB, Inc.: *A GIANT LEAP*. https://www.mongodb.org, . – Accessed: 2016-01-20

[Monc] MongoDB, Inc: *Security Checklist — MongoDB Manual 3.2*. https://docs.mongodb.org/manual/administration/security-checklist/, . – Accessed: 2016-03-19

[Mor] Moreau, Cédric: *GitHub - c-geek/merkle: Node.js module implementing Merkle tree algorithm*. https://github.com/c-geek/merkle, . – Accessed: 2016-03-18

[npm] *npm*. https://www.npmjs.com, . – Accessed: 2016-02-19

[oeca] *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. http://www.oecd.org/sti/ieconomy/oecdguidelinesontheprotectionofprivacyandtransborderflowsofpersonaldata.htm, . – Accessed: 2015-12-05

[oecb] *Organization for Economic Co-operation and Development*. http://www.oecd.org, . – Accessed: 2015-12-05

[OT] Otto, Mark ; Thornton, Jacob: *Bootstrap*. http://getbootstrap.com/, . – Accessed: 2016-03-09

[owi] *Gesetz über Ordnungswidrigkeiten (OWiG)*. http://www.gesetze-im-internet.de/bundesrecht/owig_1968/gesamt.pdf, . – Accessed: 2016-01-16

[Pat] Patalong, Frank: *Urteil gegen Deutschland: Europa befreit Datenschützer von politischem Druck*. http://www.spiegel.de/netzwelt/netzpolitik/urteil-gegen-deutschland-europa-befreit-datenschuetzer-von-politischem-druck-a-682540.html, . – Accessed: 2015-12-11

[Pug] Pugliese, Gabriel H.: *CodersTV - I want to watch videos about any language.* http://coderstv.com, . – Accessed: 2016-02-19

[Reg] *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung.* http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/QES/Veroeffentlichungen/Algorithmen/2016Algorithmenkatalog.pdf, . – Accessed: 2016-01-16

[REI] REINER Kartengeräte GmbH & Co. KG: *Reiner SCT Shop.* https://www.chipkartenleser-shop.de/shop/cert_bdr, . – Accessed: 2016-02-18

[Res] Respondly Inc.: *respondly - The Fastest Way to Tackle Twitter as a Team.* https://respond.ly, . – Accessed: 2016-02-19

[Ros15] Rossi, Ben: *New EU data law's go-live date finally revealed – and why its costs will run into the billions.* http://www.information-age.com/technology/information-management/123459991/new-eu-data-laws-go-live-date-finally-revealed-and-why-its-costs-will-run-billions, August 2015. – Accessed: 2015-12-04

[RSA] RSA: *RSA SECURID.* https://www.rsa.com/de-de/products-services/identity-access-management/securid, . – Accessed: 2016-03-05

[Scha] Schmerler, Peter: *Voice Of Information.* http://www.voi.de, . – Accessed: 2015-12-11

[Schb] Schwarz, Nina: *Was war die Stasi?* http://www.bstu.bund.de/DE/Wissen/Bildung/Einstieg/_node.html, . – Accessed: 2015-10-25

[Sec] SecCommerce Informationssysteme GmbH: *SecSigner.* https://www.seccommerce.de/secsigner, . – Accessed: 2016-02-19

[Ser] SerNet GmbH: *OpenSource Informationssicherheit | verinice.org.* http://verinice.org, . – Accessed: 2016-03-24

[sha] *FIPS PUB 180-4.* http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, . – Accessed: 2016-01-14

[siga] *Signatur Gesetz.* http://www.gesetze-im-internet.de/sigg_2001/, . – Accessed: 2015-12-03

[sigb] *Signatur Verordnung.* http://www.gesetze-im-internet.de/sigv_2001/, . – Accessed: 2016-01-09

[SSDG] Secure Solutions Deutschland GmbH exceet: *SIGNAMUS - Abrechnung und Preise.* https://www.signamus.com/de/funktion-leistung/abrechnung-preise.html, . – Accessed: 2016-02-19

[Sta] StackStorm: *StackStorm | Event-driven automation.* https://stackstorm.com, . – Accessed: 2016-03-16

[Swa] Swapp, Ryan: *GitHub - node-webshot.* https://medium.com/@ryanswapp/getting-started-with-meteor-and-react-d411a87a1674, . – Accessed: 2016-03-09

[T-S] T-Systems International GmbH: *TeleSec - Trust Center der Deutschen Telekom AG.* https://www.telesec.de/, . – Accessed: 2016-03-19

[TB] Troan, Erik ; Brown, Preston: *logrotate.* http://www.linuxcommand.org/man_pages/logrotate8.html, . – Accessed: 2016-03-18

[Thea] The FreeBSD Project: *17.6. Creating and using DVD Media.* https://www.freebsd.org/doc/handbook/creating-dvds.html, . – Accessed: 2016-03-20

[Theb] The Unicode Consortium: *The Unicode Standard.* http://www.unicode.org/versions/latest, . – Accessed: 2016-01-20

[Thy] Thylmann, Marc ; Bundesdruckerei GmbH (Hrsg.): *Pressemitteilung: Bundesdruckerei treibt die Online-Unterschrift mit dem neuen Personalausweis voran.* https://www.bundesdruckerei.de/de/2149-bundesdruckerei-treibt-die-online-unterschrift-mit-dem-neuen-personalausweis-voran, . – Accessed: 2016-02-18

[Typ] Typaldos, Eric: *GitHub - Meteor Package accounts-ldap.* https://github.com/typ90/meteor-accounts-ldap/, . – Accessed: 2016-03-07

[Unia] Universitä Regensburg: *Verfahrensbeschreibungen.* http://www.uni-regensburg.de/sprache-literatur-kultur/fakultaet/verfahrensbeschreibungen/index.html, . – Accessed: 2015-12-11

[Unib] Universität Regensburg: *Dekanat der Universität Regensburg.* http://www.uni-regensburg.de/sprache-literatur-kultur/fakultaet/dekanat/index.html, . – Accessed: 2015-12-11

[Uru] Urushima, Kenji: *jsrsasign - cryptography library in JavaScript.* https://www.npmjs.com/package/jsrsasign, . – Accessed: 2016-03-18

[Ver] Verein zur Förderung eines Deutschen Forschungsnetzes e. V.: *DFN-PKI: Zeitstempeldienst.* https://www.pki.dfn.de/zeitstempeldienst/, . – Accessed: 2016-03-19

[Vet] Vetter, Reinhard: *Summary of the essentials of the Bavarian State Law for Data Protection and Privacy.* https://www.datenschutz-bayern.de/recht/baydsgsum.htm, . – Accessed: 2015-11-08

[VOI] VOI: *Merksätze des VOI zur revisionssicheren elektronischen Archivierung.* http://www.ulshoefer.de/voi_merksaetze_der_archivierung.pdf, . – Accessed: 2015-12-05

[WMC] WMC GmbH: *GRC & ISMS Software.* http://grc-qsec.com/qsec/software, . – Accessed: 2016-03-24

[Wula] Wulff, Fiete: *Aktuell tätiger Zertifizierungsdiensteanbieter: Deutsche Telekom AG.* http://www.bundesnetzagentur.de/cln_1422/DE/Service-Funktionen/QualifizierteelektronischeSignatur/WelcheAufgabenhatdieBundesnetzagentur/AufsichtundAkkreditierungvonAnbietern/Deutsche_Telekom_AG_Basepage.html, . – Accessed: 2016-03-07

[Wulb] Wulff, Fiete: *Zertifizierungsdienstanbieter.* http://www.bundesnetzagentur.de/cln_1932/DE/Service-Funktionen/QualifizierteelektronischeSignatur/WelcheAufgabenhatdieBundesnetzagentur/AufsichtundAkkreditierungvonAnbietern/ZertifizierungsDiensteAnbietr_node.html, . – Accessed: 2016-02-18

[x50] *X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.* http://www.itu.int/rec/T-REC-X.509/en, . – Accessed: 2016-01-15

[zpo] *Zivilprozessordnung.* http://www.gesetze-im-internet.de/zpo/index.html, . – Accessed: 08.01.2016