universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Design and Implementation of a Prototype for Enhanced Self-Protection against Unauthorized Access"

verfasst von / submitted by

Thomas Wesenauer, BSc

gemeinsam mit / in collaboration with

Patrik Pollak, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik Master's degree programme Business Informatics
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. DDr. Gerald Quirchmayr

Eidesstattliche Erklärung

Hiermit erklären wir eidesstattlich, die vorliegende Arbeit selbstständig und ohne Benutzung anderer, als der angegebenen Quellen und Hilfsmittel verfasst zu haben. Die aus fremden Quellen übernommenen direkten oder indirekt Gedanken, Konzepte und Zitate sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher oder ähnlicher Form verfasst, noch einer anderen Prüfungsbehörde vorgelegt.

Wien, 03. September 2018

Unterschrift

Patrik Pollak

Thomas Wesenauer

1 EINFÜHRUNG 9

1.1 HINTERGRUND UND MOTIVATION 9
1.2 ZIELSETZUNG UND ERWARTETE ERGEBNISSE 10
1.3 AUFBAU DER ARBEIT UND ARBEITSTEILUNG 10

2 ANWENDUNGSFALL 11

2.1 SZENARIO 12
2.2 AKTEURE 13
2.3 BEISPIEL FÜR EINE EINSTWEILIGE VERFÜGUNG 14
2.4 USE-CASE-DIAGRAMM 16
2.5 ABLAUF EINES TYPISCHEN ARBEITSGANGES 17
2.6 EXEMPLARISCHE DARSTELLUNG 19

3 STATE OF THE ART UND HERAUSFORDERUNGEN 26

3.1 SYSTEM-LEVEL-ACCESS-CONTROL 27
3.2 DE-PERIMETERISATION 27
3.3 BESTEHENDE DE-PERIMETERISATION-PROTECTION-ANSÄTZE 28
3.3.1 DIGITAL-RIGHTS-MANAGEMENT 29
3.3.2 CREATIVE-COMMONS-RIGHTS-EXPRESSION-LANGUAGE 30
3.3.3 DAS SPIDER-PROJEKT 31
3.4 VERGLEICHSMATRIX 32
3.5 ABGRENZUNG ZU BLOCKCHAINS 34

4 IMPLEMENTIERUNGSUMFELD 36

4.1 UNIFORM-RESOURCE-IDENTIFIER 37
4.2 HTTP/1.1 – HYPERTEXT TRANSFER PROTOCOL 38
4.3 WEBSERVICEARCHITEKTUREN 41
4.3.1 SERVICEORIENTIERTE ARCHITEKTUREN 41
4.3.2 SOAP – SIMPLE-OBJECT-ACCESS-PROTOCOL 43
4.3.3 RESTFUL WEBSERVICES – REPRESENTATIONAL-STATE-TRANSFER 44

5 SECURITY-MANAGEMENT 49

5.1 Symmetrische Verschlüsselung 49
5.1.1 Blockchiffren 51
5.1.2 Stromchiffren 57
5.2 Secure-Hash-Funktionen 58
5.3 Asymmetrische Verschlüsselung 61
5.3.1 Public-Key-Verschlüsselung 61
5.3.2 Asymmetrische Verschlüsselungsverfahren 65
5.4 Secure Sockets Layer (SSL) und Transport Layer Security (TLS) 66
5.5 HTTPS (HTTP over SSL) 71

6 SYSTEMKONZEPTION 73

6.1 Grundlegende Idee 74
6.2 Client 74
6.3 Service 74
6.4 Datenhaltung 75
6.5 Sicherheitsmaßnahmen 75

7 IMPLEMENTIERUNGSANSATZ 77

7.1 JavaFX 78
7.2 Digital Envelope 80
7.3 De-Perimeterisation 82
7.4 Security-Management und Continous-Access-Control 83
7.4.1 Identitätsmanagement – Was ist X.509? 84
7.4.2 Multiple-Key-Management mittels Extended X.509 90
7.4.3 Gruppenrichtlinien und Gültigkeitsrahmen 92
7.4.4 Single-Session-Security 94
7.4.5 Layer-Security 95
7.5 Das ISAC-Zertifikat 96
7.6 Konkreter Umsetzungsansatz 100

8 SYSTEMARCHITEKTUR UND -KOMPONENTEN 104

8.1 Architektur 105

8.1.1 CLIENT 106
8.1.2 SERVER 107
8.1.3 DATABASE 109
8.1.4 DATENTRANSFER 110
8.2 SYSTEMKOMPONENTEN 111
8.2.1 SERVER 112
8.2.2 CLIENT 115

9 PROZESSBESCHREIBUNG 117

9.1 NEUE DATEI SPEICHERN 120
9.2 DATEI LADEN 124
9.3 BESTEHENDE DATEI SPEICHERN 127
9.4 USER ERSTELLEN 131
9.5 RECHTE VERGEBEN UND VERWALTEN 136

10 REALISIERUNG 138

10.1 KLASSENDIAGRAMM ÜBERBLICK 138
10.1.1 SERVER 139
10.1.2 CLIENT 142
10.2 DATENMODELL 145
10.3 SCHNITTSTELLENDEFINITION UND –BESCHREIBUNG 148
10.3.1 DATENTRANSFER-FORMAT 149
10.3.2 DATENSTRUKTUR 149
10.3.3 SERVICE-ENDPUNKTE UND RESSOURCEN 150

11 IMPLEMENTIERUNGSBESCHREIBUNG 163

11.1 ENTWICKLUNGSUMGEBUNG 163
11.2 IMPLEMENTIERUNG DES LAYOUTS 164
11.3 IMPLEMENTIERUNG DER ZERTIFIKATSGENERIERUNG 167
11.3.1 CLIENT – ISAC REQUEST ABSETZEN (.CSR) 167
11.3.2 SERVER – ISAC-ZERTIFIKAT FÜR USER/FILE KREIEREN 175

12 INTERFACEBESCHREIBUNG 182

12.1 ÜBERBLICK 182
12.2 BEDIENELEMENTE 183
12.2.1 SHORTCUTS 185
12.3 PROZESS „LADEN EINER DATEI“ 185
12.3.1 LADEDIALOG 186
12.3.2 FILEBROWSER 186
12.3.3 POLICY-EDITOR 187
12.3.4 GRUPPEN- UND ZUGRIFFSRECHTEÜBERSICHT 188
12.3.5 GRUPPENERSTELLUNG 189
12.3.6 BENUTZERAUSWAHL 190
12.3.7 GRUPPEN- UND ZUGRIFFSRECHTEÜBERSICHT 191

13 EINORDNUNG UND DISKUSSION DER ERGEBNISSE 191

13.1 UMSETZUNGSPHASEN 193
13.2 OFFENE FRAGEN UND AUSBLICK 196

14 ZUSAMMENFASSUNG 198

1 Einführung

1.1 Hintergrund und Motivation

Sicherheit ist nach Maslow[1] ein Grundbedürfnis der Menschen. Gerade in der heutigen Zeit besitzt dieser Aspekt in Hinblick auf die Sicherheit in Informationssystemen mehr Aktualität denn je. Dies zeigt sich in täglich neuen Nachrichten über Datendiebstähle, Hackerangriffe, unsichere Datenbanken und irrtümliche Offenlegungen von teils Millionen von Nutzerdaten. Aus diesem Grund beschäftigt sich die Informatik auch mit der Forschung und Weiterentwicklung von Sicherheitsmechanismen und Ideen, die den täglichen Informationsaustausch sicherer und zugleich einfacher gestalten sollen. Das Ziel der Bestrebungen ist die höchstmögliche Sicherheit bei einfacher Handhabung, wobei der Schutz der Privatsphäre im Zusammenhang mit einem modernen Zugriffsmanagement schon immer eine Herausforderung im Bereich der digitalen Kommunikation war.

Ein Blick in die jüngere Vergangenheit zeigt bereits ein leicht differenziertes Bild des Sicherheitsgedankens. Anfang der 1990er Jahre stellte sich im Bereich der Informationssicherheit mit Aufkommen des allgemein nutzbaren Internets Handlungsbedarf ein. Waren komplizierte Verschlüsselungsalgorithmen und Textchiffrierung bis dahin fast ausschließlich für Firmen oder im Bereich der Forschung relevant, so wurden nun auch sichere Protokolle und Mechanismen zum Datenaustausch bei privaten Anwendern benötigt. Diese Entwicklungen wurden durch immer bessere und vor allem weiter verbreitete Geräte für Privatanwender gestützt, wodurch die Nutzung komplizierter Verschlüsselungsverfahren überhaupt erst ermöglicht wurde. Dies könnte also als Beginn der modernen IT-Sicherheit und damit als Fundament der vorliegenden Arbeit angesehen werden.

[1] Maslow, A. H. (1943). A theory of human motivation. Psychological Review, 50(4), 370–396.

1.2 Zielsetzung und erwartete Ergebnisse

Es ist nicht immer problemlos, ein sicheres Textverarbeitungs-System zu schaffen, das modernen Sicherheitsstandards genügt und dabei gleichzeitig seinen Benutzern ein größtmögliches Maß an Funktionsumfang und Bearbeitungsoptionen bietet. Je mehr Features ein System hat, umso mehr Fehler können dabei auftreten. Nachfolgend sollen die Ziele und Forschungsfrage eruiert und evaluiert werden.

Die wissenschaftliche Frage hinter dieser Arbeit ist, ob das eSprint-Konzept technisch umsetzbar ist. Das eSprint-System basiert auf dem so genannten Self-Protecting-Information-for-De-Perimeterised-Electronic-Relationship-System oder kurz SPIDER-Projekt. Hierbei handelt es sich lediglich um eine Idee mit Mock-ups[2] – jedoch existierte hierfür nie ein funktionstüchtiges Tool. Aus diesem Grund war anfangs nicht klar, ob sich alle geplanten Sicherheitsmaßnahmen – Stichwort ‚Extended x.509' oder ‚Multiple-Keys' – technisch umsetzen lassen. Kurz gesagt ist es die Aufgabe dieser Masterarbeit, das Proof of Concept der von Frau Nisreen Alam Aldeen in ihrer Arbeit zur AsiaAres Konferenz 2013 entwickelten Idee, auf seine technische Machbarkeit zu prüfen und eine passende Systemlandschaft hierfür zu entwickeln. Eine detaillierte Analyse der zu erfüllenden Umsetzungskriterien wird im Kapitel 3 genauer beleuchtet und ähnlichen Technologien vergleichend gegenübergestellt [BHT09, ARE13].

1.3 Aufbau der Arbeit und Arbeitsteilung

Diese Arbeit wird sich kurz mit dem Thema Sicherheit im Allgemeinen auseinandersetzen. Welche bekannten Mechanismen zur Sicherung von Informationen gibt es? Worin liegen die Schwächen dieser Systeme und wobei gibt es Verbesserungsmöglichkeiten? Was sind System-Level-Sicherheitsmechanismen und warum sind diese oft nicht ausreichend für manche Einsatzgebiete?

Das Kernthema dieser Arbeit setzt sich danach mit einer bestehenden Arbeit von der Universität Cardiff auseinander – dem SPIDER Projekt. Dieser Ansatz diente als Grundlage, um ein in Zusammenarbeit mit Frau Nisreen Alam Aldeen erdachtes und

[2] Eine nicht funktionsfähige (grafische) Darstellung einer Anwendung

umgesetztes Werkzeug zum sicheren Informationsaustausch zu entwickeln – das Enhanced-Self-Protecting-Information-Technology oder auch kurz eSprint-System. Unter Zuhilfenahme moderner Sicherheitsmechanismen wie x.509-Zertifikaten, AES-Verschlüsselung, sicheren Hyper-Text-Tunneln und durchdachten Austauschmechanismen, basierend auf Java, JavaScript, JavaFX und HTML5, soll diese neu entwickelte Systemarchitektur auf den Ideen ihres geistigen Vorgängers (SPIDER) aufbauen und zu verbessern. Hierfür wurden von Grund auf sowohl Server als auch Client-Applikationen implementiert, die einzig auf die Gewährleistung von de-perimeterisierter Informationssicherheit hin entwickelt wurden.

Nach der detaillierten Erläuterung der Umsetzungsphasen, verwendeten Technologien, dahinterliegenden Ideen und praktischen Umsetzung mit Darstellung markanter Code-Teile soll auch ein praktisches Einsatzbeispiel gezeigt werden. Dieses soll die Anwendbarkeit der Applikation und zudem ein mögliches Anwendungsszenario demonstrieren.

Abschließend wird kritisch hinterfragt, ob und wie die entwickelten Ideen in der Zukunft weitergeführt werden könnten und wo es Verbesserungspotenzial gibt, vor allem durch den Fortschritt bei technischen Umsetzungsmöglichkeiten.

Prinzipiell wurde die technische Umsetzung dieser Arbeit in die Bereiche Server (bearbeitet durch Thomas Wesenauer) und Client (bearbeitet durch Patrik Pollak) unterteilt. Eine detaillierte Aufschlüsselung der gesamten Arbeit befindet sich im Annex.

2 Anwendungsfall

Um einen Bezug zu den in den nachfolgenden Kapiteln vorgestellten Funktionalitäten und Anforderungen des eSprint-Systems herstellen zu können, wird zunächst ein möglicher Anwendungsfall näher betrachtet. Das Use-Case-Diagramm beziehungsweise dessen Beschreibung soll einen realitätsnahen Überblick darüber verschaffen, wofür die entwickelte Softwarelösung eingesetzt werden kann. Hierdurch sollen die wichtigsten Funktionen der Software herausgearbeitet und in Beziehung zu einem realen Kontext gesetzt werden.

2.1 Szenario

Für diesen Zweck wurde ein Anwendungsszenario im juristischen Umfeld gewählt, bei dem es sich um die Erstellung und Weiterleitung einer einstweiligen Verfügung (EV) durch den zuständigen Richter an die beteiligten Anwaltskanzleien handelt. Der hypothetische Sachverhalt zur in Abbildung 1 dargestellten EV lautet wie folgt.

In diesem Szenario lernen sich Alice Alba und Bob Brauer über eine Dating-App kennen. Da Alice nach einem ersten Treffen jedoch kein Interesse an einer tiefergehenden Bekanntschaft hat, möchte sie Bob nicht wiedersehen. Trotz der ablehnenden Haltung von Alice versucht er, sie innerhalb von 2 Monaten 893-mal sowohl persönlich als auch telefonisch sowie mithilfe des Messenger-Dienstes WhatsApp zu erreichen. Nachdem Bob Brauer am Arbeitsplatz von Alice Alba erscheint und sie beharrlich verfolgt, entschließt sich Alice dazu, eine einstweilige Verfügung gegen Bob zu erwirken, die ihm jedwede Kontaktaufnahme untersagt.

Während des nachfolgend beschriebenen Prozesses haben mehrere Personen – oder allgemeiner formuliert Akteure – auf die einstweilige Verfügung Zugriff. Jedoch sollen für jeden der Beteiligten jeweils andere Teile einseh- beziehungsweise bearbeitbar sein. Hierbei bietet sich der Einsatz von eSprint an.

Grob umrissen sind an diesem Prozess insgesamt sieben Rollen (Akteure) beteiligt: Der Richter, der die einstweilige Verfügung schreibt, die Gerichtskanzleisekretärin, die sich um die Weiterleitung der Dokumente und die Kontrolle der Fristen kümmert, die Sekretärinnen der Anwaltskanzleien und die beiden Anwälte beziehungsweise ein Konzipient einer der beiden Anwaltskanzleien. In der aktuellen Situation kann der Austausch einer einstweiligen Verfügung zwischen dem zuständigen Richter und den beteiligten Parteien bereits über den elektronischen Weg stattfinden, wobei jedoch zu jeder Zeit jede beteiligte Partei die einstweilige Verfügung einsehen kann und somit gegebenenfalls auch mehr Informationen als notwendig erhält. An diesem Punkt könnte durch den zusätzlichen Einsatz von eSprint eine deutliche Verbesserung sowohl der Informationssicherheit als auch der Informationsverteilung erzielt werden.

2.2 Akteure

Wie bereits erwähnt, existieren in diesem Szenario sieben verschiedene Rollen (Richter, Gerichtssekretärin, Anwaltssekretärinnen und -sekretäre, Anwälte, Konzipient), für die jeweils unterschiedliche Teile der einstweiligen Verfügung interessant sind.

Der Richter erstellt das Dokument beziehungsweise schreibt die einstweilige Verfügung und darf somit alle Teile sowohl einsehen als auch bearbeiten. Nachdem er das Dokument an seine Gerichtskanzleisekretärin weitergeleitet hat, benötigt diese zunächst die erste Seite des Dokuments. Diese enthält die so genannte Gerichtszahl (GZ), worüber sich die einstweilige Verfügung einem eindeutigen Akt zuordnen lässt. Die GZ muss von der Sekretärin lediglich eingesehen, aber nicht verändert werden können. Außerdem ist für sie die Verfügung des Richters relevant, die sich am Ende des Dokuments befindet und notwendige Informationen bezüglich der Parteien, an die das Schriftstück weitergeleitet werden muss sowie über die einzuhaltenden Rechtsmittelfristen enthält. Diese Informationen sind ebenfalls nur einzusehen und dürfen keinesfalls manipulierbar sein.

Ausgehend von einer elektronischen Übermittlung der einstweiligen Verfügung an zwei Anwaltskanzleien (Denkbar wäre auch eine postalische Zustellung, die jedoch außerhalb des Fokus dieses Beispiels ist.) teilt sich nun das betrachtete Umfeld dahingehend, dass jeweils eine identische Kopie des Dokuments an alle beteiligten Parteien gesendet wird. Die nachfolgenden Ausführungen beziehen sich somit immer auf eine Partei und laufen identisch bei gegnerischen beziehungsweise allen anderen in dieser Gerichtssache beteiligten Parteien ab.

In der jeweiligen Anwaltskanzlei empfängt zunächst die Rechtsanwaltssekretärin das gesamte Dokument. Diese darf die Geschäftszahl, die Namen der beteiligten Parteien sowie die Rechtsmittelbelehrung einsehen, um die ihr obliegenden Tätigkeiten durchführen zu können. Hierzu zählt beispielsweise die Berechnung des Endes der Rechtsmittelfrist, die ab Erhalt des Dokuments zu laufen beginnt. Die Rechtsanwaltssekretärin kann die Frist am Ende des Dokuments einfügen, sofern dies durch den Rechtsanwalt (RA) erwünscht wird. Danach gibt sie das Dokument für den zuständigen RA beziehungsweise Konzipienten frei.

2.3 Beispiel für eine einstweilige Verfügung

REPUBLIK ÖSTERREICH
BEZIRKSGERICHT INNERE STADT

11 C 1/19a
(Bitte in allen Eingaben anführen)

Marxergasse 1A,
1030 Wien

Tel.: + 43 1 515 28 0

EINSTWEILIGE VERFÜGUNG

RECHTSSACHE:

gefährdete Partei/en:
Alice Alba
Kärntner Straße 1/ 4. Stock/ Tür 10
1010 Wien

vertreten durch:
Kanzlei Recht & Haben
Rotenturmstraße 13
1010 Wien

Gegner der gefährdeten Partei/en:
Bob Brauer
Herrengasse 17 /3. Stock/ Tür 6
1010 Wien

vertreten durch:
Kanzlei Gewissen & Los
Schottenbastei 10-16
1010 Wien

Wegen:
§ 382g EO

1.) Dem Antragsgegner werden die persönliche Kontaktaufnahme und Verfolgung der gefährdeten Partei sowie auch die briefliche, telefonische oder sonstige Kontaktaufnahme verboten.

2.) Dem Antragsgegner wird verboten, einen Dritten zur Aufnahme von Kontakten mit der gefährdeten Person zu veranlassen.

3.) Die Sicherheitsbehörden werden mit dem Vollzug der Punkte 1.) und 2.) dieser einstweiligen Verfügung jeweils auf Ersuchen der Antragstellerin beauftragt.

Diese einstweilige Verfügung gilt bis zum 03.04.2019.

BEGRÜNDUNG:

Am 03.04.2018 beantragte die gefährdete Partei im Wesentlichen wie aus dem Spruch ersichtlich und brachte vor, dass sich beide Parteien am 14.02.2018 auf der Dating-Plattform Tinder kennengelernt haben. Nach dem erstmaligen und einzigen durch die nunmehr gefährdete Partei tatsächlich gewollten Treffen am 16.02.2018 in der Wohnung des Antragsgegners gab die Antragstellerin jedoch gegenüber dem Antragsgegner unmissverständlich an, kein Interesse an einer weiteren Vertiefung der Bekanntschaft zu haben.

Abbildung 1: einstweilige Verfügung – Teil 1

Der Antragsgegner versuchte dennoch, die gefährdete Partei im Zeitraum vom 20.02.2018 bis 01.03.2018 sowohl telefonisch als auch mit Hilfe des Messengerdienstes WhatsApp laut vorgelegter Unterlagen 261-mal zu erreichen. Nachdem diese Versuche der Kontaktaufnahme jedoch von Seiten der Antragstellerin unbeantwortet geblieben sind, suchte der Antragsgegner am 02.03.2018 den Arbeitsplatz der gefährdeten Partei gegen 16:30 Uhr auf und wartete dort deren Dienstschluss ab. Der Antragsgegner versuchte in weiterer Folge, persönlich mit der gefährdeten Partei ein Gespräch zu beginnen und folgte dieser beharrlich. Die Antragstellerin flüchtete daraufhin in eine nahe gelegene Polizeidienststelle. Die dort um Hilfe ersuchten Beamten konnten den Antragsgegner jedoch nicht mehr auffinden.

Aus Sicherheitsgründen wohnt die Antragstellerin zurzeit bei ihrer Freundin und Arbeitskollegin Frau Carina Cook in der Schönlaterngasse 7, 1010 Wien, um den Antragsgegner nicht allein antreffen zu müssen.

Der Antragsgegner versuchte in weiterer Folge, die gefährdete Partei selbst oder durch unterschiedliche, mit ihm befreundete Personen, im Zeitraum vom 06.03.2018 bis 30.03.2018 insgesamt 632-mal abermals telefonisch oder mittels des Messengerdienstes WhatsApp zu erreichen.

Rechtlich ist dazu auszuführen, dass gemäß § 382g EO das Gericht einer Person, die eine andere durch andauernde und unerwünschte persönliche, telefonische, briefliche oder sonstige Kontaktaufnahme gefährdet, jedwede Kontaktaufnahme zur gefährdeten Person untersagen kann.

Die erforderlichen Voraussetzungen für die Erlassung der beantragten einstweiligen Verfügung gemäß § 382g EO sind bescheinigt und gemäß § 382g EO war die einstweilige Verfügung daher antragsgemäß bis zum 03.04.2019 zu erlassen. Gemäß § 382g Abs. 3 EO waren die Sicherheitsbehörden damit zu beauftragen, die Entscheidung spruchgemäß zu vollziehen.

Gegen die Bewilligung der einstweiligen Verfügung kann der Gegner der gefährdeten Partei gemäß § 397 EO Widerspruch erheben. Der Widerspruch muss innerhalb von 14 Tagen nach Zustellung des Beschlusses bei dem Gericht erhoben werden, bei dem der Antrag auf Bewilligung der einstweiligen Verfügung angebracht wurde. Durch die Erhebung des Widerspruches wird die Vollziehung der getroffenen Verfügung nicht gehemmt.

Bezirksgericht Innere Stadt Wien, Abteilung 11
Wien, 03. April 2018
Dr. Oscar Order, Richter

Elektronische Ausfertigung
Gemäß § 79 GOG

Abbildung 2: Einstweilige Verfügung – Teil 2

2.4 Use-Case-Diagramm

Anhand dieser Erörterungen lassen sich nun die Rollen und deren Hauptaufgabengebiete während des Prozessablaufs mittels des nachfolgenden Use-Case-Diagramms in Abbildung 3 darstellen.

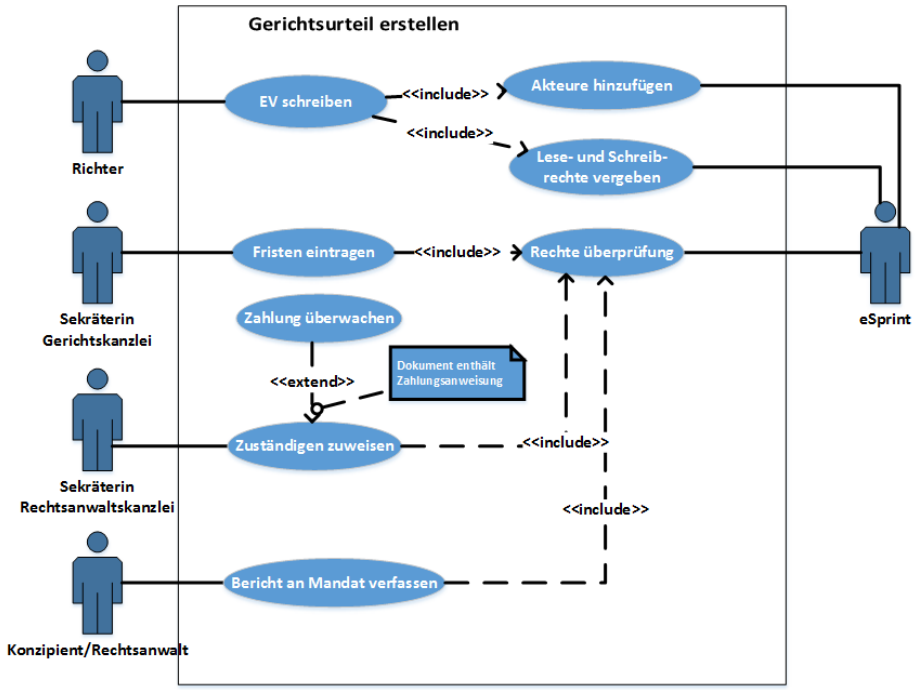

Abbildung 3: Anwendungsfall – Übersicht

2.5 Ablauf eines typischen Arbeitsganges

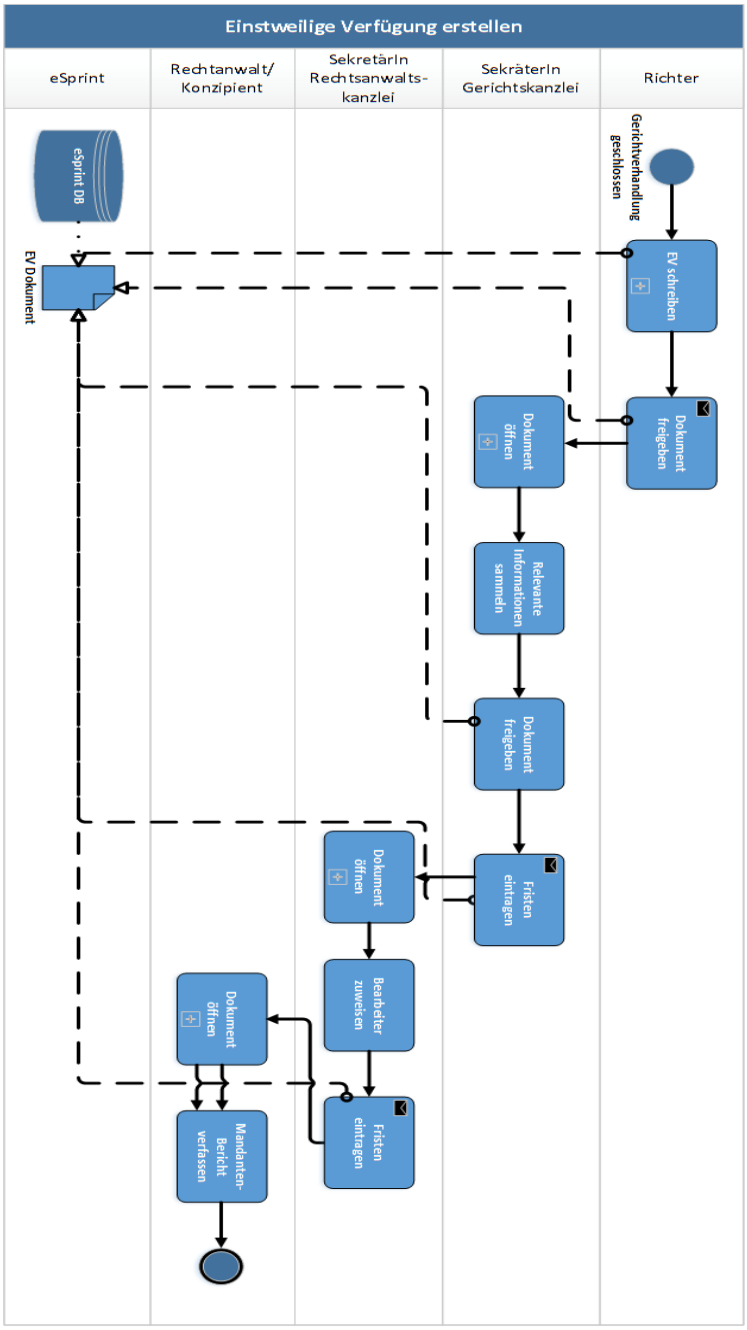

Abbildung 4: Anwendungsfall – einstweilige Verfügung erstellen

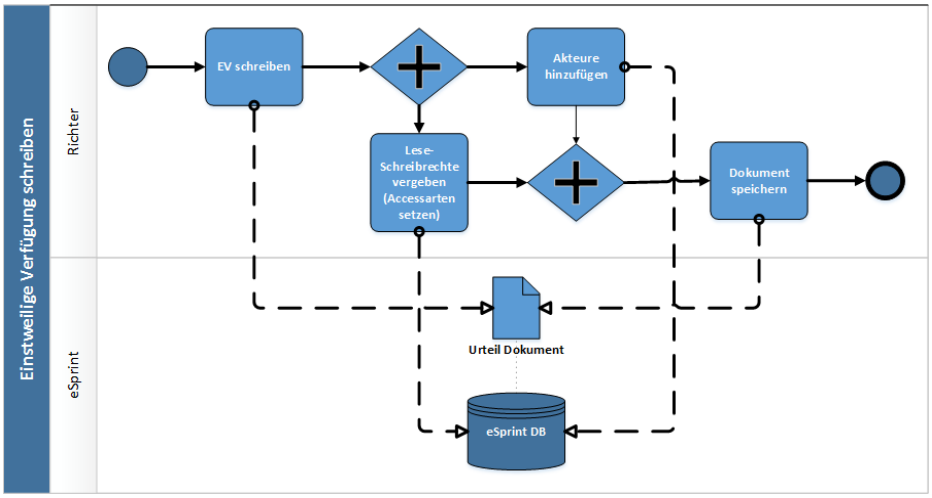

Abbildung 5: Anwendungsfall – einstweilige Verfügung schreiben

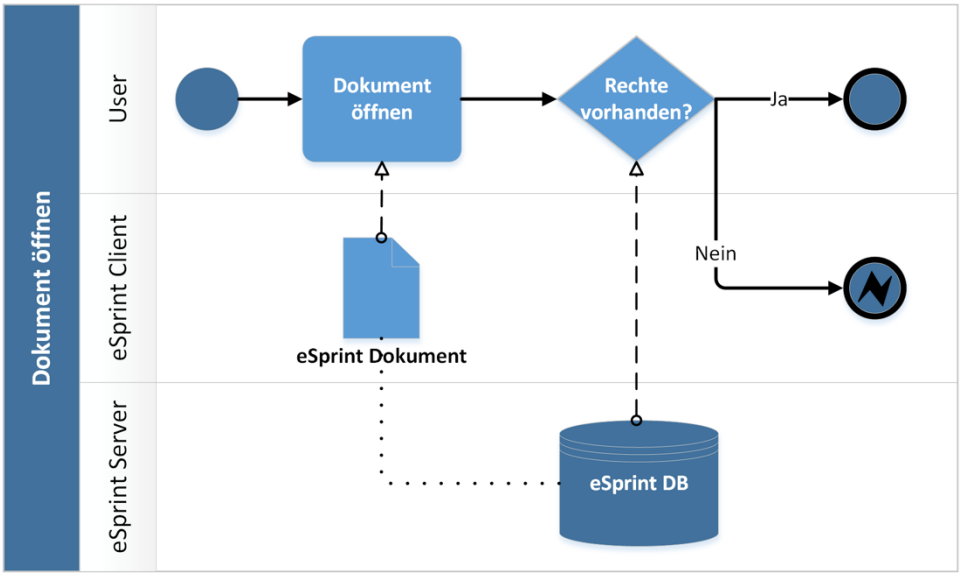

Abbildung 6: Anwendungsfall – Dokument öffnen

2.6 Exemplarische Darstellung

Hier folgt eine Darstellung der verschiedenen Sichtweisen der sieben beteiligten Akteure anhand des eSprint-Clients. Bei den Akteuren handelt es sich um:

- *den zuständigen Richter:* **Dr. Oscar Order**
- *die Gerichtssekretärin:* **Christina Colber**
- *die Sekretärin der Rechtsanwaltskanzlei der Antragstellerin:* **Daisey Damberger**
- *der Rechtsanwalt der Antragstellerin:* **Mag. Richard Recht**
- *der Sekretär der Rechtsanwaltskanzlei des Antragsgegners:* **Franz Frust**
- *der Rechtsanwalt des Antragsgegners:* **Mag. Leopold Los**
- *eine Konzipientin einer der beiden Rechtsanwaltskanzleien:* **Emily Eschelboeck**

Wie den unten dargestellten Abbildungen entnommen werden kann, ist der jeweils nicht einsichtige Teil bei den beteiligten Parteien geschwärzt und somit nicht bearbeitbar.

Die Aufteilung der Berechtigungen sieht dabei wie folgt aus:

- *Restricted Access*

  Bei den Daten, die mittels „Restricted Access" gekennzeichnet werden, handelt es sich um personenbezogene sensible Daten, die nur von einer Partei gesehen werden dürfen. **Akteure**: *Richter Dr. Oscar Order, Anwalt Mag. Richard Recht*
- *Organisation Access*

  Daten, die mit der Zugriffseinschränkung „Organisation Access" gekennzeichnet wurden, beziehen sich auf fallrelevante Informationen. Diese sollen lediglich von den Anwälten beider Parteien, dem Richter selbst und dem Konzipienten, nicht aber von den Sekretärinnen der beiden Anwälte eingesehen werden können. **Akteure**: *Richter Dr. Oscar Order, Anwalt Mag. Richard Recht, Anwalt Mag. Leopold Los*
- *Community Access*

  Bei mit „Community Access" markierten Parts handelt es sich um personenbezogene Informationen, die für den Konzipienten keine Rolle spielen und somit nicht von ihm eingesehen werden dürfen.

- *Open Access*

  Zuletzt gibt es die Möglichkeit der Vergabe eines „Open Access-“ Zugriffsschutzes, der jedoch im Zuge dieses Fallbeispiels nicht benötigt wurde.

Die zugehörige Berechtigungsmatrix sieht dabei wie folgt aus:

Akteur	Open Access	Community Access	Organisation Access	Restricted Access
Dr. Oscar Order (Richter)	●	●	●	●
Christina Colber (Sekretärin Gericht)	●	●		
Daisey Damberger (Sekretärin Anwalt)	●	●		
Mag. Leopold Los (Anwalt Antragsgegner)	●	●	●	
Franz Frust (Sekretär Anwalt)	●	●		
Mag. Richard Recht (Anwalt Antragstellerin)	●	●	●	●
Emily Eschenboeck (Konzipientin)	●		●	

Tabelle 1: Exemplarische Übersicht zu Gruppenrechten

Die hierzu passende Darstellung der Gruppen im eSprint-System sieht wie in der unten gezeigten Abbildung 7 aus. Da sich die Darstellungen der jeweiligen Sekretärinnen und Sekretären, wie aus Tabelle 1 ersichtlich, jedoch nicht voneinander unterscheiden würden, wurde für dieses Beispiel nur eine exemplarische Sekretärin einer Gruppe zugeordnet.

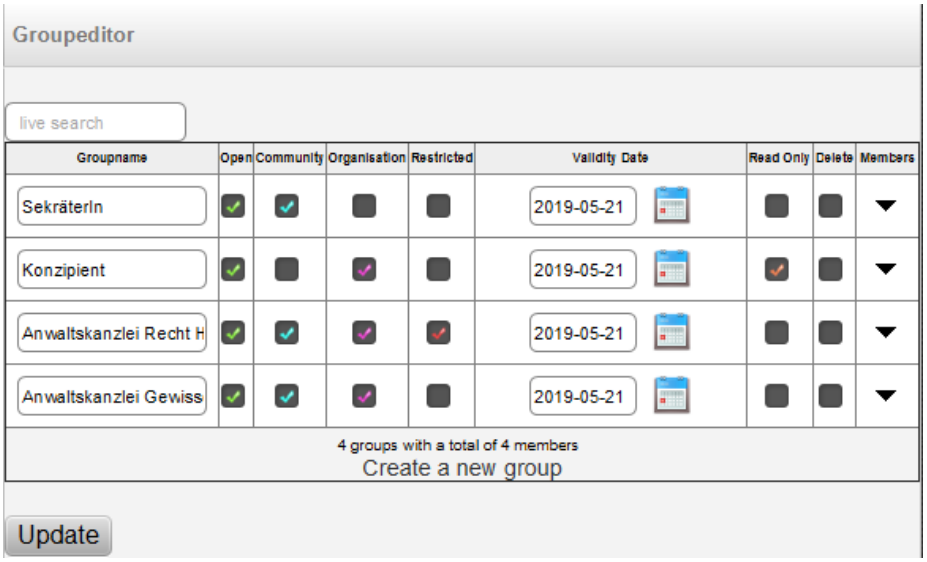

Abbildung 7: Gruppenrechte – eSprint-System

Abbildung 7 zeigt den Groupeditor des Dokument-Erstellers, in unserem Fall des Richters. Innerhalb dieses Bearbeitungsbildschirms können Gruppen erstellt und wieder gelöscht werden. Um den unterschiedlichen Benutzern eine jeweilig andere Sicht auf das Dokument zu gewähren, erstellt der Richter in diesem Fall vier Gruppen. Diesen weist er dann die oben erwähnten Akteure entsprechend ihrer Zugriffsrechte zu. Die folgenden Abbildungen 8–11 zeigen die einzelnen Ansichten der erstellten Gruppen. Textpassagen mit zugewiesenem Zugriffsrecht werden vom Editor farblich gekennzeichnet. Ist der Text rot umrandet, weiß der Betrachter, dass diesem Part „Restricted Access“ zugewiesen ist. Rosa steht für „Organisation Access“, Blau für „Community Access“ und Grün für „Open Access“. Gehört der Betrachter einer Gruppe an, der nicht alle Zugriffsarten zugewiesen wurden, sieht dieser anstatt des Textes einen schwarzen Balken.

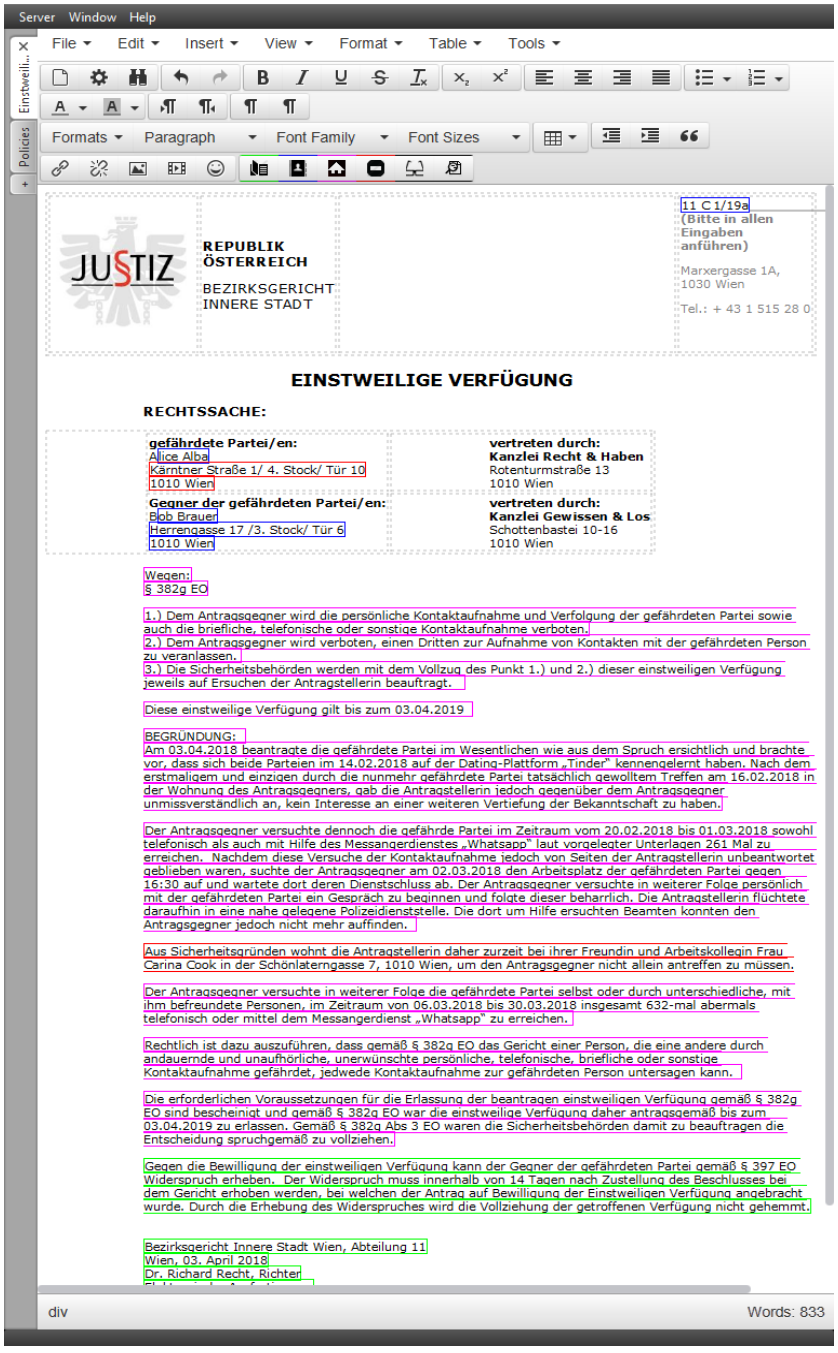

Abbildung 8: eSprint-Ansicht Richter und Rechtsanwalt Antragstellerin

Abbildung 9: eSprint-Ansicht (aller) Sekretärinnen und Sekretäre

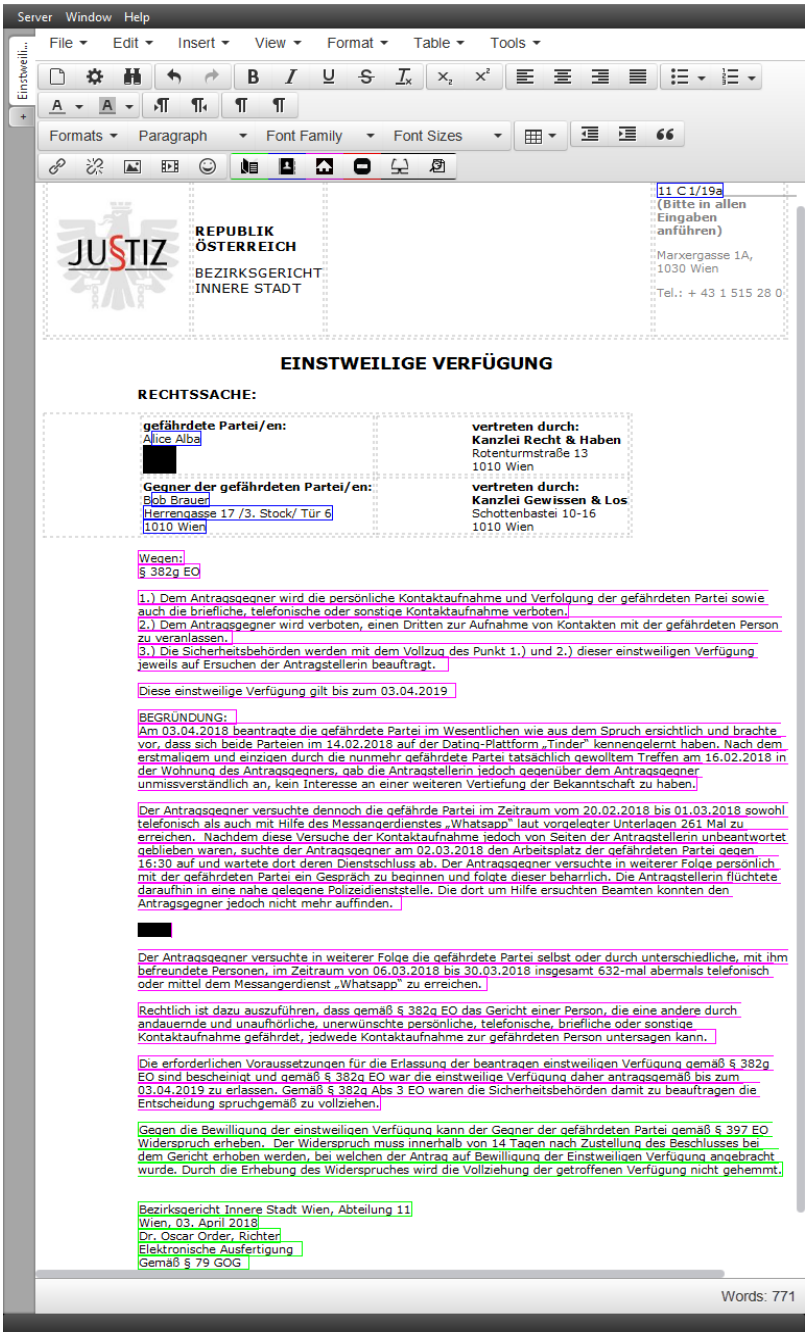

Abbildung 10: eSprint-Ansicht Rechtsanwalt Antragsgegner

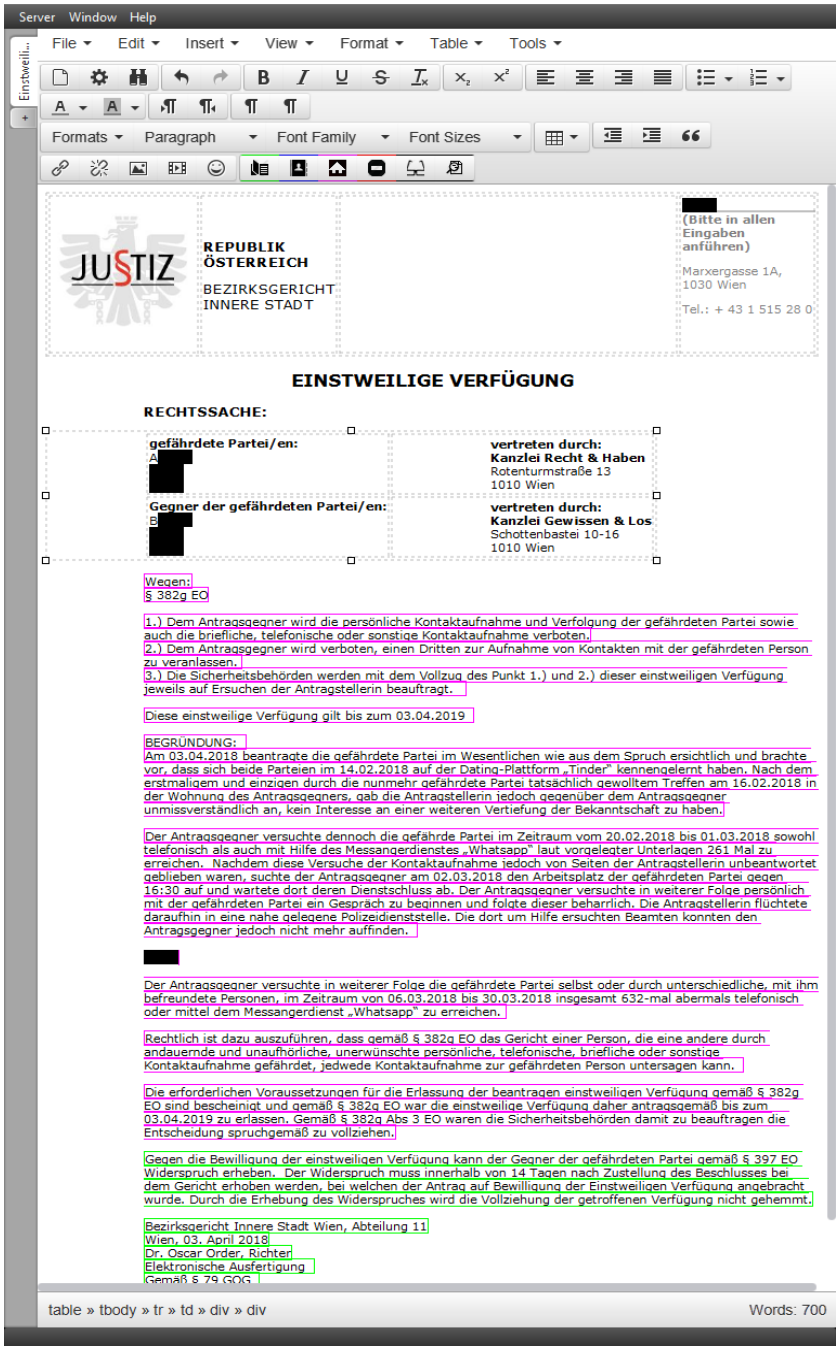

Abbildung 11: eSprint-Ansicht Konzipient

3 State of the Art und Herausforderungen

Bevor es um die Umsetzung des eSprint-Systems geht, war es zunächst notwendig, zu evaluieren, welche weiteren Technologien mit gleichem oder ähnlichem Fokus am Markt existieren. Im Zuge dessen führten die Nachforschungen zu den Themengebieten des Digital-Rights-Managements (DRM) und der Creative Commons Rights Expression Language (kurz ccREL), die mit dem bestehenden Konzept des Vorgängers des hier vorgestellten eSprint-Systems, dem SPIDER-Projekt, verglichen werden.

Das DRM kann mit seinem Sicherheitskonzept (Verschlüsselung, digitales Wasserzeichen, Zertifikate und Signaturen) als Non-Perimeter-Sicherheitsmodel angesehen und als Schutz einer Ressource auch außerhalb von Systemgrenzen angewendet werden. Doch die DRM-Access-Control ist statisch, also nicht anpassbar, und ein feingranulares Klassifikationsschema fehlt gänzlich [BH09].

Das von der Universität Cardiff entwickelte und JISC[3]-unterstützte SPIDER-Projekt stellt hierfür einen vielversprechenden, ressourcenbezogenen Sicherheitsmechanismus-Ansatz vor. Die De-Perimeterisation ermöglicht es, Sicherheitsmaßnahmen auch außerhalb von Systemgrenzen unter Kontrolle zu halten und implementiert eine sogenannte Continous-Access-Control. Das Ziel des SPIDER-Projektes war es, ein Klassifikations-Schema, eine persistente und modifizierbare Zugangskontrolle und einen Zugriffmechanismus für geteilte Informationen innerhalb eines verteilten Netzwerkes zu schaffen, ohne die Möglichkeit zu verlieren, Informationen sicher innerhalb von Unternehmensgrenzen zu auszutauschen. Dennoch blieben vereinzelt Limitierungen bezüglich der Authentifizierung und Identifizierung zurück, die somit die Effektivität und Vertrauenswürdigkeit in Zweifel zogen [BHT09, BH09, MIY06].

In den folgenden Abschnitten wird die Lücke zwischen dem SPIDER-Project, dem DRM und der ccREL betrachtet, beginnend mit einem Überblick bereits existierender Technologien, einer Analyse ihrer Limitierungen und den daraus gezogenen Schlüssen, die gemeinsam mit Frau Nisreen Alam Aldeen zu einem eigenen Forschungsansatz führten. Abschließend werden die Blockchain-Technologie und die von den Autoren vorgenommene Abgrenzung von dieser untersucht [WM08].

[3] JISC, Joint Information Systems Committee, www.jisc.ac.uk

3.1 System-Level-Access-Control

Die traditionelle Strategie zur Absicherung von Ressourcen nennt sich System-Level-Access-Control. Der Grundgedanke eines *perimeterbasierten* Schutzes liegt darin, dass Informationen mittels vorher zugewiesenen Privilegien an einen Nutzer vergeben werden. Durch diese Strategie kann auf eine einfache Weise der Zugang zu sensiblen Informationen eingeschränkt und somit die Einsicht durch unbefugte Dritte verhindert werden und bietet so eine zulängliche Zugangsbeschränkungs- und Authentifizierungsinfrastruktur für ganze Entitäten innerhalb seines Systems. Hierin liegt jedoch bereits die erste Einschränkung, nämlich die Beschränkung auf Bereiche, die innerhalb dieses Systems und dessen Grenzen liegen. Weiters fehlt es dieser Umsetzung an Möglichkeiten für Klassifizierungs-Schemas, nachfolgend in dieser Arbeit auch oft als Access- oder Zugriffsarten bezeichnet, um Informationen spezifischeren Richtlinien unterwerfen zu können. Darüber hinaus gibt es bei diesem Modell auch keine nachhaltig durchdachten Handhabungen für modifizierbare Zugriffsrechte außerhalb der zuvor angesprochenen Systemgrenzen [BH09, TOL07, EPP11].

3.2 De-Perimeterisation

Die oben erwähnten Punkte führen zu der Fragestellung, wie diese Probleme gelöst werden können. Das eingangs genannte SPIDER-Projekt, das die Grundlage dieser Arbeit bildet, beantwortet die Frage mit der sogenannten De-Perimeterisation. Dieses Konzept sieht eine Umorientierung weg von einem systemweiten Sicherheitsansatz und hin zu ressourcen- beziehungsweise datenbezogenen Sicherheitsmechanismen vor. Dieser Schritt ermöglicht es einem Unternehmen, beispielsweise seine Sicherheitsmaßnahmen auch außerhalb seiner Systemgrenzen unter Kontrolle zu behalten. Es implementiert also eine sogenannte „Continous Access Control“ [BH09].

Die erste Beschreibung von De-Perimeterisation ging vom Jericho-Forum aus und deklarierte diese nicht als konkreten Lösungsansatz, sondern vielmehr als eine Art grundlegenden Gedanken für eine erfolgreiche Umsetzung. Die Eckpunkte der damaligen Überlegungen waren [TOL07, NAI00, PG05, CW09]:

- *Inner Protection*

  Die Sicherheit muss dort liegen, wo auch die Ressource ist – dies wird auch als Data-Level-Authentication and Protection bezeichnet. Bei einem Textdokument etwa muss der Text an sich geschützt sein. Hierdurch werden eine vollständige Unabhängigkeit von äußerlichen Sicherheitsmechanismen erzeugt und somit eine Trennung zwischen Fernzugriff und lokalem Zugriff auf eine Ressource obsolet gemacht.
- *Selbstbestimmung*

  Der Owner oder auch Inhaber einer Datenquelle kann selbst darüber entscheiden, wer Zugriff auf die Daten erhält und noch wichtiger, in welchem Zeitraum der jeweilige Nutzer Zugriff auf die Ressource hat. Jedem Client und Server ist es somit möglich, sich zu schützen und selbst nachträglich Nutzungsrechte einzuschränken oder zu erweitern.
- *Zentralität*

  Es gibt ein zentrales System zur Authentifizierung.
- *Tools*

  Die bereitgestellten Werkzeuge umfassen Verschlüsselung, sichere Protokolle und Computersysteme sowie Data-Level-Protection.

3.3 Bestehende De-Perimeterisation-Protection-Ansätze

Es gibt zahlreiche Ansätze, die das neuartige Datenschutzverständnis teilweise in ihrer Implementierung enthalten, so zum Beispiel das klassische DRM, das bereits eine Kontrolle von Inhalten über die Unternehmensgrenzen hinaus zulässt. Dieser Ansatz ist zwar sinnvoll, jedoch umfasst er nicht den Punkt 2 des De-Perimeterisierungs-Gedankens: die nachträgliche Kontrolle und Modifikation der vergebenen Rechte. Weiters gab es in den Jahren 2001 und 2002 zwei Ansätze dazu, wie Informationen in verteilten Systemen über das Internet verwaltet werden könnten. Diese Entwicklung sah einen Schutz der Informationen mittels XML-Definitionen vor, ließ aber ebenfalls das nachträgliche Managen und Modifizieren von Zugriffsrechten außer Acht. An diesen beiden Beispielen lässt sich erkennen, dass zur Lösung der Problemstellung ein anderer Weg erforderlich ist [BC01, DVS02].

3.3.1 Digital-Rights-Management

Das DRM ist weder ein Standard noch eine Architektur. Es umfasst lediglich eine Vielzahl an Ansätzen zum Schutz digitaler Besitztümer. Ein DRM-System sollte folgenden Kriterien entsprechen [SB12, BG08, DRM06]:

1. Bereitstellen eines andauernden Schutzes von Daten gegen unbefugten Zugriff, Schaffen eines Zugangs nur für entsprechend autorisierte Entitäten,
2. technische Unterstützung verschiedenster digitaler Medien wie Musikdateien, Video-Inhalte, Texte (Bücher) und Bilder,
3. Unterstützung bei einer Verwendung dieser Medien auf verschiedensten Plattformen (z. B.: PC, Smartphone, iPods),
4. Unterstützung einer Datenverteilung über verschiedenste Datenträger wie etwa via USB-Sticks oder DVDs.

Eine Umsetzung dieses Gedankens findet sich heute in einer Vielzahl von Anwendungen wie Adobe Digital Editions, Microsoft Windows Media Digital-Rights-Management oder auch der Computerspieleplattform Steam. Die dort umgesetzten DRM-Ansätze sind zwar gut, jedoch sind sie aus der Sicht der Autoren dieser Arbeit zu statisch und umfassen nicht vollends den Punkt 2 des De-Perimeterisierungs-Gedankens – die nachträgliche Kontrolle und Modifikation der vergebenen Rechte.

Detaillierter hat Frau Nisreen Alam Aldeen zwei weitere Ansätze aus 2001 und 2002 betrachtet. Diese befassten sich im Speziellen mit der Thematik, wie Informationen in verteilten Systemen über das Internet verwaltet werden können. Diese Entwicklung sah einen Schutz der Informationen mittels XML-Definitionen vor, um auch eine Fragmentierung der zu schützenden Ressourcen zu erlauben. Dies ermöglicht, unterschiedliche Kontrollen spezifischer Schutzanforderungen an eine Ressource mit Zuhilfenahme von Verschlüsselungstechniken und einem granularen Schlüsselmanagement durchzuführen. Durch die dort beschriebene Herangehensweise kann auf Teile einer Ressource mittels verschiedener Schlüssel, basierend auf zuvor festgelegten Rechten, zugegriffen werden. Ein Anwender kann so also nur auf jene Teile zugreifen, die ihm vorher zugewiesen wurden. Dennoch fehlen auch diesen beiden Ansätzen eine nachhaltige Kontrollmöglichkeit und Modifizierbarkeit der vergebenen Rechte sowie ein entsprechendes Key-Management (inklusive Vergabe

und Rücknahme der Schlüssel) für manipulierbare Daten innerhalb eines kollaborativen Umfelds [ARE13].

3.3.2 Creative-Commons-Rights-Expression-Language

Die ccREL folgt einem ähnlichen Grundgedanken wie das DRM und versucht dabei, schützenswerte Ressourcen durch die Verwendung von verschiedenen Kennzeichen, beispielsweise innerhalb eines Textes, mit unterschiedlichen Zugriffsrechten zu versehen. Hierfür wird je gekennzeichneter Datei eine passende Meta-Datei benötigt, die zum Beispiel im RDFa-[4] oder auch XMP[5]-Format vorliegen kann [DVS02, BC01].

Diese Art der Umsetzung verfügt über zwei besonders hervorzuhebende Eigenschaften. Zum einen stellt einem ccREL kein eigenes Werkzeug zur Umsetzung der Zugriffskontrolle zur Verfügung, sondern gibt einem eine Reihe an Möglichkeiten, um mittels der Expression-Language schützenswerte Passagen zu kennzeichnen und so eine effektive Verwaltung der Userberechtigungen zu erhalten. Zum anderen fügt ccREL eine Live-URL in eine Ressource ein, anhand derer die verlinkten Copyrights des Contents verfolgt werden können.

Um sich einen kurzen Einblick in diese Technologie verschaffen zu können, wird nachfolgend ein Beispiel für eine mögliche Umsetzung gezeigt.

```
the study by Wesenauer Thomas and Pollak Patrik is licensed under a
<a rel="license" href="http://creativecommons.org/licenses/by-
nc/3.0/">Creative Commons
Attribution Non-Commercial 3.0 License</a>.
Permissions beyond the scope of this license may be available at
<a xmlns:cc="http://creativecommons.org/ns#" rel="cc:morePermissions"
href="https://esprint.org/permissions/">eSprint</a>.
```

Abbildung 12: ccREL-Textpassage mit Klassifikations-Tag

Wie in Abbildung 12 erkennbar ist, ließ sich anhand dieser Technologie ein feingranulares Klassifikations-Schema erzielen.

[4] Resource Description Framework for Attributes
[5] Extensible Metadata Platform

Der Grundgedanke hinter ccREL ist vielversprechend, jedoch gibt es auch hier wiederum einige Unzulänglichkeiten in Bezug auf eine nachhaltige und vor allem skalierbare Zugriffskontrolle. Zum einen fehlt es dieser Technologie an einer Möglichkeit, die gesetzten Metainformationen nachträglich zu modifizieren und zum anderen können die zugriffsberechtigten Ressourcennutzer schwierig administriert werden. Darüber hinaus existiert bisher kein umgesetztes Tool zur nutzerfreundlichen Erstellung von ccREL-geschützten Dokumenten [AAL08].

3.3.3 Das SPIDER-Projekt

Das SPIDER-Projekt hatte hingegen ein dem Prinzip De-Perimeterisation am ehesten entsprechenden Ansatz gewählt. Die dort gewählten ressourcen- beziehungsweise datenbezogenen Sicherheitsmechanismen dienten Frau Nisreen Alam Aldeen und in weiterer Folge auch den Autoren dieser Arbeit als Inspiration für den hier entstandenen eSprint-Lösungsansatz (siehe Kapitel 7 Implementierungsansatz). SPIDER stellt ein entsprechendes Klassifikations-Schema sowie eine persistente und modifizierbare Zugangskontrolle bereit [BHT09]. Zudem versucht es durch Kombination vorangegangener Methoden [AAL08, DVS02, BC01] und der daraus extrahierten angedachten Continous-Access-Control, einen praktikablen Weg für einen Zugriffsmechanismus zu schaffen, um geteilte Informationen innerhalb eines verteilten Netzwerkes bereitzustellen, ohne die Möglichkeit zu verlieren, Informationen sicher innerhalb von Unternehmensgrenzen zu teilen. Hierzu verwendet SPIDER den Ansatz aus der zuvor vorgestellten ccREL-Technologie, indem es eine Möglichkeit schafft, die von dort bekannten Klassifikations-Schemata in Form von Tags mittels einer Schaltfläche in eine Textpassage zu injizieren. Diese Tags dienen auch dazu, die fixen Grenzen rund um eine Ressource zu entfernen und diese innerhalb der Ressource neu, nur rund um den zu schützenden Inhalt, definieren zu können [BH09]. Das Projekt produzierte auch ein grafisches User-Interface, das über dem darunterliegenden XML-Daten-Mechanismus operierte, um den eingebetteten Access-Control-Mechanismus zu unterstützen [BHT09, BH09, MIY06].

Abbildung 13 zeigt die Architektur des SPIDER-Projektes. Hier sitzt die Access-Control-Policy in einer zentralen Datenbank serverseitig und speichert

Nutzerinformationen, Klassifikationsdaten, Verschlüsselungs-Schlüssel sowie die eindeutigen Identifizierer zu den einzelnen Ressourcen.

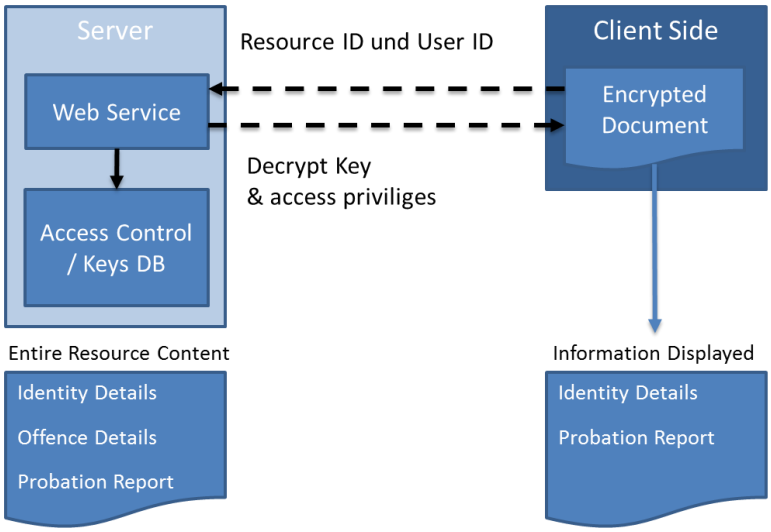

Abbildung 13: Architektur des SPIDER-Projekts

Aufgrund der unvollständigen Umsetzung des SPIDER-Projektes blieben vereinzelt Limitierungen bezüglich der Authentifizierung und Identifizierung zurück, die somit die Effektivität und Vertrauenswürdigkeit in Zweifel zogen. So sieht das SPIDER-Projekt nur eine Verschlüsselung einer gesamten Ressource vor, womit nicht anzuzeigende Informationen innerhalb des eigenen Client-Speichers lesbar sein dürften. Weiterhin ist ein Schutz des übertragenen Schlüssels und der Zugriffskontrollmechanismen während der Übertragung nicht definiert. Darüber hinaus gewährt SPIDER auch keinen Schutz gegenüber einer möglichen Vervielfachung geteilter Informationen. Für einen Identifizierungsmechanismus reicht eine einfache Benutzername-/Passwort-Authentifizierung nicht aus. Gedanken über den Einsatz eines digitalen Zertifikates sollten letztlich umgesetzt werden.

3.4 Vergleichsmatrix

Die Idee hinter De-Perimeterisation war der Grund, warum dieses Projekt realisiert wurde. Hierzu wurden in den vorherigen Abschnitten zwei Umsetzungen von DRM

sowie ccREL beleuchtet und ihre Lücken in Bezug auf De-Perimeterisation betrachtet. Tabelle 2 zeigt die von Frau Nisreen Alam Aldeen im Zuge ihrer Arbeit „Enhancing Privacy Protection in Distributed Environments through Identification and Authentification-Based Secure Data-Level Access Control“ aufgestellte Vergleichsmatrix und erweitert diese um das in der vorliegenden Arbeit entwickelte eSprint-System. Inspiriert vom SPIDER-Projekt wurde versucht, die Schwächen seiner Vorgänger auszubessern und allen auferlegten Kriterien zu genügen.

Property		Access control technique					
		DRM	2001	2002	ccREL	SPIDER	eSPRINT
1	Apply control outside perimeter boundaries	●	●	●	●	●	●
2	No additional burden for key management including issuing and revoking keys	○	○	○	●	●	●
3	Actual technical tool of access control	●	●	●	○	●	●
4	Fine grained classification scheme	○	●	●	●	●	●
5	Continuous and modifiable control after information is being shared	○	○	○	○	●	●
6	Managed easily by the owner	○	○	○	○	●	●
7	Managed easily by the client	○	○	○	○	●	●
8	Encryption through Multiple Key Mechanism	○	●	●	○	○	●
9	Identity management using digital certificates	○	○	○	○	○	●
Points Covered		2	4	4	3	7	**9**

Tabelle 2: Vergleichsmatrix behandelter Methoden [ARE13]

3.5 Abgrenzung zu Blockchains

Die innovative Datenbanktechnologie Blockchain bringt bedeutende Veränderungen im Bereich der IT sowie unterschiedlicher Branchen der Wirtschaft und Industrie. Ihre Daten- und Manipulationssicherheit gegenüber herkömmlichen Datenbanken ist ihr größtes Asset. Das Modewort Blockchain ist in allen Arbeitsbereichen der Wirtschaft entweder bereits vertreten oder über deren Einsetzbarkeit wird zumindest nachgedacht beziehungsweise teilweise entwickelt. So könnten im Bereich der Justiz durch Smart-Contracts ohne den Einsatz von Notaren und Anwälten geschlossen werden. Im

Bankensektor könnte die Blockchain zu einer wesentlichen Verschlankung der Infrastruktur führen. Großunternehmen wie SAP betreiben Blockchain-Initiativen und möchten mithilfe dieser Technologie eine digitale Logistikkette in ihre Cloud integrieren. Eine Vielzahl an Anwendungen befindet sich derzeit bereits in Alpha- oder Betaphasen.

Die mediale Berichterstattung und der Hype um Bitcoin, Ethereum und Co. haben die Frage aufgeworfen, ob eine Anwendung der Blockchain auf die bestehende Forschungsfrage zielführend sein könnte.

Eine gut auf diese Anwendung umlegbare und vergleichbare Applikation wurde bisher noch nicht entwickelt. Deswegen soll hier kurz Überlegungen zu Smart-Contracts vorgestellt werden. Smart-Contracts sind Computerprogramm-Logiken, die Verträge abbilden oder überprüfen können. So können die Verhandlung und Abwicklung eines Vertrags technisch unterstützt und eine schriftliche Fixierung überflüssig gemacht werden. Dieser Anwendungsfall der Blockchain-Technologie soll in Zukunft bürokratische Strukturen in allen Bereichen abbauen. In einer verteilten Systemlandschaft können einzelne Teilnehmer vertrauenswürdig miteinander interagieren, ohne eine vertraute dritte Partei zu benötigen. Mit der Veröffentlichung von Ethereum beginnen die ersten Unternehmen, eine Vielzahl an Smart-Contracts zu schaffen. Die auf Ethereum aufgebauten Applikationen vertrauen meist auf die dezentrale Sicherheit der Blockchain, wobei auch zu erwähnen ist, dass bereits Attacken auf verschiedene Aspekte des Systems identifiziert wurden [BIT14, BIT15]. Eine weitere bedeutende Frage ist der mit den Blockchain-Technologien verbundene Aufwand. Nachdem jeder Miner die Smart-Contract-Programme ausführen und verifizieren muss, entstehen Transaktionskosten, die dem Gesamtnutzen gegenübergestellt werden müssten [BLK16].

Diese und die Frage der Abbildung der von den Autoren angestrebten Kriterien der Continous-Access-Control, der Suche nach einem feingranularen Klassifikationsschema sowie der Nutzbarkeit und Verwaltbarkeit von Ressourcen durch einzelne Teilnehmer wären zwar interessant, können allerdings aufgrund des begrenzten Rahmens dieser Arbeit nicht erforscht werden.

4 Implementierungsumfeld

Dort, wo sich das Internet zuvor noch mit rein statischen Dokumenten und deren Interpretation durch einen Browser präsentiert hatte, sind heute dynamische Webinhalte und Services, mit deren Hilfe sowohl Menschen als auch Maschinen miteinander kommunizieren können. Hierzu zählen beispielsweise Angebote wie Twitter, Facebook, Wikis und Onlinehandel, aber auch ausschließlich maschinenbezogene Services wie etwa ein Dienst zur Synchronisierung der Systemuhrzeit. Ein Service stellt also einem User beziehungsweise einem Client eine bestimmte Funktionalität zur Verfügung und ist darüber hinaus im Idealfall durch einen ähnlichen anderen Dienst substituierbar, wodurch eine erhöhte Ausfallsicherheit erzielt werden kann.

Als Beispiel hierfür kann das bekannte Social-Media-Portal Facebook betrachtet werden. Abseits der Weboberfläche für Endnutzer kann der Zugriff hier auch über einen entsprechenden Webservice erfolgen. Beliebigen Applikationen wird es hierdurch ermöglicht, Zugriff zur Aktualisierung des eigenen Status oder zum Anzeigen von Neuigkeiten von anderen Benutzern zu erlangen – vorausgesetzt, die entsprechende Autorisierung wurde durch den Nutzer erteilt. Dies soll als einfaches Beispiel für die Möglichkeiten durch die vielfältige Anwendbarkeit von Webservices dienen.

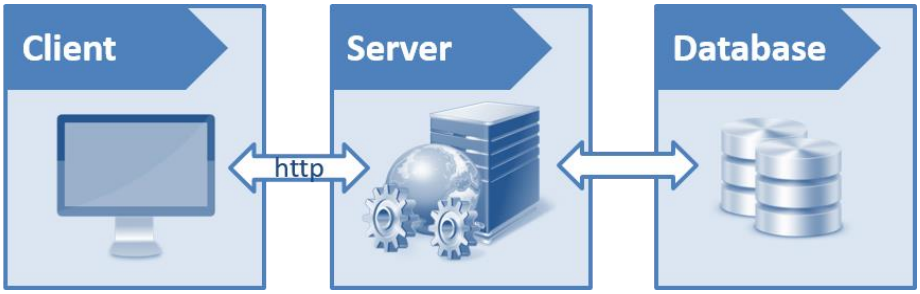

Abbildung 14: eSprint-3-Tier-Architektur (Überblick)

Zur detaillierteren Erläuterung der eingesetzten Kommunikationstechnologien wird nachfolgend zunächst eine grobgranulare Darstellung des eSprint-Systems, wie in Abbildung 14 ersichtlich, vorgenommen. Als Architektur wurde eine klassische 3-Tier-

Architektur gewählt, die als klare Trennung zwischen Client- (Presentation Layer) und Serverlogik (Application Layer) dient sowie eine separate Speicherung der persistenten Daten (Data Layer) gewährleistet. Der größte Vorteil in der getroffenen Wahl liegt in der einfacheren Wartbarkeit sowie der schnellen Austauschbarkeit der Webservicemodule. Zuerst werden die beiden grundlegenden Technologiekomponenten, die zur Kommunikation mit einem Webservice notwendig sind, betrachtet: die sogenannten Uniform Resource Identifier (URI) und das Hypertext Transfer Protocol (HTTP) [BLBC+02].

4.1 Uniform-Resource-Identifier

Mittels eines URI lassen sich Ressourcen im Web eindeutig identifizieren. Unter Ressourcen (Resources) werden im Zusammenhang mit Webservices die Zugangspunkte verstanden, unter denen die jeweilige Funktion des Service angesprochen werden kann. Eine URI kann durch einen Unified Resource Locator (URL) oder durch einen Uniform Resource Name (URN) repräsentiert werden oder aus einer Kombination dieser beiden bestehen. Die am meisten verwendete Art der Adressangabe stellt die URL dar. Diese setzt sich aus dem verwendeten Protokoll, der Hostadresse inklusive des Ports und dem absoluten Pfad der Webserviceressource zusammen. Letztere kann bei Bedarf in eigene Unterfunktionen gegliedert werden, um eine noch genauere Granularität der Funktionen gewährleisten zu können.

Abbildung 15: Webservice-URI – allgemeines Beispiel

Einen Endpunkt im eSprint-System stellt, wie in Abbildung 15 gezeigt, beispielsweise die Funktion http://serverurl:8441/connect oder http://serverurl:8441/store dar. Diese beiden URLs stellen exemplarisch die Funktion zum Verbindungsaufbau zum Service und zur Speicherung neuer Daten bereit. Eine mögliche Unterfunktion der angebotenen Services wäre etwa

http://serverurl:8441/store/projects, die wiederum alle aktuellen Projekte als Rückgabe liefert [BLFM05, RR07, BLBC+02].

4.2 HTTP/1.1 – Hypertext Transfer Protocol

Zunächst werden theoretische Grundlagen zur Datenübertragung angeführt. Für den Informationsaustausch im World Wide Web gibt es eine Vielzahl an möglichen (Anwendungs-) Protokollen. Zu diesen zählen beispielsweise SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol), FTP (File Transfer Protocol) oder das für diesen Kontext relevante Hypertext Transfer Protocol (HTTP). Letzteres stellt ein zustandsloses Protokoll zur Übertragung von Daten dar. Dies bedeutet, dass *keine* Sitzungsinformationen der beteiligten Kommunikationspartner gespeichert werden. Stattdessen wird jeder Verbindungsaufbau als eigenständige Transaktion betrachtet, wobei alle benötigten Daten zum Datenaustausch in der jeweils individuellen Nachricht enthalten sein müssen [RFC2616].

Die Nachricht, die zwischen Client und Server ausgetauscht wird, kann entweder ein sogenannter Request (eine Anfrage) oder eine Response (eine Antwort) sein. Bei einer Kommunikation sendet der Client, der im Kontext eines Webservice sowohl durch einen Menschen als auch durch ein anderes Programm repräsentiert werden kann, eine dreiteilige HTTP-Anfrage an den Server.

Der erste Teil enthält hierbei den HTTP-Header, der sich wiederum aus diversen Headerfeldern zusammensetzt. Der Header enthält zunächst die Adressangabe der angesprochenen Ressource, an welche die Anfrage gestellt wird. Diese wird durch eine eindeutige URI identifiziert. Außerdem wird hier die Art der HTTP-Anfragemethode festgelegt, die auch oft als ‚Verb' bezeichnet wird. Die Verben ermöglichen es dem Server, unter ein und derselben Ressource-URI gleich mehrere Ansprechpunkte für den Client bereitzustellen. Zu solchen Verben gehören [RFC2616, KR01, RO08]:

- GET

  Mittels der GET-Methode wird eine Instanz der angesprochenen Ressource angefordert. Die Besonderheit hierbei ist, dass die Daten per Definition lediglich angefordert, nicht aber verändert werden und somit keine Seiteneffekte beziehungsweise Veränderungen an der Ressource stattfinden. Zwar können

beispielsweise einfache Strings vom Client an den Webservice übermittelt werden – wie etwa Suchparameter – jedoch kann beziehungsweise sollte dies nur in begrenztem Ausmaß genutzt werden. Die Strings werden an die URI der angesprochenen Services angehängt, wodurch die Nutzung in Verbindung mit sicherheitsrelevanten Informationen nicht empfohlen wird. Das bedeutet, dass ein Client, der einen GET-Request versendet, keine Verpflichtungen eingeht. Daher wird ein GET generell auch als sicher oder idempotent[6] bezeichnet.

- POST

  Die POST-Methode kann – im Gegensatz zur GET-Methode – zur umfangreichen Duplex-Datenübertragung zwischen Client und Webservice genutzt werden. Diese Methode findet im eSprint am häufigsten Anwendung, da sie den direkten Austausch von Informationen ermöglicht. Die zu sendende Information besteht aus sogenannten Bodyparts, welche die eigentliche Information beinhalten, sowie einem Header, in dem der übermittelte (Internet-) Medientyp festgelegt wird. Dieser gibt den übermittelten Datentyp der Information an. Der Server erfährt hierüber, wie er die Daten interpretieren muss. In diesem Projekt wurde der Application-/Octet-Stream-Typ gewählt, da die gesendeten Daten zumeist aus Rohdaten-Strings bestehen, die keinem der vordefinierten Datentypen entsprechen. Weiters werden diese Octet-Stream-Objekte in ein sogenanntes Multipart-Objekt verpackt, um gleich mehrere voneinander differenzierbare Objekte in einem Datenstrom senden zu können. Wenn die zu sendende Nachricht, bestehend aus dem Header und dem (Multipart-) Body, vollständig zusammengesetzt wurde, kann diese an den Webservice übermittelt werden. Die Daten befinden sich hier innerhalb des HTTP-Request-Streams und sind somit auch durch gegebenenfalls implementierte Sicherheitsmechanismen wie HTTPS geschützt. Auf diese wird in Kapitel 5.5 näher eingegangen.

[6] Ein Stück Programmcode, das bei mehrmaliger Ausführung gleiche Ergebnis wie bei einmaligen Aufruf liefert

- HEAD

  Bei einer Head-Anfrage sendet der Client – ähnlich wie bei einem GET-Request – eine Nachricht an den Webservice. Dieser antwortet jedoch, ohne einen Nachrichtenrumpf an den Client zurückzusenden, sondern lediglich Meta-Informationen, die sich im Message-Header befinden.
- DELETE

  Über eine Delete-Anfrage des Clients können Ressourcen vom Webservice gelöscht werden. Diese werden mittels einer URI identifiziert. Diese Methode findet vor allem in Verbindung mit RESTful-Webservices Anwendung.
- TRACE

  Anhand der Trace-Methode lässt sich eine gesendete Nachricht an einen Webservice nachverfolgen. Dies ermöglicht eine Interpretation, über welche Route eine Nachricht übermittelt wurde, ob gegebenenfalls Informationen entfernt oder hinzugefügt wurden und bei Bedarf ein Debugging bei Verbindungsproblemen durchgeführt wird.
- OPTIONS

  Sendet der Client eine Options-Anfrage an eine spezifische URL eines Webservice, so gibt dieser eine Liste der unterstützten HTTP-Methoden zurück.
- CONNECT

  Die Connect-Methode findet meist beim Verbindungsaufbau einer sicheren SSL-Verbindung zu einem Webservice über einen Proxy-Server Anwendung. Hierdurch wird im Falle von eSprint sichergestellt, dass es dem Client immer möglich ist, eine gesicherte Verbindung mit dem Server herstellen zu können. Im Falle eines fehlgeschlagenen Verbindungsaufbaus gibt das eSprint-System dem Benutzer eine entsprechende Rückmeldung sowie eine detaillierte Fehlermeldung zurück.

Darüber hinaus existieren die Methoden PUT und PATCH, die im Zuge dieser Arbeit jedoch nicht benötigt wurden und daher auch nicht näher betrachtet werden. Um einen entsprechenden Informationsaustausch zwischen Client und Webservice besser veranschaulichen zu können, wird in der Tabelle 3 ein Beispiel einer POST-Methode

aufgeführt, das eine exemplarische Anfrage der zugehörigen Dateien zu einem bestehenden Projekt veranschaulicht.

	Request	*Response*
Request / Status Line	POST /Master_Extended/rest/getprojects HTTP/1.1	HTTP/1.1 200 OK
Header	Host: https://80.108.204.240:8443 Content-Type: multipart/form-data Content-Length: **length**	Content-Type: multipart/form-data Content-Length: **length**
Message-Body	userid=1411&projectid=1337	projectid=1337&docid=15

Tabelle 3: Beispiel eines POST-Requests und der zugehörigen Response

4.3 Webservicearchitekturen

Für die Untersuchung von Webservices und deren Umsetzung ist zunächst eine Beschreibung der serviceorientierten Architekturen (SOA, Service-oriented Architecture) notwendig. Seit der Einführung des SOA-Konzepts wurde eine Vielzahl von Büchern geschrieben, die sich der Einführung, Umsetzung und Pflege einer solchen IT-Landschaft widmen. Die Gemeinsamkeit all dieser Werke ist, dass es keine standardisierte Definition dessen gibt, was notwendig ist, um ein SOA-konformes Serviceangebot bereitzustellen [CD+02].

4.3.1 Serviceorientierte Architekturen

Der Ansatz von SOA ist die Gewährleistung von skalierbaren und flexiblen Systemen, die in ihrem Umfang auch jederzeit wachsen und sich verändern können. Die drei Kernpunkte, auf denen serviceorientierte Architekturen basieren, umfassen:

- *Services*

  Serviceorientierte Architekturen sollen möglichst atomare Systeme darstellen. Sie sollen einzelne Geschäftsfälle in ihrer Funktion abbilden und modular den Servicenutzern über diverse Technologien und Plattformen bereitgestellt werden.

- *Infrastruktur*
  Es gibt eine spezielle Infrastruktur, auch Enterprise-Service-Bus genannt, die es ermöglicht, die angebotenen Webservices einfach und flexibel zu verbinden und anzubieten.
- *Richtlinien*
  Es müssen Richtlinien und Prozessbeschreibungen festgelegt werden, die der Veränderbarkeit und dem möglichen Wachstum von Servicelandschaften Rechnung tragen.

Diese Definition macht SOA zu einem gut durchdachten Konzept, wodurch die Komplexität von großen Unternehmen in kleine verwaltbare Einheiten unterteilt werden kann. Die Hauptziele hierbei liegen in der Vereinfachung von Systemintegrationen von Neu- und Legacysystemen, in der Förderung der Wiederverwendbarkeit von Modulen sowie in der problemloseren Adaptierung vorhandener Programm-APIs und somit der Senkung der insgesamt anfallenden Betriebskosten der Infrastruktur. Jedoch kann SOA nicht gekauft oder einfach installiert werden. Es ist lediglich eine Vorstellung der idealen unternehmerischen Aufteilung einer Diensteinfrastruktur und muss somit individuell an die Anforderungen des Unternehmens angepasst werden [CD+02].

Dass das Paradigma zumindest theoretisch umsetzbar ist, wurde bereits durch die zugrundeliegenden Technologien bewiesen. Diese befinden sich im aktiven Einsatz und wurden ständig den wachsenden Anforderungen angepasst. Aus diesem Grund wurde auch für das vorliegende eSprint-Projekt eine dieser Technologien gewählt, nämlich der sogenannte RESTful-Ansatz. Um dessen Vorzüge im Zusammenhang mit der gegebenen Aufgabenstellung gegenüber der anderen gängigen Technologie – dem Simple-Object-Access-Protocol – besser erläutern zu können, werden nachfolgend die beiden genannten Umsetzungsvarianten zur Bereitstellung von Webservices vorgestellt und miteinander verglichen [RO08, RR07].

4.3.2 SOAP – Simple-Object-Access-Protocol

Das Simple-Object-Access-Protocol (SOAP) stellte lange Zeit die gängigste Technologie dar, um Webservices in Netzwerken und über das World Wide Web zugänglich zu machen. Erstmals vorgestellt Ende der 1990er Jahre, gestattet es dem Client, die serverseitig bereitgestellten Methoden ansprechen und nutzen zu können. Diese Aufrufe geschehen durch W3C[7] weitestgehend standardisierte und teils komplexe XML-Nachrichten, die über HTTP ausgetauscht werden [SOW3C].

Genau diese Kombination aus zwei anerkannten und weitverbreiteten Standards hat SOAP zu einem Standard bei der Umsetzung von Webservices gemacht. Es ermöglicht einen plattformunabhängigen Datenaustausch, auch zwischen unterschiedlich programmierten Kommunikationspartnern. So stellt etwa der Aufruf einer serverseitigen Java-Methode von einem in C++ programmierten Client aus kein Problem dar.

Um den Datenaustausch zwischen Server und Client zu ermöglichen, setzt SOAP neben dem eigentlichen Transportprotokoll auf XML-basierte Nachrichten. Diese Technologie nennt sich auch Web-Service-Description-Language – kurz WSDL. Primär dient WSDL dazu, dem Client die vom Server angebotenen Interaktionsmöglichkeiten bereitzustellen. Es listet den Namen der Services auf, welche Operationen diese dem Client zur Verfügung stellen sowie welche Ein- und Ausgabeparameter erwartet oder zurückgegeben werden und welcher Datentyp diese sind. Eine WSDL dient also als Richtlinie, wie die ausgetauschten Daten strukturiert werden müssen, damit beide Kommunikationspartner diese auch verarbeiten können. Diese kurze Beschreibung umfasst nur die wichtigste Funktionalität einer WSDL. Zusätzlich können eine Vielzahl anderer Felder definiert und übertragen werden, was auf die größte Schwäche dieser Technologie hindeutet. Auch wenn die Idee eines WSDL-basierten Services wegen seiner Interoperabilität, der guten Lesbarkeit und des damit verbundenen Charakters einer gleichsam automatischen Dokumentation aller angebotenen Informationen zunächst vorteilhaft wirkt, so gibt es nicht zu vernachlässigende Nachteile dieser Technologie.

[7] World Wide Web Consortium

Eine WSDL-Datei kann im übertragenen Sinn auch als eine Art elektronischer Vertrag zwischen dem Serviceprovider und dem jeweiligen Client betrachtet werden, der von beiden zur erfolgreichen Kooperation eingehalten werden muss. Ändert sich nun eine Kleinigkeit am Service selbst, beispielsweise ein neu hinzugefügter optionaler Parameter, so muss auch die WSDL entsprechend angepasst werden, andernfalls kann diese nicht an den Client weitergegeben werden. Dies bedeutet, dass die Clients anhand der geänderten WSDL erneut kompiliert werden müssen. Diese Einschränkung wird auch oft als eine Art Version-Lock-In bezeichnet, da der Serviceprovider durch die nur erschwert aktualisierbare API diese im Nachhinein meist nicht mehr verändert. Dies wäre in Bezug auf das hier entwickelte eSprint-System nachteilig für die fortwährende Verbesserung und Erweiterung der verwendeten Sicherheitsstandards sowie des Funktionsumfanges gewesen und war ein Grund dafür, diese Technologie nicht zu wählen.

Ein weiterer Schwachpunkt von SOAP ist der produzierte Overhead beim Austausch jeder einzelnen Nachricht. Die hierfür benötigten XML-Nachrichten beinhalten zahlreiche Metadaten und vergleichsweise wenige Nutzdaten, was vor allem das Zusammensetzen und Auslesen eben dieser zu einer rechenintensiven Aufgabe machen kann. Dies ist in zweierlei Hinsicht nachteilhaft: Einerseits wurde das eSprint-System unter der Voraussetzung entwickelt, dass dessen Client auch auf einem hardwaremäßig schwächer ausgestatteten Computersystem lauffähig bleibt und andererseits müsste der nicht benötigte Overhead nicht nur jedes Mal mitübertragen, sondern auch ver- und entschlüsselt werden. Dies würde den Aufwand auf Seiten der Anwender unnötig erhöhen, wodurch sich diese Technologie letzten Endes als nicht zielführend für die Umsetzung herausgestellt hat [WC+05, ML07, GHM+00].

4.3.3 RESTful Webservices – Representational-State-Transfer

Der RESTful-Webservicestandard stellt eine neue Herangehensweise an das Thema Webservices dar. Erstmals wurde dieser Standard durch Roy Thomas Fielding in seiner Dissertation beschrieben und er repräsentiert eher eine Leitlinie zur Umsetzung von Webservices. Fielding, der bereits an der Entwicklung vieler Webstandards mitwirkte und ehemaliger Vorsitzender der Apache Software Foundation war, beschrieb RESTful

als einen Architekturstil, der bereits bestehende Webstandards zur Umsetzung nutzen sollte. Es handelt sich weder um ein eigenständiges Protokoll noch um eine eigene Art von Nachricht, wie es beispielsweise beim zuvor erwähnten SOAP-Standard der Fall ist. In der Regel findet die Kommunikation zwischen Services und Clients über das herkömmliche HTTP-Protokoll statt [FLD00, RO08].

Bei der Umsetzung eines Webservice mittels des REST-Prinzips dreht sich alles um sogenannte Ressourcen. Eine Ressource steht hierbei immer für einen Ansprechpunkt für den Client, also eine Möglichkeit, mit dem Server zu interagieren oder Informationen abzurufen. Eine Ressource kann alles sein, was bedeutend genug ist, um eine eigene Referenz zu bekommen. Eine Zusammenfassung dafür, was als Ressource betrachtet werden kann, liefert folgendes Zitat [RR07]:

> *If your users might 'want to create a hypertext link to it, make or refute assert ons about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it', then you should make it a resource.*

Ressourcen

Damit eine Ressource als solche in einen RESTful-Webservice eingegliedert werden kann, sollte diese einige grundlegende Anforderungen erfüllen. Hierzu zählen [RR07, PZ+08, SA10]:

- *Adressierbarkeit*

  Die eindeutige Adressier- und somit Identifizierbarkeit jeder Ressource muss gegeben sein. Dies geschieht über den bereits angesprochenen URI beziehungsweise über eine URL. Ein Projekt mit der Projektnummer 123 könnte also beispielsweise über die URI https://esprint.at/projects/123 adressiert werden. Für die Art der URI-Gestaltung und des Aufrufs dieser Ressource gibt es wiederum verschiedene Ansätze, die später genauer eruiert werden.
- *Zustandslosigkeit*

  Prinzipiell gilt bei REST die Zustandslosigkeit der Kommunikation der Teilnehmer untereinander. Es werden also keine – wie im World Wide Web normalerweise üblichen – Sitzungen in Form von Cookies oder Sessions für den jeweiligen Benutzer angelegt, sondern die notwendigen Informationen müssen

bei jedem Datenaustausch erneut mitgeschickt werden. Im Fall von eSprint bedeutet dies, dass bei jedem Datenaustausch mit dem Server eine Authentifizierung mittels eines Zertifikates stattfinden muss, das gleichzeitig zur Erkennung des jeweiligen Benutzers verwendet wird. Wird zum Beispiel eine Datei geöffnet, bearbeitet und deren neue Version gespeichert, so müssen sowohl beim Vorgang des Ladens als auch des Speicherns alle relevanten Informationen zum Benutzer und über die betreffende Datei an den Server übermittelt werden, damit dieser die Informationen eindeutig zuordnen und verarbeiten kann.

Die Zustandslosigkeit bedeutet zwar einen großen *Overhead*, also viele zusätzlich zur Kommunikation benötigte Daten neben den Nutzdaten, jedoch bietet sie auch Vorteile.

Zum einen ist es zumindest theoretisch möglich, die Last der Anfragen von Clients auf mehrere Server zu verteilen, da ohnehin immer alle relevanten Daten mitgeschickt werden müssen. Der Zusatz „theoretisch" soll darauf hindeuten, dass dies beim vorgestellten eSprint-System nicht benötigt wird, da der Rechenaufwand zu einem großen Teil auf Seiten der Clients stattfindet und somit eine Entlastung der Serverinfrastruktur überflüssig macht. Zum anderen bieten sich durch die Zustandslosigkeit Vorteile hinsichtlich sicherheitstechnischer Aspekte. So erschwert das Wegfallen einer Session-Variable – wie sie etwa bei klassischen Client-Serververbindungen genutzt wird – es einem möglichen Angreifer, sich unbefugten Zugriff auf sensible Daten durch das Auslesen und Nutzen einer solchen zu verschaffen.

- *Standardisierte Schnittstellen*

  Bezugnehmend auf die zuvor angesprochenen HTTP-Methoden, auf denen RESTful-Services aufbauen, ist dies die dritte Anforderung an die zur Verfügung gestellte Ressource. Mittels einer oder mehrerer Standardmethoden POST, GET, PUT, CONNECT etc. muss jede angebotene Leistung eines Services durch den Client ansprechbar sein.

- *Repräsentation und Ressourcen*

  Ressourcen können bei herkömmlichen RESTful-Services dem Nutzer auf verschiedene Arten zur Verfügung gestellt werden. Diese Repräsentationen ein

und derselben Ressource können etwa im XML-, JSON-, aber auch in einem einfachen Plain-Text-String-Format vorliegen. Hierdurch ermöglicht der Serviceanbieter seinen Nutzern ein breitgefächertes Spektrum an Verarbeitungsmöglichkeiten der angebotenen Daten und dadurch eine Interoperabilität. Beim eSprint-System ist das Anbieten in mehreren Formaten nicht notwendig, da die auf die Ressource zugreifenden Clients alle über eine einheitliche und nicht veränderbare Datenverarbeitung zurückgreifen. Dies geht mit dem Umstand der aufwändigen Datenverschlüsselung zur Sicherstellung höchstmöglicher Vertraulichkeit der Daten einher.

Standards

Bei der Entwicklung von REST wurde Wert auf die Verwendung von Web-Standards gelegt. Zum einen wurde diesem Grundsatz durch die Tatsache Rechnung getragen, dass die Ansprechbarkeit der Services über die üblichen HTTP-Methoden zu erfolgen hat und zum anderen können die zuvor angesprochenen Ressourcen und Repräsentationen in einem beliebig verarbeitbaren Format vorliegen. Letztere Aussage umfasst ebenfalls zukünftige Formate und Technologien, was REST zu einer sicheren Entscheidung macht.

Demgegenüber steht die SOAP-Technologie, die vorwiegend auf eigens geschaffene Standards wie WSDL, WS-Security, WS-Transaction, UDDI etc. setzt. Diese Liste hat sich in der Entwicklung von SOAP immer mehr erweitert und weist mittlerweile eine beachtliche Menge an Umsetzungsmöglichkeiten auf. Dies bedeutet zwar eine bessere individuelle Anpassbarkeit, jedoch steht es konträr zu dem Gedanken und den damit einhergehenden Vorzügen einer generischen Webarchitektur mit allgemein bekannten und gültigen technischen Übereinkünften.

Umsetzung des Webservices – SOAP vs. REST

Da das Ziel dieses Projektes unter anderem die Umsetzbarkeit eines neuen Verschlüsselungsmodells ist, liegt der Schwerpunkt auf der Entwicklung, die Implementierung des übrigen Systems möglichst einfach zu halten. Da es vorrangig um einen effizienten und auf die Anforderungen anpassbaren Webservice ging, fiel die

Wahl auf REST. Diese Entscheidung wurde unter anderem auf Grundlage der nachfolgenden Erörterungen getroffen: [PZ+08, RR07]

- *Frei verfügbar*

  Da REST auf keiner speziell entwickelten Technologie basiert, sondern auf bestehenden Technologien aufbaut, fallen keine zusätzlichen Gebühren zum Betrieb eines solchen Systems an. Es bedarf also keiner speziellen Tools oder teuren Software, um ein Webservice mittels REST umzusetzen.
- *Flache Lernkurve*

  Bis zur tatsächlichen Umsetzung eines REST-basierten Webservices müssen nicht zahlreiche Standards und Implementierungsrichtlinien studiert werden. Es können mit relativ geringem Aufwand in kurzer Zeit leistungsfähige und individuell angepasste Webservices entworfen und implementiert werden, die im Falle von eSprint alle Anforderungen erfüllen.
- *Hohe Effizienz*

  Im Vergleich zu SOAP setzt REST auf bestehende Webtechnologien und ist individuell anpassbar. Zur Übermittlung eines einfachen Strings muss nicht zwangsläufig eine komplette XML-Nachricht zusammengestellt werden. Dadurch konnte bei eSprint der Overhead der einzelnen Nachrichten deutlich verringert werden, was im Falle der eingesetzten Verschlüsselungstechniken zur Reduzierung der Übertragungs- sowie Ver- und Entschlüsselungszeiten geführt hat. Dieses Themengebiet würde jedoch im Zuge dieser Arbeit zu weit führen und wird außerdem durch die Dissertation von Frau Nisreen Alam Aldeen abgedeckt.
- *State of the Art*

  Dieser Punkt stellt eher einen philosophischen als einen rationalen Entscheidungspunkt dar. Da sich der RESTful-Webservice näher an den tatsächlichen Webtechnologien bei seiner Umsetzung orientiert, ist er eine geeignete Wahl bei der Umsetzung eines Webservices. Er verzichtet auf die Etablierung weiterer Pseudo-Standards und stellt die Idee einer umfangreichen und zugleich problemlos realisierbaren Technologie in den Mittelpunkt.

5 Security-Management

Ein bedeutendes Element in Computer-Security-Services und Applikationen ist die Verwendung von kryptografischen Algorithmen. Dies gilt in besonderem Maße auch für unsere eSprint-Applikation und ihre Kommunikation mit dem dazugehörigen Service. In diesem Kapitel soll daher zuerst ein Überblick über die symmetrische Verschlüsselung und die verschiedenen Typen von Algorithmen sowie ihre Einsatzgebiete gegeben werden. Im Anschluss werden die Hash-Funktionen und ihre Bedeutung in der Authentifizierung von Nachrichten betrachtet. Weiters wird die asymmetrische Verschlüsselung (auch als Public-KeyVerschlüsselung bezeichnet) behandelt, die auch für die späteren Kapitel über TLS und HTTPS von Relevanz ist.

5.1 Symmetrische Verschlüsselung

Die universelle Technik zur Gewährleistung der Vertraulichkeit für zu übertragende oder gespeicherte Daten ist die symmetrische Verschlüsselung. Hier werden zunächst das Grundkonzept dieser Verschlüsselungsart sowie die zwei wichtigsten symmetrischen Verschlüsselungsalgorithmen, der Data-Encryption-Standard (DES) und der Advanced-Encryption-Standard (AES) erörtert. Anschließend wird das Konzept der symmetrischen Stream-Verschlüsselung-Algorithmen behandelt.

Symmetrische Verschlüsselung

Diese Art der Verschlüsselung wird auch oft als herkömmliche Verschlüsselung oder Single-Key-Verschlüsselung bezeichnet und war bis zur Einführung der Public-Key-Verschlüsselung in den späten 1970er Jahren die einzig bekannte Art der Verschlüsselung. Bereits Julius Cäsar oder die deutsche U-Boot-Flotte im Zweiten Weltkrieg haben sich diese Methode zunutze gemacht, um ihre Kommunikation zu verschlüsseln. Auch heute noch ist sie die am häufigsten verwendete Art der Verschlüsselung.

Im Gegensatz zur asymmetrischen Verschlüsselung braucht die symmetrische Verschlüsselung nur einen Schlüssel, der von allen Beteiligten gekannt werden muss. Dieser Schlüssel muss allerdings zuvor unter den Akteuren ausgetauscht werden, was

in der Praxis ein Problem darstellt. Insgesamt muss die symmetrische Verschlüsselung die folgenden fünf Eigenschaften aufweisen [SB12, ST13, HC17]:

- *Klartext*
  Dies sind die Originaldaten, die dem Verschlüsselungs-Algorithmus als Input mitgegeben werden.
- *Verschlüsselungs-Algorithmus*
  Der Verschlüsselungs-Algorithmus führt verschiedene Transformationen und Substitutionen mit dem Klartext durch.
- *Sicherheitsschlüssel*
  Dieser dient ebenfalls als Input für den Verschlüsselungs-Algorithmus. Die dort stattfindenden Operationen hängen direkt vom Sicherheitsschlüssel ab.
- *Verschlüsselter Text*
  Dies ist der unleserlich oder durcheinandergebrachte Text und Output der Verschlüsselung. Der verschlüsselte Text hängt direkt mit dem Klartext und dem verwendeten Schlüssel zusammen. Derselbe Klartext mit zwei unterschiedlichen Schlüsseln würde beispielsweise zu zwei unterschiedlichen Ergebnissen, also different verschlüsselten Texten, führen.
- *Entschlüsselungs-Algorithmus*
  Dieser ist essentiell zur Dechiffrierung des Textes. Mithilfe des verschlüsselten Textes und des Keys wird wieder der Klartext hergestellt.

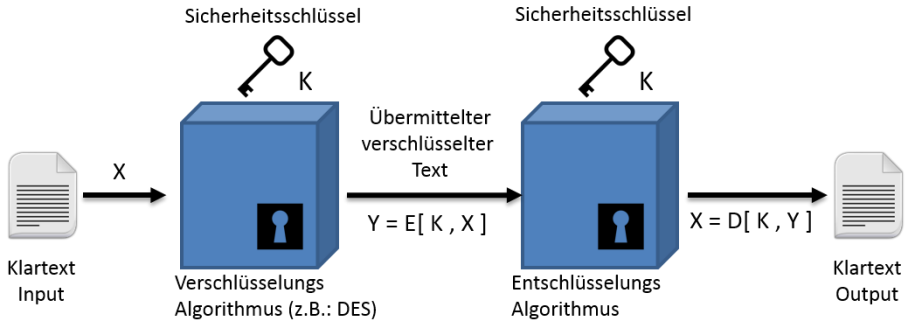

Abbildung 16: Simples Modell der symmetrischen Verschlüsselung

Es gibt zwei Anforderungen für eine sichere Verwendung der symmetrischen Verschlüsselung:

1. Es wird ein starker Verschlüsselungs-Algorithmus benötigt. Die Mindestanforderung ist, dass ein Gegner, selbst wenn er den Algorithmus kennt und auch Zugriff auf einzelne oder mehrere verschlüsselte Texte hat, nicht in der Lage ist, diese zu entschlüsseln oder den Sicherheitsschlüssel herauszufinden. Eine noch verstärkte Form dieser Anforderung: Ein Angreifer soll selbst dann nicht in der Lage sein, den Text zu entschlüsseln beziehungsweise den Sicherheitsschlüssel zu erkennen, wenn er verschlüsselte Texte und dazu passende entschlüsselte Texte hat.
2. Der Sender und Empfänger einer Nachricht müssen den Sicherheitsschlüssel sicher verwahren. Ist dieser Dritten bekannt, kann jede Nachricht mit der Kenntnis des Algorithmus lesbar gemacht werden.

In der in Abbildung 16 gezeigte Notation Y = E [K, X] steht E für den Verschlüsselungs-Algorithmus, der den Klartext (X) mithilfe des Schlüssels (K) verschlüsselt und so ein für Dritte nicht mehr lesbares Ergebnis (Y) erzeugt. Die Entschlüsselung X = D [K, Y] ist der gegensätzliche Prozess. Hier verarbeitet der Entschlüsselungs-Algorithmus (D) den zuvor verschlüsselten Text (Y) mithilfe desselben Schlüssels (K) in seine ursprüngliche Klartext-Form (X).

Wie zu erkennen ist, nimmt der Verschlüsselungs-Algorithmus die zentrale Rolle in diesem Prozess ein. Verschlüsselungs-Algorithmen werden in zwei übergeordnete Gruppen unterteilt: Blockchiffren und Stromchiffren [SB12, SC10].

5.1.1 Blockchiffren

Blockchiffren teilen die zu verschlüsselnde Nachricht in Blöcke fester Längen. Jeder Block wird unter Verwendung des Schlüssels einzeln chiffriert. Typische Werte für die Blockgröße sind hierfür 64, 128 oder 256 Bit. Falls die Länge der Nachricht kein Vielfaches dieser Werte ist, muss die Nachricht mithilfe eines festgelegten Verfahrens

(Padding) aufgefüllt werden. Die bekanntesten Blockchiffren sind der DES und sein Nachfolger AES [SC10].

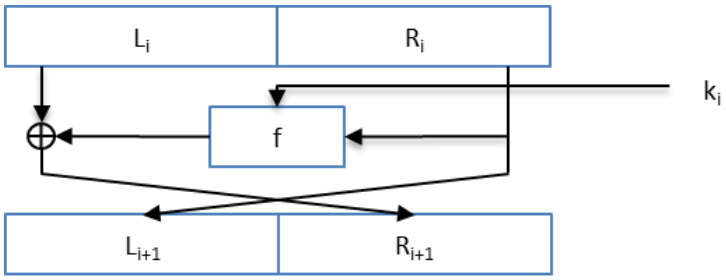

Abbildung 17: Feistel-Grundstruktur des DES-Algorithmus

Data-Encryption-Standard

Der DES ist bis heute ein weitverbreiteter Verschlüsselungs-Algorithmus. Dieser durch die National Security Agency (NSA) überprüfte und durch das National Institute of Standards and Technology (NIST) 1976 zum Standard erklärte Algorithmus wurde durch ein Team rund um Horst Feistel und Don Copperschmidt in der Firma IBM entwickelt. Die Zusammenarbeit zwischen IBM und der NSA gilt noch bis heute als umstritten. Weiters wurde die Schlüssellänge von 128 Bit auf 64 Bit reduziert, wobei 56 Bit die tatsächliche DES-Schlüssellänge bilden und 8 Bits auf sogenannte Parity-Check-Bits fallen, die zur Erkennung fehlerhaft übertragener Informationswörter dienen. Der DES verwendet eine Blockgröße von 64 Bit. Dies bedeutet, dass ein 64-Bit-Block-Klartext in einen 64-Bit-Block-Chiffretext transformiert wird [SC10].

Für die Verschlüsselung erfolgt im ersten Schritt mit einem 64-Bit-Block-Klartext eine initiale Permutation. Nach dieser wird, dargestellt in der Ferstl-Grundstruktur in Abbildung 17, der 64-Bit-Block in zwei Teile aufgeteilt. In jeder Runde werden die linke und rechte Hälfte der Daten vertauscht, wobei die linke Hälfte zuvor mit der rechten Hälfte, auf welche die Funktion f angewendet bitweise XOR verknüpft wird. Nach der letzten Runde werden die Hälften wieder zusammengeführt und eine finale Permutation vorgenommen. Eine nicht schnell lösbare Funktion f ist hierbei essentiell für die Stärke des Verschlüsselungs-Algorithmus. Für die Entschlüsselung wird der gleiche Algorithmus verwendet. Es werden lediglich die einzelnen Rundenschlüssel in umgekehrter Reihenfolge genutzt.

Der Nachteil des DES liegt in seiner Schlüsselgröße von 56 Bit. Der DES konnte deshalb schon mittels einer Brute-Force-Attacke[8] gebrochen werden. Die NSA besaß bereits in den 1970er Jahren die nötige Rechnerkapazität, um anhand des Probierens aller möglichen Schlüssel (2^{56} = ca. 72 Billiarden) den Algorithmus zu brechen [SB12, ST16].

Triple-DES

Aufgrund der Angreifbarkeit des DES war dieser schon bald nicht mehr ausreichend. Als erster Nachfolger der DES gilt der Triple-DES-Algorithmus. Die Idee hierbei ist, durch Mehrfachausführung des DES mit drei unabhängigen Schlüsseln den Algorithmus zu stärken. Bei der am häufigsten verwendeten Methode, der Encrypt-Decrypt-Encrypt (EDE), wird zuerst mit einem DES-Schlüssel K_1 chiffriert, dann mit einem Schlüssel K_2 dechiffriert und schließlich nochmal mit einem Schlüssel K_3 chiffriert. Aufgrund verschiedener möglicher Angriffe auf Triple-DES, wie die Man-in-the-Middle-Methode, wurde der Algorithmus von der NSA nur mit einem Sicherheitsniveau von 80 Bits bewertet [SC10].

Advanced-Encryption-Standard

Im Oktober 2000 veröffentlichte das NIST den neuen Verschlüsselungs-Standard AES. Dieser wurde unter anderem von Joan Daemen und Vincent Rijmen entwickelt, weshalb er auch Rijndael-Algorithmus genannt wird. In einem von dem NIST ausgeschriebenen Wettbewerb wurde der Rijndael-Algorithmus gegenüber anderen Verschlüsselungs-Algorithmen (Twofish, RC6, Serpent, MARS) aufgrund seiner Einfachheit und Performance leicht modifiziert zum AES-Standard ausgewählt. Das Verfahren arbeitet mit einem Datenblock, der die Daten und einen Cipher-Block, in dem sich der Schlüssel befindet, enthält. Die Blockgröße des AES ist mit 128 Bit fest gewählt. Die Schlüssellänge kann 128, 192 oder 256 Bit betragen. Der originale Rijndael hingegen hat eine variable, voneinander unabhängige Block- und Schlüssellänge von 128, 160, 192, 224 und 256 Bit ermöglicht. Die Schlüssellänge bestimmt auch, wie oft der Algorithmus den Daten ausgesetzt wird. Bei einem 128-Bit-

[8] Bei der Brute-Force-Methode werden alle möglichen Schlüssel-Kombinationen durchlaufen

Schlüssel wird der Algorithmus 10-mal, bei 192-Bit-Schlüssel 12-mal und bei 256-Bit 14-mal durchgeführt. Während des Verschlüsselungsverfahrens werden wiederholt vier verschiedene Funktionen durchlaufen (SubBytes, ShiftRows, MixColumn und AddRoundKey). Ein kompletter Durchlauf dieser Schritte wird als Runde bezeichnet. In der letzten Runde wird die Funktion MixColumn nicht mehr ausgeführt. Die einzelnen Runden verwenden jeweils einen eigenen Schlüssel. Dieser berechnet sich aus dem vorherigen Cipher-Block (KeySchedule).

Für die Entschlüsselung wird der Algorithmus rückwärts ausgeführt, wobei die anschließend erklärte Substitutionstabelle gegenläufig aus der Verschlüsselungs-Substitutionstabelle berechnet und angewandt wird. Die Zeilenverschiebung, ShiftRows, wird nach rechts und nicht nach links durchgeführt und die Subschlüssel werden in umgekehrter Reihenfolge angewandt [SB12, ST13].

- AddRoundKey

  Vor der eigentlichen Verschlüsselung werden der Datenblock und der Cipher-Block ähnlich des DES-Verfahrens bitweise miteinander verknüpft. Wie in Abbildung 18 zu sehen ist, wird jede Zelle im Datenblock (blau) und Cipher-Block (rot) durch XOR verknüpft und das Ergebnis in den neuen Datenblock eingetragen.

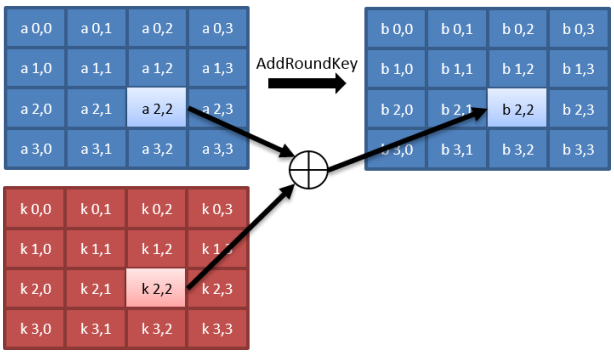

Abbildung 18: AddRoundKey

- SubBytes

  Dieser Schritt der Verschlüsselung ersetzt jedes Byte im Datenblock mit einem anderen fest definierten Byte. Jedes Zeichen wird einzeln betrachtet und mit

einem anderen Hexadezimalwert aus der Substitutionstabelle ersetzt, also monoalphabetisch verschlüsselt. Für einen Beispielwert 19 im Datenblock bestimmt 1 die Zeile und 9 die Spalte in der Substitutionstabelle (siehe Abbildung 19). Der der dort gefundene Wert 2A ersetzt 19.

hex		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	19	52	56	6E	9B	1C	93	54	1D	BF	85	D7	9D	E5	78	9E
	1	DC	EC	30	E4	AB	B7	7B	70	3D	2A	A2	6D	96	E4	84	91
	2	6E	5D	E4	B	8D	E4	BC	A	AF	92	B5	75	46	71	43	1B
	3	C2	1C	5E	C4	F9	F6	A4	F1	8B	95	AA	D2	AC	4	ED	5
	4	EA	8D	32	A9	92	8D	98	72	63	6	F0	F4	77	63	2B	1
	5	4F	95	42	E7	F3	CD	65	100	E8	49	BC	29	A	29	F9	C
	6	5F	BC	65	98	AB	BB	CB	A1	9F	89	5	72	66	61	37	5D
	7	1F	9A	6F	34	A8	FC	73	6F	DF	62	2A	73	D9	D6	2C	100
	8	43	32	E8	F7	59	AD	57	C8	89	AA	CE	90	F2	91	41	BF
	9	C2	EE	38	56	A2	26	76	92	4E	FD	CE	9F	EA	5D	CE	DA
	A	8E	60	43	6B	4F	3	57	42	10	AE	50	F0	2C	45	6B	93
	B	AC	26	BD	3C	72	F5	44	79	AB	91	99	3A	D9	7C	4C	F4
	C	9	B7	46	F2	6F	AC	F	16	DE	90	B0	D9	57	60	AD	EC
	D	71	C1	5A	B2	31	3B	18	EA	5B	2B	3D	35	51	F6	20	E7
	E	C	50	B8	73	34	E4	55	78	66	6C	23	35	AF	56	2F	9A
	F	7D	BD	30	33	37	5F	18	5D	CC	B7	9B	EB	77	2D	DD	3F

Abbildung 19: SubBytes – Substitutionstabelle

- ShiftRows

  Der nächste Verschlüsselungsschritt innerhalb jeder Runde ist ShiftRows. Die Zeilen innerhalb eines Datenblocks werden mit Ausnahme der ersten nach links verschoben. Wie in Abbildung 20 erkennbar ist, wird die zweite Zeile um eine Stelle verschoben, die dritte um zwei Stellen, die vierte um drei Stellen. Für überlaufende Stellen wird die Zeile von rechts fortgesetzt.

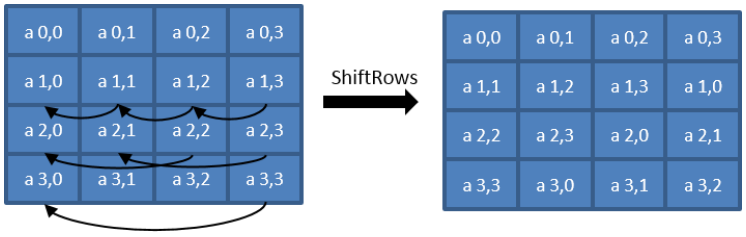

Abbildung 20: ShiftRows

- MixColumn

  Der finale Schritt jeder Runde, mit Ausnahme der letzten, ist der MixColumn. Wie in Abbildung 21 zu sehen ist werden hierbei die Daten mithilfe eines komplexen mathematischen Verfahrens, des sogenannten Galois-Feldes, innerhalb der Spalten vermischt. Im Folgenden wird die Grundidee des MixColumn zusammengefasst. Als Erstes werden alle Einträge der Spalten mit Konstanten polynommultipliziert. Nach dieser Multiplikation wird das Ergebnis jeder Spalte mit XOR verknüpft und in den neuen Datenblock gestellt. Die Reihenfolge der Konstanten ändert sich dann für die nächsten Spalten. Dies wird je nach Stelle im neuen Datenblock festgelegt. Nach vollständiger Befüllung des Datenblocks ist das Verfahren abgeschlossen.

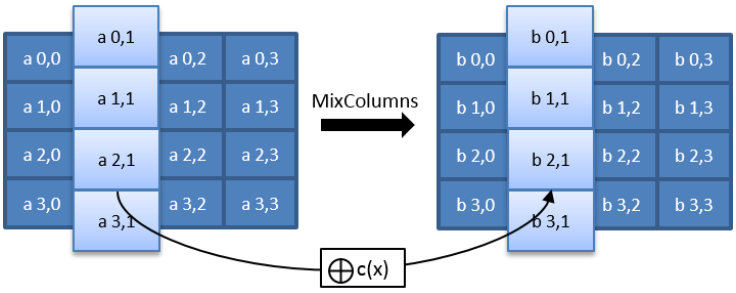

Abbildung 21: MixColumns

- KeySchedule

  Wie eingangs erwähnt, verwenden die einzelnen Runden der Ver- beziehungsweise Entschlüsselung jeweils einen eigenen Schlüssel. Hierzu dient das KeyScheduling. Der Userschlüssel, der sich im 128-Bit-Cipher-Block befindet, fungiert als Basis. Eine Erweiterung dieses Schlüssels wird in 16-Byte-große Teile auf die einzelnen Runden aufgeteilt. Je nach Bit-Größe des Datenblocks wird die Anzahl der Runden beziehungsweise Schlüssel bestimmt (10, 12 oder 14). Für die Erweiterung des Userschlüssels werden die zwei Funktionen SubWord, RotWord sowie eine Konstante Rcon angewendet. SubWord ersetzt, genau wie die zuvor erwähnte Funktion SubBytes, jede Zelle

des Userschlüssels mittels der Substitutionstabelle (siehe Abbildung 19). Die zweite Funktion RotWord verschiebt dann die einzelnen Spaltenwerte um Eins nach oben. Als Letztes wird die 4-Byte-große Konstante Rcon auf das erste Byte der Spalten des Cipher-Blocks angewendet, wie in Abbildung 22 veranschaulicht. Jede neue Spalte, die Rcon liefert, wird mit der vorher neu berechneten Spalte XOR verknüpft. Der Vorgang ändert sich allerdings für den ersten Durchlauf, in dem die erste Spalte des neuen Cipher-Blocks berechnet wird. Hier werden die letzte und die erste Spalte des alten Cipher-Blocks für die Berechnung herangezogen. Zuerst wird auf die letzte Spalte RotWord angewandt und anschließend mit SubWord substituiert. Das Ergebnis wird anschließend mit der unveränderten ersten Spalte und mit der von Rcon gelieferten Spalte XOR verknüpft und bildet so die erste Spalte des neuen Subschlüssels. Für die weiteren Spalten werden die Spalte des alten Cipher-Blocks und die vorherige Spalte im Subschlüssel XOR verknüpft und in die neue Spalte des Subschlüssels geschrieben [FI08, SB12].

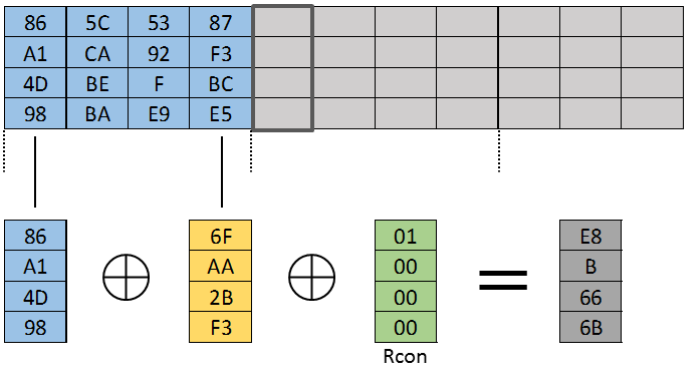

Abbildung 22: KeySchedule – Berechnung der ersten Spalte

5.1.2 Stromchiffren

Eine Blockchiffre produziert für jeden Input-Block einen Output-Block. Eine Stromchiffre verarbeitet die Input-Elemente hingegen kontinuierlich und produziert so ein Element nach dem anderen. Blockchiffren sind verbreiteter, doch Stromchiffren sind für manche Applikationen auch eine geeignete Wahl.

Eine typische Stromchiffre verschlüsselt ein Byte nach dem anderen. Abbildung 23 repräsentiert eine Stromchiffre-Struktur. In dieser Struktur erhält ein Pseudorandom-Byte-Generator einen Key als Input und produziert eine zufällige Zeichenfolge, die Keystream genannt wird. Dieser wird dann mit dem Klartext bitweise XOR verknüpft. Mit einem gut designten Pseudorandom-Byte-Generator kann eine Stromchiffre genauso sicher wie eine Blockchiffre gleicher Schlüssellänge sein. Allgemein ist festzuhalten, dass Stromchiffren besser geeignet sind für Applikationen, welche die Verschlüsselung von Datenstreams verlangen, also Applikationen, die einen länger bestehenden Kommunikationskanal oder einen Browser-Weblink verwenden. Block-Cipher hingegen sind besser geeignet für den Transfer von Daten, E-Mails oder Datenbanken. Beide können aber für jegliche Applikationen angewendet werden. Da die eSprint-Applikation Dateien fixer Größe verschlüsselt, wird hierfür der Blockchiffre AES verwendet [SB12, ST13].

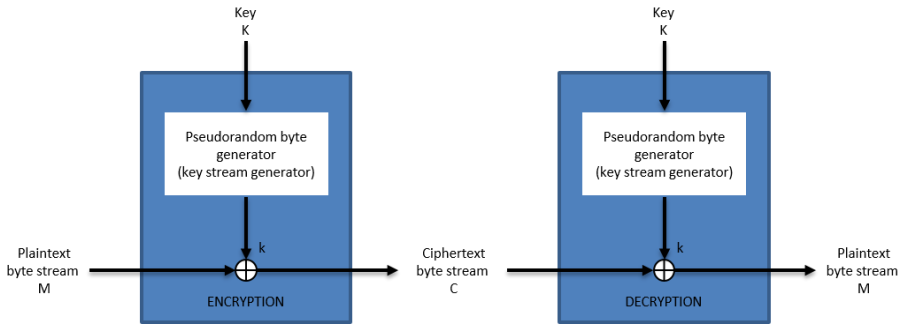

Abbildung 23: Stromchiffre

5.2 Secure-Hash-Funktionen

Ein Hash oder *message digest* ist eine Zahl, die aus einem Text oder Wort generiert wird. Diese Zahl ist in der Regel kürzer als der Text selbst und wird durch eine Formel beziehungsweise einen Algorithmus gebildet. Diese Formel ist so gewählt, dass die Wahrscheinlichkeit, dass ein anderer Text denselben Hash-Wert generiert, möglichst gering ist. Innerhalb von Computer-Security-Systemen werden Hash-Werte dazu verwendet, die übermittelten Nachrichten auf ihre Authentizität zu überprüfen. Als

authentisch gilt eine Nachricht, wenn diese unverfälscht ist und vom richtigen respektive erwarteten Absender kommt.

Abbildung 24 zeigt Hash-Funktionen vereinfacht in einem Blockdiagramm. In diesem können folgende elementare Attribute einer Hash-Funktion erkannt werden. Eine Hash-Funktion H akzeptiert eine variabel lange Eingangsnachricht M und produziert daraus einen Hash-Wert H (M) oder *message digest* fixer Länge. Typischerweise ist dieser Hash-Wert ein Vielfaches von 1024 Bit und beinhaltet sowohl den durch den Hash-Algorithmus erzeugten Wert als auch die Länge der ursprünglichen Nachricht. Das Festhalten der ursprünglichen Textlänge ist eine Sicherheitsmaßnahme und dient der Erschwerung eines Angriffs durch Dritte. Vorrangig soll so gewährleistet werden, dass keine zweite Nachricht mit demselben Hash-Wert, aber einer anderen Nachricht erzeugt werden kann [SB12, RH18].

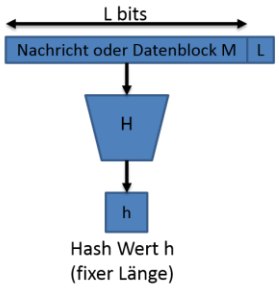

Abbildung 24: Blockdiagramm eines Secure-Hash-Algorithmus h=H (M)

Eine für die Kryptographie verwendete Hash-Funktion muss also den folgenden Kriterien entsprechen [SB12, ST13]:

- geringe Kollisionswahrscheinlichkeit der Hash-Werte für die Inputdaten, also eine möglichst gute Verteilung der Hash-Werte auf die erwarteten Eingangswerte
- Der Hash-Wert soll deutlich weniger Speicherbedarf als die Eingangsnachricht benötigen.
- Ähnliche Eingangsnachrichten müssen verschiedene Ergebnisse liefern. Das Ändern eines einzelnen Bits in der Eingangsnachricht sollte durchschnittlich die Hälfte aller Bits in der Ausgangsnachricht verändern.

- Alle möglichen Ergebniswerte sollten auch tatsächlich nach Durchlauf des Algorithmus vorkommen können.
- Der Algorithmus sollte effizient sein. Eine Funktion muss also mit geringem Speicherbedarf auskommen und schnell berechenbar sein.

In Abbildung 25 wird die in der vorliegenden Untersuchung angewendete Secure-Hash-Funktion innerhalb der Public-Key-Verschlüsselung betrachtet.

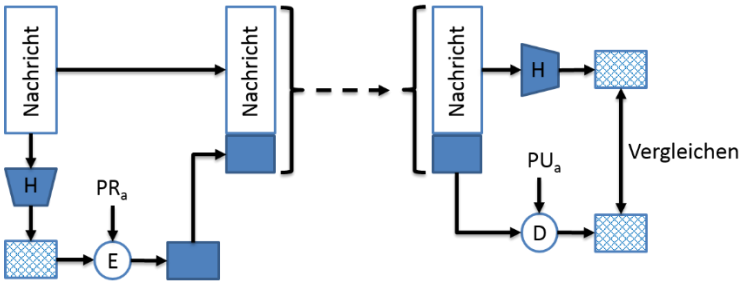

Abbildung 25: Public-Key-Nachrichten-Authentifizierung

Wie bereits in Abbildung 24 erklärt wird, ist bei der Public-Key-Verschlüsselung die zu übermittelnde Nachricht mittels einer Hash-Funktion zu einem Hash-Wert gewandelt. Dieser wird mit dem Private Key des Senders verschlüsselt und zusammen mit der Nachricht an den Empfänger übermittelt. Beim Empfänger wird der verschlüsselte Hash-Wert mittels des Public Keys des Senders entschlüsselt und die übermittelte Nachricht mittels der vereinbarten Hash-Funktion ebenfalls zum Hash-Wert gewandelt. Nun können der selbst erzeugte und der übermittelte Hash-Wert verglichen werden. Stimmen diese überein, kann der Empfänger davon ausgehen, dass die Nachricht unverfälscht und vom erwarteten Empfänger gekommen ist. Diese Verwendung eines Hash-Algorithmus bei der Public-Key-Verschlüsselung hat vor allem zwei Vorteile, denn neben der Nachrichten-Authentifikation dient der Hash-Wert hier auch als digitale Signatur, ohne dass ein Austausch der Schlüssel zwischen den Parteien notwendig wäre.

In der Vergangenheit war der *secure hash algorithm* oder kurz SHA die am häufigsten angewendete Hash-Funktion. Diese wurde vom NIST entwickelt und 1993 publiziert. Nachdem die ersten Schwächen des SHA entdeckt wurden, wurde 1995 eine überarbeitete Version unter dem Titel SHA-1 publiziert. Im Original produziert SHA-

1 einen Hash-Wert mit einer Länge von 160 Bits. 2002 wurden vom NIST drei neue Versionen (SHA-224, SHA-256, SHA-384 und SHA-512) mit den Längen 224 Bits, 256 Bits, 384 Bits und 512 Bits als SHA-2 publiziert. Nachdem für SHA-1 bereits Kollisionsangriffe gelangen, in denen durch Brute-Force-Attacken erfolgreich Hashzwillinge gebildet werden konnten, ist es empfehlenswert, auf Hash-Funktionen der SHA-2- Gruppe oder auf den 2012 publizierten SHA-3-Algorithmus umzusteigen [SB12, ST13].

5.3 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung oder auch Public-Key-Verschlüsselung ist ebenso von Bedeutung wie die symmetrische Verschlüsselung. Sie dient als Anwendung in der Nachrichten-Authentifizierung und im Schlüsselaustausch. An dieser Stelle wird zuerst die Grundstruktur der Public-Key-Verschlüsselung behandelt. Danach folgt ein kurzer Überblick über die wichtigsten asymmetrischen Verschlüsselungsalgorithmen.

5.3.1 Public-Key-Verschlüsselung

Der Diffie-Hellmann-Schlüsselaustausch wurde erstmals 1976 von Whitfield Diffie und Martin Hellmann unter dem Titel „New Directions in Cryptography“ [DIH76] vorgestellt. Vorarbeiten hierzu wurden von Ralph Merkle mit dem nach ihm benannten Merkle‘s Puzzle geleistet, weswegen auch oft vom Diffie-Hellmann-Merkle-Schlüsselaustausch oder auch kurz DHM-Schlüsselaustausch gesprochen wird. Es war das erste der sogenannten asymmetrischen Kryptoverfahren und stellte eine Revolution für Verschlüsselungsverfahren dar. Public-Key-Algorithmen basieren im Gegensatz zu symmetrischen Verfahren auf mathematischen Funktionen und nicht auf simplen Bit-Operationen. Das bedeutendste Merkmal der asymmetrischen Verfahren ist die Verwendung von zwei separaten Schlüsseln, was Veränderungen in den Bereichen der Vertraulichkeit, des Schlüsselaustausches und der Authentifizierung bringt. Symmetrische Verfahren verwenden nur einen Schlüssel. Dies bedeutet aber nicht, dass die Public-Key-Verschlüsselung bisherige symmetrische Verfahren obsolet macht. Beide Arten der Verschlüsselung besitzen ihre Vorteile und werden je nach

Anforderung eingesetzt. Weiters ist zu erwähnen, dass das Konzept der Public-Key-Verschlüsselung nicht automatisch sicherer als eine symmetrische Verschlüsselung ist. Die Sicherheit jedes Verschlüsselungskonzeptes hängt von der Länge des Schlüssels und der benötigten Rechenleistung, um die Verschlüsselung zu brechen, ab.

Die Abbildung 26 und Abbildung 27 zeigen die sechs Bestandteile der Public-Key-Verschlüsselung [SB12, ST13, ST05]:

- *Klartext*
  Dieser ist die lesbare Nachricht, die versendet werden soll und als Input dem Verschlüsselungsalgorithmus mitgegeben wird.
- *Verschlüsselungsalgorithmus*
  Dieser Algorithmus führt diverse Transformationen des Klartextes durch.
- *Public und Private Key*
  Bei diesem Schlüsselpaar wird ein Key für die Verschlüsselung und der andere für die Entschlüsselung des Klartextes verwendet. Die Keys dienen somit genauso wie der Klartext als Input für die auszuführenden Algorithmen.
- *Verschlüsselter Text*
  Der verschlüsselte Text ist die vom Verschlüsselungsalgorithmus als Output produzierte, unlesbar gemachte, zu übermittelnde Nachricht. Diese wird durch die Inputs Klartext und Schlüssel produziert. Zwei unterschiedliche Schlüssel würden also zwei unterschiedlich verschlüsselte Texte produzieren.
- *Entschlüsselungsalgorithmus*
  Dieser nimmt den verschlüsselten Text und den passenden Schlüssel des Schlüsselpaares als Input, um das Original (den Klartext) wiederherzustellen.

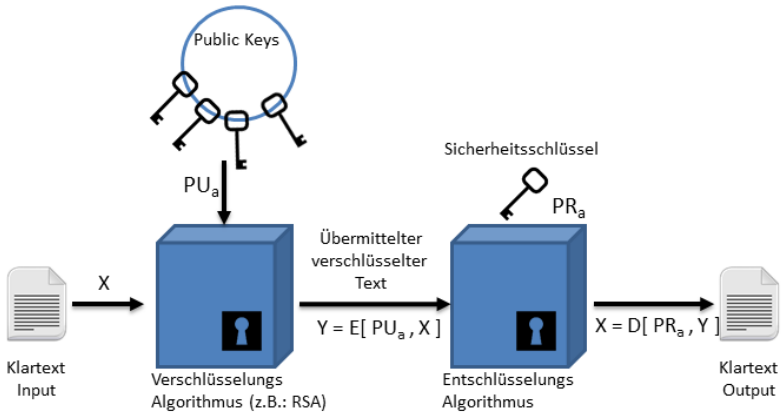

Abbildung 26: Verschlüsselung mit Public Key

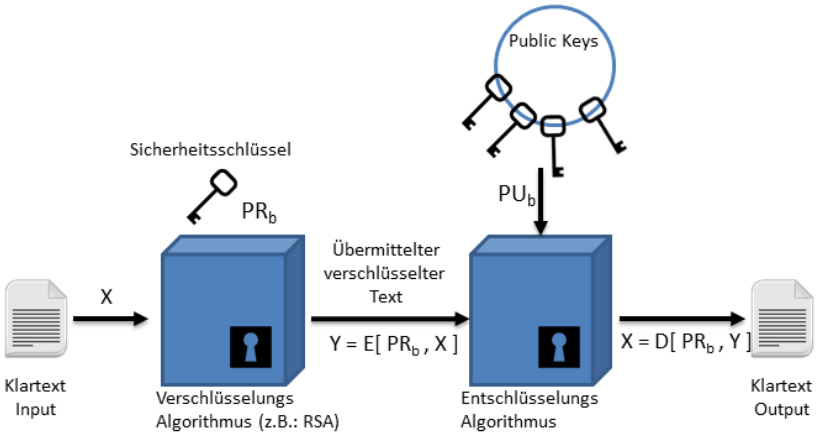

Abbildung 27: Verschlüsselung mit Private Key

Wie die Namen implizieren, ist der Public Key anderen Teilnehmern öffentlich zur Verfügung gestellt, während der Private Key nur dem Besitzer bekannt sein sollte. Das Prinzip der Public-Key-Verschlüsselung ist es, dass je nach Anforderung einer der beiden Schlüssel für die Verschlüsselung und der andere für die Entschlüsselung verwendet wird.

Das Prinzip folgt dabei folgenden Schritten [SB12, SC10]:

1. Beide an einer Übertragung beteiligten Nutzer bilden ein sogenanntes Key-Pair, also ein Schlüsselpaar, das für die Ver- bzw. Entschlüsselung verwendet wird.
2. Jeder Nutzer platziert einen der beiden Schlüssel in ein öffentlich zugängliches Register oder eine andere öffentlich zugängliche Datei. Dieser Schlüssel ist der Public Key. Der zweite Partnerschlüssel wird geheimgehalten und ist der Private Key des Nutzers. Wie in Abbildung 26 zu sehen ist, hält jeder User öffentliche Schlüssel (Public Keys) anderer Benutzer.
3. Möchte nun Nutzer B dem Nutzer A eine private Nachricht senden, verschlüsselt Nutzer B diese Nachricht mit dem Public Key von Nutzer A (PU_a).
4. Wenn Nutzer A die Nachricht erhält, kann er diese mithilfe des eigenen Private Keys (PR_a) entschlüsseln. Kein anderer Empfänger kann diese Nachricht lesbar machen, da nur Nutzer A im Besitz des passenden Schlüssels ist.

Für die Public-Key-Verschlüsselung gilt also, dass alle Zugriff zu anderen Public Keys haben, die dazu passenden Private Keys aber von jedem Nutzer selbst gehalten werden und somit nie übermittelt werden müssen. Solange ein Nutzer seinen eigenen Schlüssel geheimhält, ist die eingehende Kommunikation sicher. Ein Nutzer kann seine Schlüssel immer neu generieren und den passenden Public Key des Schlüsselpaares an andere Teilnehmer übermitteln. Diese müssen lediglich den neuen Public Key des Nutzers mit dem zuvor gespeicherten Schlüssel austauschen.

Abbildung 27 zeigt eine alternative Anwendung der Public-Key-Verschlüsselung. Hier wird nicht, wie zuvor beschrieben, der Klartext mit dem eigenen Private Key verschlüsselt. Es ist nun also jedem Nutzer, der seinen Public Key besitzt, möglich, diesen wieder zu entschlüsseln.

Wird die Abbildung 27 mit der Abbildung 26 verglichen, ist zu erkennen, dass die beiden unterschiedlichen Methoden verschiedenen Anforderungen an eine Kommunikation entsprechen. Das Ablaufschema aus Abbildung 26 erfüllt die Anforderung der Vertraulichkeit, da durch die Verschlüsselung mit dem Public Key des angedachten Empfängers die versandte Nachricht auch nur dieser lesbar machen kann. Hier hängt die Vertraulichkeit aber auch von der Sicherheit des verwendeten Algorithmus, der Geheimhaltung des Private Keys und der Sicherheit des verwendeten

Protokolls ab, von dem die Verschlüsselungsfunktion ein Teil ist. Abbildung 27 hingegen erfüllt die Anforderungen der Authentizität und/oder der Datenintegrität. Es kann ein Nutzer A die von Nutzer B gesendete Nachricht mit dessen Public Key (PUb) entschlüsseln. Nutzer A kann davon ausgehen, dass nur Nutzer B die Nachricht gesendet haben kann. Eine Änderung des Textes kann also nur von Nutzer B durchgeführt worden sein, da dieser den Text mit seinem Private Key verschlüsselt haben muss [SB12, HC17].

5.3.2 Asymmetrische Verschlüsselungsverfahren

Je nach Applikation verwendet der Sender also den Public und/oder Private Key zur Verschlüsselung der Daten. Allgemein können die Public-Key-Verschlüsselungs-Systeme somit in drei Applikationskategorien zusammengefasst werden: digitale Signatur, symmetrischer Schlüsselaustausch und Verschlüsselung von Sicherheitsschlüsseln. Manche Algorithmen sind für alle drei Bereiche passend, andere aber auch nur für eine oder zwei der Kategorien. Tabelle 4 zeigt einen Überblick über Public-Key-Algorithmen und welche Applikationskategorien diese unterstützen.

Algorithmus	*Digitale Signatur*	*Symmetrischer Schlüsselaustausch*	*Verschlüsselung von Sicherheitsschlüsseln*
RSA	Ja	Ja	Ja
Diffie-Hellman	Nein	Ja	Nein
DSS	Ja	Nein	Nein
Elliptic Curve	Ja	Ja	Ja

Tabelle 4: Public-Key-Applikationen

RSA ist eines der ersten Public-Key-Verfahren und wurde 1977 von Ron Rivest, Adi Shamir und Len Adleman am Massachusetts Institute of Technology entwickelt und 1978 veröffentlicht [RSA78]. Seitdem hat sich das RSA-Verfahren zu dem am häufigsten akzeptiertesten und verwendeten Public-Key-Verschlüsselungsverfahren entwickelt. RSA kann für digitale Signaturen, für Verschlüsselungen sowie für den Schlüsselaustauch genutzt werden. 1977 haben die drei Erfinder des RSA einen verschlüsselten Text veröffentlicht und eine 100-$-Belohnung für dessen Rückführung

in den Klartext versprochen. Dies gelang schließlich nach 8 Monaten Arbeit einer über das Internet zusammenarbeitenden Gruppe 1994, wobei 1600 Computer verwendet wurden [LEU94]. Der Sieger des Wettbewerbs nutzte für die Verschlüsselung einen 428-Bit-Schlüssel. Dieser erfolgreiche Angriff bedeutet allerdings nur, dass für eine sichere Kommunikation die Schlüssellänge erhöht werden muss. Heutzutage gilt eine Schlüssellänge von 1024 Bit als sicher.

Der Diffie-Hellmann-Schlüsselaustausch war, wie schon oben erwähnt, das erste Public-Key-Verfahren [DIH76]. Das Ziel dieses Verfahrens ist es, zwei Nutzern zu ermöglichen, sich auf einen gemeinsamen Schlüssel zu einigen, der einen sicheren Nachrichtenaustauch zwischen diesen Parteien zulässt. Der Algorithmus ist auf den Austausch des Schlüssels beschränkt.

1991 wurde der Digitale-Signature-Standard (DSS) im Federal Information Processing Standard, FIPS PUB 186, veröffentlicht. Vom NIST entwickelt, empfiehlt der DSS den Digitale-Signature-Algorithm (DAS) mit Verwendung des SHA-1 für digitale Signaturen. Der DSS verwendet einen Algorithmus, der explizit für digitale Signaturen designt wurde und somit auch nicht für einen sicheren Schlüsselaustausch oder für eine Verschlüsselung verwendet werden kann.

Eine Vielzahl an Produkten und Standards verwendet heutzutage den RSA-Standard für die Public-Key -Verschlüsselung und für digitale Signaturen. Die Bit-Länge für einen sicheren RSA ist aber gestiegen und hat somit die Rechenlast erhöht. Dies beeinträchtigt vor allem E-Commerce-Seiten, die zahlreiche sichere Transaktionen ausüben müssen. Aus diesem Grund ist die Elliptic Curve Cryptography (ECC) als Alternative geeignet. Im Vergleich zu RSA bietet ECC vergleichbare Sicherheit mit einer deutlich kleineren Bit-Länge. Nachdem nun immer mehr Produkte auftauchen, die ECC verwenden, steigert sich auch das Interesse, ihre Schwachstellen auszutesten. Auch deswegen vertraut die Community weiterhin eher dem schon mehrfach geprüften RSA [SB12, ST16].

5.4 Secure Sockets Layer (SSL) und Transport Layer Security (TLS)

Eines der am häufigsten verwendeten Security-Services ist der Secure Sockets Layer (SSL) und der Nachfolge-Internetstandard, der Transport Layer Security (TLS).

SSL wurde entworfen, um das darunterliegende Protokoll TCP anzuwenden und einen verlässlichen End-to-end-Sicherheitsservice zur Verfügung zu stellen. SSL ist kein einzelnes Protokoll, sondern besteht aus zwei Schichten von Protokollen, wie in Abbildung 28 ersichtlich ist. Das SSL-Record-Protocol stellt grundlegende Sicherheitsfunktionen für verschiedene höhere Protokolle zur Verfügung, im Speziellen das Hypertext Transfer Protocol (HTTP), das als Übertragungsservice für die Web-Client-/Server-Interaktion dient und über SSL eingesetzt werden kann. Drei höhere Schichten der Protokolle sind als Teil von SSL definiert: das Handshake-Protocol, das Change-Cipher-Spec-Protocol und das Alert-Protocol [SB12, RI14].

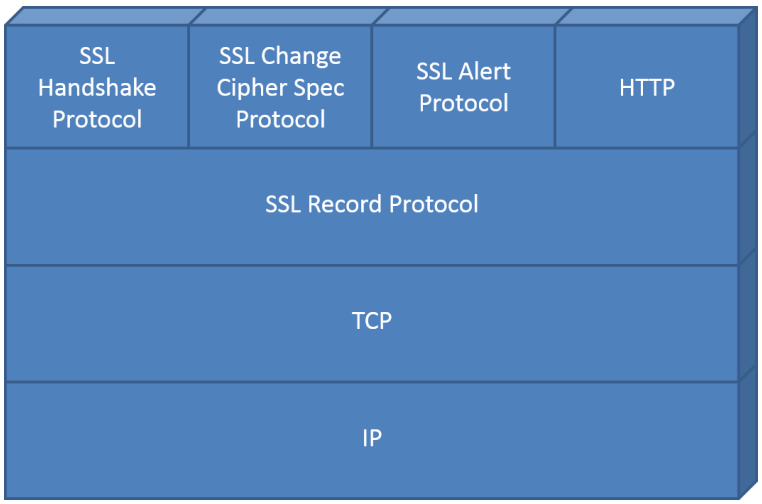

Abbildung 28: SSL-Protocol-Stack

Zwei essenzielle SSL-Konzepte sind die SSL-Session und die SSL-Connection, die in der Spezifikation wie folgt beschrieben sind:

Connection: Eine Connection ist ein Transport (OSI-Schichtenmodell Definition), der einen passenden Servicetyp zur Verfügung stellt. Für SSL sind solche Verbindungen Peer-to-peer-Beziehungen. Diese sind vorübergehend und jede Verbindung ist assoziiert mit einer Session.

Session: Eine SSL-Session ist ein Verbund zwischen einem Client und einem Server. Sessions werden vom Handshake-Protocol erstellt und definieren diverse

Kryptographie-Sicherheits-Parameter, die mit mehreren Verbindungen geteilt werden können. Sie werden verwendet, um die teure Erstellung von neuen Sicherheitsparametern für jede einzelne Verbindung zu vermeiden.

Zwischen zwei Endpunkten (Applikationen wie HTTP auf Client und Server) können mehrere Sicherheitsverbindungen bestehen. In der Theorie könnten auch mehrere simultane Sessions zwischen den Endpunkten existieren, wobei dieses Feature oft nicht verwendet wird.

Das SSL-Record-Protocol stellt zwei Services für SSL-Verbindungen zur Verfügung:

Vertraulichkeit: Das Handshake-Protocol definiert einen geteilten Sicherheitsschlüssel, der für die symmetrische Verschlüsselung von SSL-Nutzdaten verwendet wird.

Nachrichtenintegrität: Das Handshake-Protocol definiert einen geteilten Sicherheitsschlüssel, der zur Erstellung eines *message authentification codes* (MAC) dient.

Abbildung 29 zeigt den gesamten Einsatz des SSL-Record-Protocol. Der erste Schritt ist die Fragmentierung, wobei jede Nachricht in Blöcke mit einer Größe von 214 Bytes (16.384 Bytes) oder weniger aufgespalten wird. Als Nächstes ist eine Kompression optional angefügt. Dann wird in der Verarbeitung ein *message authentification code* über die komprimierten Daten erstellt. Anschließend wird der erstellte MAC zusammen mit den komprimierten Daten mittels eines symmetrischen Schlüssels verschlüsselt. Der finale Schritt des SSL-Record-Protocol-Processing ist das Voranstellen eines Headers bestehend aus den Feldern [SB12, ST13]:

- *Content Type (8 bits)*
  Beinhaltet den Content Type der das höher liegende Protokoll verwendet hat, um das beigefügte Fragment zu verarbeiten.
- *Major Version (8 bits)*
  Diese 8 Bits stehen für die verwendete SSL-Version. Für SSLv3 ist der Wert 3.
- *Minor Version (8 bits)*
  Sie zeigt die verwendete Unterversion.

- *Compresses Length (16 bits)*
  Dies ist die Länge des versendeten Fragments in Bytes. Die maximale Zahl ist 2^{14} + 2048 Bytes.

Die definierten *content types* sind *change_cipher_spec*, *alert*, *handshake* und *application_data*. Dabei handelt es sich bei den ersten drei um SSL-spezifische Protokolle, die als Nächstes betrachtet werden.

Das *record protocol* transferiert die resultierte Einheit in einem TCP-Segment. Empfangene Daten werden entschlüsselt, verifiziert, dekomprimiert und wieder zusammengesetzt und dem höherliegenden User geliefert.

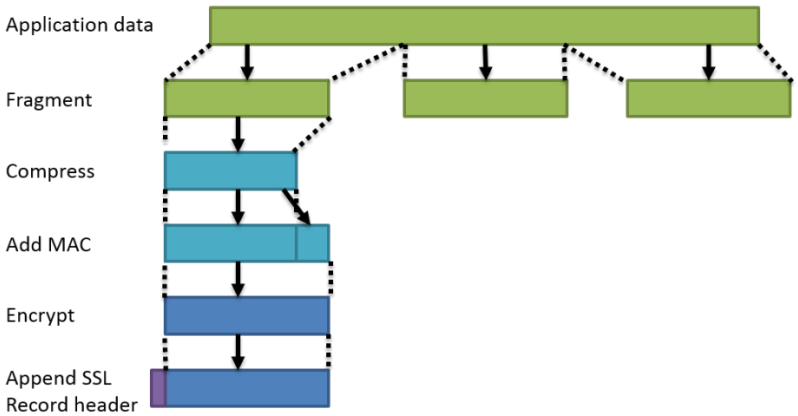

Abbildung 29: SSL-Record-Protocol-Operation

Change-Cipher-Spec-Protocol

Das Change-Cipher-Spec-Protocol ist eines der drei SSL-spezifischen Protokolle, die das SSL-Record-Protocol verwendet und das einfachste Protokoll. Es besteht aus einer einzelnen Nachricht, einem einzelnen Byte mit dem Wert 1. Der Sinn dieser Nachricht ist, den bevorstehenden in den aktuellen Status zu kopieren, der die für diese Verbindung verwendete Verschlüsselungsfolge aktualisiert [SB12].

Alert-Protocol

Das Alert-Protocol wird für die Übermittlung von SSL-bezogenen Alarmsignalen zum Kommunikationspartner verwendet. Die Alarmsignale werden komprimiert und, wie vom aktuellen Status spezifiziert, verschlüsselt.

Jede Nachricht in diesem Protokoll besteht aus zwei Bytes. Das erste Byte kann die Werte Warnung (1) und Fatal (2) annehmen, um die Gewichtung der Nachricht zu übermitteln. Wenn der Wert Fatal einnimmt, terminiert SSL die Verbindung. Andere Verbindungen derselben Session können weiterbestehen, aber es werden keine neuen Verbindungen in der Session aufgebaut. Das zweite Byte beinhaltet einen Code, der die spezifische Alarmierung angibt. Ein Beispiel eines fatalen Alarms ist ein inkorrekter MAC. Ein Beispiel für einen nicht fatalen Alarm (Warnung) ist eine *close_notify*-Nachricht, die dem Empfänger zu verstehen gibt, dass innerhalb dieser Verbindung keine weiteren Nachrichten versendet werden.

Handshake-Protocol

Der komplexeste Teil von SSL ist das Handshake -Protokoll. Es erlaubt dem Server und dem Client, sich gegenseitig zu authentifizieren und sich eine Verschlüsselungs-Methode, einen MAC-Algorithmus und kryptografische Schlüssel auszuhandeln. Diese Parameter werden verwendet, um die geschützten Daten dann in einem SSL-Record zu versenden. Das Handshake-Protokoll wird verwendet, bevor jegliche Applikationsdaten ausgetauscht werden. Es besteht aus einer Serie von Nachrichten zwischen Client und Server. Abbildung 30 zeigt den initial gebrauchten Austausch, um eine Verbindung zwischen Client und Server herstellen zu können.

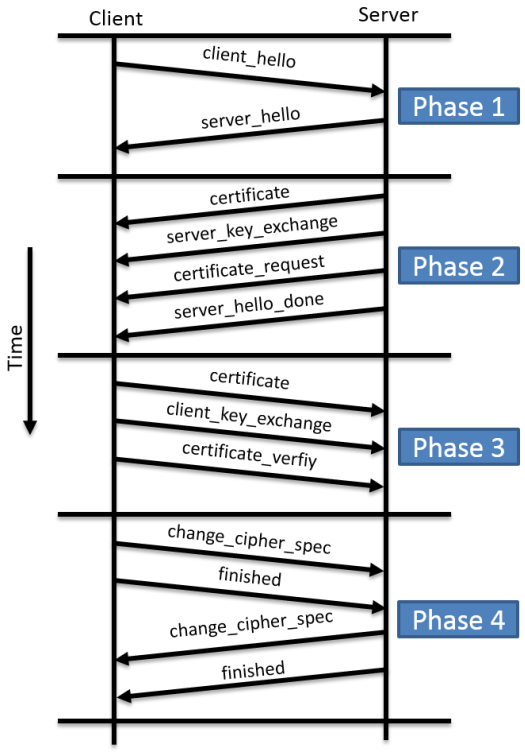

Abbildung 30: Handshake-Protocol-Action

5.5 HTTPS (HTTP over SSL)

HTTPS ist eine Kombination von HTTP und SSL. Das Ziel ist es, die Kommunikation über eine sichere Verbindung zwischen einem Web-Browser und einem Web-Server zu ermöglichen. HTTPS als Funktion ist in allen modernen Web-Browsern integriert. Ihre Verwendung wird von den Fähigkeiten und der Einstellung des Web-Servers bestimmt.

Der für den User ersichtliche Unterschied liegt in der URL: Adressen beginnen mit https:// statt mit http://. Eine normale http-Verbindung verwendet den Port 80, während bei HTTPS Port 443 genutzt wird.

Eine Verwendung von HTTPS bedeutet, dass folgende Kommunikationselemente verschlüsselt sind [SB12, RR07]:

- URL des angeforderten Dokuments
- Inhalt des Dokuments
- Inhalte von Browser-Formularen
- Cookies, die vom Browser an den Server und umgekehrt gesendet werden
- Inhalte des HTTP-Headers

HTTPS wurde dokumentiert im RFC2818, HTTP over TLS. Es besteht kein signifikanter Unterschied bei der Verwendung von HTTP mit SSL oder TLS. Beide Implementierungen werden als HTTPS angesehen [SB12].

Verbindungsaufbau

Für HTTPS agiert der Agent sowohl als HTTP-Client als auch als TLS-Client. Der Client initiiert eine Verbindung zum Server auf den passenden Port und sendet dann ein TLS *client_hello*, um mit einem TLS-Handshake zu beginnen. Nach Vollendung dieses Handshakes startet der Client seinen ersten HTTP-Request. Jegliche HTTP-Daten werden als TLS-Applikationsdaten versendet. Ab hier wird mit Standard-HTTP-Verhalten, inklusive *retained connections*, weiter verfahren.

Es gibt drei zu beachtende Ebenen für Verbindungen mit HTTPS. Auf der Ebene von HTTP fordert ein Client eine Verbindung von einem HTTP-Server durch Senden einer Verbindungsanfrage an die nächstniedrige Schicht. Diese ist typischerweise TCP, aber kann auch TLS/SSL sein. Auf der Ebene von TLS wird eine Session zwischen dem TLS-Server und dem TLS-Client erstellt. Diese kann eine oder mehrere Verbindungen gleichzeitig unterstützen. Wie zuvor behandelt, startet ein TLS-Request mit der Erstellung einer TCP-Verbindung zwischen der TCP-Entität des Clients und der TCP-Entität der Serverseite.

Trennung der Verbindung

Ein HTTP-Client oder Server kann das Schließen einer bestehenden Verbindung durch Senden eines HTTP-Records mit dem Inhalt *connection_close* auslösen. Diese

Nachricht gibt den Hinweis, dass die Verbindung nach dem Versand der Nachricht geschlossen werden soll.

Das Schließen einer HTTPS-Verbindung verlangt zuerst eine Schließung der TLS-Verbindung zwischen TLS-Server und Client, was auch das Beenden der Verbindung auf der Ebene von TCP zur Folge hat. Auf der TLSEbene wird dies ordnungsgemäß mit dem Versand eines *close_notify alerts* durch das zuvor erwähnte TLS-Alert-Protocol bewerkstelligt. Für TLS-Implementierungen müssen die beteiligten Entitäten vor einer Trennung *closure alerts* austauschen. In manchen Fällen schließt eine Entität die Verbindung schon vor der *close_alert*-Rückantwort der anderen. Das wird *incomplete alert* genannt. Dies passiert häufig mit der Absicht, die Session zu einem späteren Zeitpunkt wiederzuverwenden, sollte aber nur dann getan werden, wenn die Applikation weiß (typischerweise durch die HTTP-boundaries), dass bereits alle relevanten Daten angekommen sind.

HTTP-Clients müssen auch mit Situationen umgehen, in denen die darunterliegende TCP-Verbindung ohne ein zuvor gesendetes *close_notify alert* und ohne ein *connection_close* geschlossen werden kann. Mögliche Auslöser hierfür sind Programmfehler am Server oder Kommunikationsfehler, welche die TCP-Verbindung verlieren. Vorsicht ist allerdings geboten, denn eine solche unangekündigte TCP-Verbindungsschließung ist häufig ein Indiz für eine Attacke. HTTPS sollte also für diesen Fall eine Security-Warnung ausgeben.

6 Systemkonzeption

Nach der Vorstellung der notwendigen Grundlagen erfolgt nun die Systemkonzeption des eSprint-Systems. Die technischen Erläuterungen und Ideen der vorangegangenen Kapitel aufgreifend, soll in den nachfolgenden Absätzen die prinzipielle Idee des zu entwickelnden Systems eruiert werden. Es werden ein kurzer Einblick in die Umsetzung des eSprint-Systems gegeben und die wichtigsten Aspekte der Security-Architektur präsentiert.

6.1 Grundlegende Idee

Wie bereits in der Einleitung angesprochen, war eine Voraussetzung bei der Umsetzung dieses Systems der Entwurf eines möglichst einfachen und gewohnten Bedienungskonzepts. Die Einarbeitungsphase für neue Nutzer und Administratoren, aber auch die Usability für erfahrene Anwender, sollten optimiert werden. Aus diesem Grund orientiert sich eSprint bei der Umsetzung von Interface und Handhabung an gängigen Textverarbeitungstools und versucht dabei, etablierte Formatierungsmuster sowie Nomenklaturen einzuhalten. Die Idee dahinter war es, mit einfachen, aber umfangreichen Mitteln die Umsetzbarkeit des Projektes zu prüfen. Es sollte bewiesen werden, dass mittels einer Erweiterung beziehungsweise Anpassung gängiger State-of-the-Art-Technologien eine weitere Stufe der Datensicherheit erreichbar ist.

6.2 Client

Eine weitere Voraussetzung für die Umsetzung des Systems war es, eine plattformübergreifende Arbeitsumgebung zu entwickeln. Um diese Anforderung an das System umsetzen zu können, wurde vorrangig auf die Technologie Java gesetzt. Diese ist auf allen gängigen Endbenutzer-Systemen lauffähig und stellt darüber hinaus alle benötigten Mechanismen zur Realisierung des Projektes zur Verfügung [JCR07].

6.3 Service

Beim Entwurf des Servicedesigns wurde nach einer möglichst leichtgewichtigen Technologie gesucht, die nicht zu viel Overhead bei der Kommunikation erzeugt und zudem eine schnelle und effiziente Implementierung ermöglicht. Weiters waren eine einfache Anpassbarkeit sowie die Option auf eine gute Skalierbarkeit essentiell. Außerdem musste es möglich sein, die geplanten Sicherheitsmechanismen damit umsetzen zu können. Aus diesem Grund fiel die Wahl auf einen RESTful-Webservice, dessen Vorzüge bereits im Kapitel 4.3.3 genauer beleuchtet wurden.

6.4 Datenhaltung

Ebenfalls unerlässlich zur erfolgreichen Implementierung des eSprint-Systems war eine persistente Datenhaltung. Diese musste vorrangig die Aufgabe effizienter Datenzugriffe ermöglichen und zudem größere, aber auch eine hohe Anzahl an Datensätzen speichern können. Es war darüber hinaus bedeutend, dass es sich dabei um eine etablierte Technologie handelt, damit eine einfache zukünftige Erweiterbarkeit durch andere Programmierer gegeben ist.

6.5 Sicherheitsmaßnahmen

Zuletzt soll der wichtigste Punkt des Grundkonzepts vorgestellt werden – das Sicherheitsframework zur Gewährleistung einer durchgehenden Wahrung des Datenschutzes. Hierbei handelt es sich um das Fundament, auf dem das eSprint-System aufgebaut wurde. Mit voranschreitender Technik – Stichwort Quantencomputer – werden selbst bestehende und als sicher geltende Technologien in Frage gestellt. Aus diesem Grund entschieden sich die Autoren der Arbeit, die zur Verfügung stehenden Sicherheitstechnologien auf mehreren Ebenen einzusetzen. Dies umfasst die grundlegende Datenebene, die mehrfach verschlüsselte Übertragungsebene sowie die Datenhaltungsebene. Das eSprint-System lenkt den Fokus also nicht nur auf eine sichere Kommunikation zwischen den Parteien, sondern vielmehr auf die Sicherung der ausgetauschten Ressourcen selbst.

Die Idee dahinter ist, dass selbst bei einer eher unwahrscheinlichen aber dennoch möglichen Kompromittierung der Datenverbindung zwischen den beteiligten Systemen (Client, Server) die Daten noch sicher sind. Auch der Schutz sensibler Daten innerhalb des Netzwerkes soll möglich sein. Dies bedeutet, dass nicht immer alle beteiligten Akteure auch alle Informationen sehen sollen oder dürfen. Diese Art des in der Ressource integrierten Schutzes wird auch als De-Perimeterisation bezeichnet und findet in abgewandelter Form auch in anderen Bereichen Einsatz. Um diese Art des Schutzes der Ressource zu ermöglichen, wird die herkömmliche x.509 Technologie – also die asymmetrische Zertifikatsverschlüsselung – mittels einer symmetrischen Verschlüsselung, des AES, erweitert. Durch das individuell ausgestellte x.509 Zertifikat können somit personalisierte Clients zur Datenverarbeitung realisiert und

durch die Erweiterung des Zertifikates mittels AES-Schlüsseln eine jederzeit anpassbare Zugriffsberechtigung auf die Ressource umgesetzt werden. Dieser Mechanismus ermöglicht eine kontinuierliche Kontrolle der Zugriffsrechte jedes beteiligten Akteurs auf die Ressource. Dieser Ansatz wird im Kapitel 7.4.2 unter dem Begriff Multiple Keys genauer erklärt.

Eine weitere Absicherung gegen Zugriffe durch unautorisierte Dritte stellt die Verschlüsselung der Kommunikation dar. Dies muss auf mehreren Ebenen geschehen, um ein möglichst hohes Maß an Sicherheit gewährleisten zu können. Hierzu ist jedoch – wie bei allen Sicherheitsmechanismen – lediglich ein höchstmögliches Maß anzustreben, da es gerade in der computergestützten Datenverarbeitung niemals eine hundertprozentige Sicherheit geben kann. Aus diesem Grund soll nicht nur ein sicherer Tunnel zwischen dem Server und dem Client (HTTPS), sondern auch eine Verschlüsselung der Daten innerhalb dieses Tunnels implementiert werden. Hierfür werden sowohl symmetrische als auch asymmetrische Verschlüsselungsverfahren angewendet. Darüber hinaus wird mit Signaturen der transferierten Pakete auch eine unveränderte Übertragung eben dieser sichergestellt.

Nicht zuletzt ist ein entscheidender Punkt in Fragen Sicherheit die Datenhaltung. Diese geschieht beim eSprint-System auf mehreren Stufen und trägt maßgeblich zum Schutz der Daten vor unbefugtem Zugriff durch Dritte bei. Die Sicherheitsmaßnahmen beginnen dabei schon bei der Verschlüsselung der sicherheitsrelevanten Daten innerhalb des Datenbanksystems. Hierin werden die notwendigen Schlüssel zur Dechiffrierung der Ressource aufbewahrt. Um diese vor illegalem Zugriff zu schützen, werden die Schlüssel mittels des Public Keys des verwaltenden Servers verschlüsselt. Dieser Public Key wird aus der symmetrischen Verschlüsselung, die in Verbindung mit weiteren Mechanismen auch zur Sicherung der Kommunikation genutzt wird, aufgerufen und verwendet. Die Daten selber werden in Form einer Datei abgelegt, die mittels eines AES-Master-Keys gesamtverschlüsselt werden. Innerhalb der Ressource werden die sensiblen Parts nochmals mittels einzelner Zugriffsrechtsschlüssel ebenfalls im AES-Format chiffriert. Diese somit doppelt mit jeweils 256 Bit verschlüsselte Ressource wird abschließend in Form einer physischen Datei auf dem zugehörigen Datenträger abgelegt. Die Ressource hat dabei die Dateiendung *.spd, die aus

historischen Gründen für diese Untersuchung aus dem Namen des SPIDER-Projektes abgeleitet wurde.

Um nun ein feingranulares Klassifikationsschema innerhalb der Ressource mithilfe der oben angesprochenen Multiple Keys bewerkstelligen zu können, soll der Eigentümer einer Ressource einzelne Textpassagen oder ganze Teile eines Dokumentes mittels dieser Verschlüsselung auf einzelne Benutzergruppen beschränken können. Diese werden zu jedem beliebigen Zeitpunkt durch dazu berechtigte User anpassbar sein. Dieses optional anpassbare Rechtemanagement wird einen wichtigen Teil der Continous-Access-Control darstellen. Hierfür wird es für eine bessere Übersicht und Bearbeitbarkeit der Rechte ein Gruppenverwaltungssystem geben, über das ebenfalls Gültigkeitszeiträume für die Zugriffsbeschränkung eingerichtet werden können. Zusätzlich werden aktive Kontrollen implementiert, die das Problem der simultanen Datenbearbeitung behandeln.

7 Implementierungsansatz

In diesem Kapitel sollen nach der im vorigen Kapitel erläuterten prinzipiellen Auslegung des Projektfokus detaillierte Einsichten in die verwendeten Technologien gegeben werden. Es sollen die grundlegenden Ideen und Entscheidungsfindungen erörtert und ein Überblick über die wichtigsten Umsetzungen dieser präsentiert werden. Nicht zuletzt wird hier bereits der Umriss der restlichen Arbeit gezeigt und anhand von praktischen Beispielen die Vorteile der gefundenen Lösungen hervorgehoben. Zu beachten gilt es jedoch, dass es sich hierbei um keine detaillierte technische Beschreibung der gewählten oder entwickelten Technologien handelt. Diese folgen in den weiteren Kapiteln dieser Arbeit und werden alle relevanten technischen Aspekte beleuchten. Konkret soll gezeigt werden, was eSprint seinem Benutzer für ein Frontend zur Datenmanipulation bietet, wie die Informationen mit dem Server und anderen Nutzern ausgetauscht werden, wie die Daten und nicht nur das darüberliegende System geschützt werden können und abschließend, wie diese Sicherungsmechanismen zielführend umgesetzt wurden. Hierbei wird der Fokus auf den Entwurf einer neuartigen Zertifikatvariation liegen, die dabei hilft, ein größtmögliches Maß an Sicherheit gewährleisten zu können.

7.1 JavaFX

Zum Startzeitpunkt der Entwicklung dieses Projektes waren mehrere Technologien am Markt verfügbar, die zur Darstellung des Frontends geeignet gewesen wären. Es galt also zu evaluieren, welche Anforderungen die Applikation für den Endnutzer erfüllen muss, um eine möglichst effiziente, aber auch übersichtliche Darstellung erzielen zu können.

Die grundlegende Frage ist: Was soll dem Benutzer am Ende präsentiert werden und welche Einschränkungen sind zu erwarten? Eine wichtige Voraussetzung für die Client-Applikation war es, dass sie – zumindest theoretisch – in den meistverbreiteten Systemumgebungen lauffähig sein muss. Es war also essenziell, eine Plattform zu wählen, die flexibel in Bezug auf die Laufzeitumgebung, umfangreich und hochgradig anpassbar hinsichtlich der Präsentationsschicht und nicht zuletzt sicher und auf etablierten Technologien basierend ist. Abgesehen von diesen Anforderungen war es entscheidend, eine eigenständige Anwendung auf Java-Basis zu erstellen. Mit diesem Gedanken geht eine persönliche Präferenz einher, jedoch eignet sich Java/JavaFX auch zu einer durchdachten und erprobten Implementierung von Client-Server-Applikationen und war daher die naheliegendste Wahl.

JavaFX stellt eine der möglichen Technologien zur Darstellung von Benutzeroberflächen mittels Java dar. Nachdem sein Vorgänger Swing nicht mehr offiziell weiterentwickelt wird, ist es die aktuellste Java-basierte Option zur Implementierung von aufwändigen interaktiven Programmen mit grafischer Oberfläche. Zwar befand sich die Technologie zu Beginn dieser Arbeit noch in der Entwicklung, jedoch bot sie die passenden Werkzeuge, um eine umfangreiche Interaktion mit dem Benutzer zu ermöglichen [WIV12].

Die Vorteile, eine Benutzeranwendung mittels JavaFX zu realisieren, sind vielfältig und führten am Ende zu einer positiven Entscheidung für diese Technologie. Zu diesen zählen unter anderem [ORA13]:

- *Verfügbarkeit*

  JavaFX ist, wie bereits oben erwähnt, auf den kommerziell erfolgreichsten Anwender-Betriebssystemen ausführbar. Hierzu zählen Windows, MacOS und

Linux. Dadurch kann eine gute Verfügbarkeit in den meisten Unternehmens-Softwarelandschaften gewährleistet werden.

- *Zukunftssicherheit*

  JavaFX stellt die Nachfolger-Technologie zu Java Swing dar. Diese Plattform wurde über viele Jahre weiterentwickelt und fand breite Akzeptanz unter Entwicklern. Nicht zuletzt deswegen, weil hinter JavaFX die Firma Oracle steht, gibt es eine Garantie für die Zukunft.

- *Community*

  Ebenfalls ausschlaggebend für dieses Projekt war eine aktive Community hinter der gewählten Technologie. Speziell bei den frühen JavaFX-Versionen waren individuelle Anpassungen der Standard-Implementierung notwendig, um zum gewünschten Ergebnis zu gelangen. Hierfür war eine breite Wissensbasis von Vorteil, die durch die frühere Swing- und nunmehr JavaFX-Entwicklergemeinde gewährleistet war und auch weiterhin sein wird.

- *Durchdachte Architektur*

  Im Vergleich zu seinem Vorgänger bietet JavaFX eine wesentlich klarer strukturierte Architektur, basierend auf Java. Hierzu zählen zum Beispiel neu implementierte Möglichkeiten für Styling, Übergangseffekte, Event Management in Verbindung mit JavaScript und WebViews sowie ein funktionierendes Zusammenspiel bei der Client-Server-Kommunikation. Die Option der Einbettung eines WebViews spielte bei der Umsetzung eine besondere Rolle, da auf diese Weise die Vorteile aus objektbasierter Programmierung und Skriptsprachen vereint werden konnten. Konkret bedeutet dies, dass zur Darstellung eines Texteditors und des zugehörigen Policy-Editors (Erklärungen hierzu folgen in den späteren Kapiteln) JavaScript und HTML in Verbindung mit CSS eingesetzt werden konnten.

- *Unabhängigkeit*

  Nicht zuletzt war ein bedeutender Punkt die laufzeitumgebungsunabhängige Ausführbarkeit von eSprint. Dies wird dadurch erreicht, dass bei der Implementierung einer JavaFX-Applikation eine spezifische Java Runtime Environment (JRE) mitgegeben werden kann. Die JRE wird zum Zeitpunkt der Erstellung der Client-Applikation festgelegt und in das Gesamtpaket integriert.

7.2 Digital Envelope

Nachdem die visuelle Repräsentation des Clients geklärt war, stellte sich die Frage: Wie kann eine sichere Kommunikation zwischen Clients und dem Server hergestellt werden? Eine symmetrische Verschlüsselung besitzt die Bedingung, dass beide Kommunikationspartner einen gemeinsamen sicheren Schlüssel teilen. Sollen die Clients nun mit dem Server über symmetrische Verschlüsselung kommunizieren, müssen die Parteien eine Methode nutzen, die einen Austausch des gemeinsamen Schlüssels ermöglicht. In diesen Fällen werden daher häufig asymmetrische Verschlüsselungsverfahren in Form der Public-Key-Verschlüsselung (Diffie-Hellmann) verwendet. In seiner simpelsten Form bietet Diffie-Hellmann allerdings keine Authentifizierung der zwei Kommunikationspartner. Zudem sind asymmetrische Verschlüsselungsverfahren hundert- bis tausendfach langsamer als symmetrische Verfahren. Um diese Probleme zu lösen, bietet sich eine Kombination der verschiedenen Verschlüsselungsarten an.

Eine Kombination aus asymmetrischer und symmetrischer Verschlüsselung erzeugt ein sogenanntes Digital Envelope, das eine Möglichkeit bietet, Nachrichten zu schützen, ohne dass zuvor Sender und Empfänger einen gemeinsamen Schlüssel besitzen. Zusätzlich kann eine Nachricht mit einer digitalen Signatur versehen werden, um dem Empfänger die Sicherheit zu geben, dass diese auch tatsächlich vom erwarteten Empfänger kommt und nicht verändert wurde.

Der genaue Vorgang der digitalen Signatur wurde im Kapitel 7.2 Secure-Hash-Funktionen behandelt [SB12].

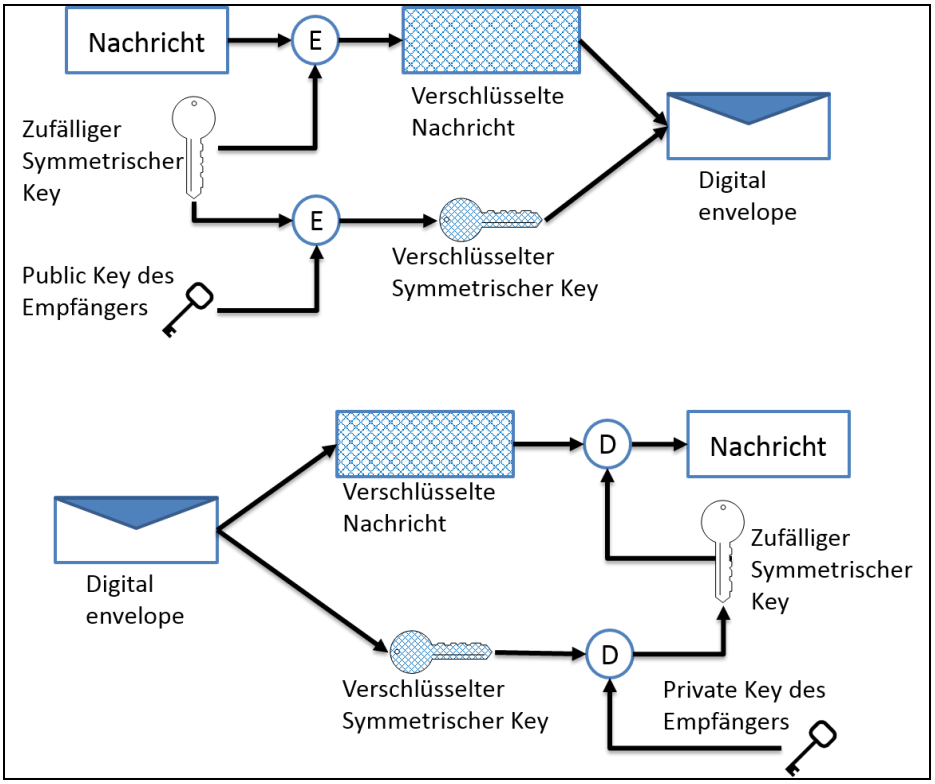

Abbildung 31: Digital Envelope Übersicht

Abbildung 31 zeigt die Erstellung und das Öffnen eines Digital Envelopes. Diese Prozesse sollen anhand eines Client-Server-Szenarios betrachtet werden, in dem der Server dem Client ein Dokument zukommen lassen möchte.

1. Der Server bereitet die zu sendende Nachricht auf. Mittels einer Hash-Funktion wird das zu übermittelnde Dokument vom Server zu einem Hash-Wert gewandelt, mit seinem Private Key verschlüsselt und dem Dokument als Signatur angehängt.
2. Anschließend generiert der Server einen neuen, zufälligen, symmetrischen Schlüssel, der nur im Zuge der Erstellung des Digital Envelope verwendet wird und daraufhin verworfen werden sollte.
3. Der Server nutzt diesen Schlüssel und verschlüsselt die zuvor aufbereitete Nachricht samt der Signatur.

4. Nachfolgend wird der zufällige, symmetrische Schlüssel mit dem Public Key des Empfängers, also des Clients, der das Dokument erwartet, verschlüsselt.
5. Abschließend wird der mit dem Public Key verschlüsselte, zufällige, symmetrische Schlüssel der verschlüsselten Nachricht angehängt und an den Client übermittelt.

Das Öffnen des Digital Envelopes auf Seiten des Clients verläuft wie folgt:

1. Der Client entnimmt dem Digital Envelope den mit seinem Public Key verschlüsselten, symmetrischen Schlüssel und entschlüsselt diesen mit seinem Private Key.
2. Der hieraus gewonnene symmetrische Schlüssel wird vom Client verwendet, um die verschlüsselte Nachricht zu entschlüsseln.
3. Der Client erhält so das Dokument und die vom Server angehängte Signatur.
4. Zur Überprüfung der Signatur wird der verschlüsselte Hash-Wert mittels des Public Key des Senders, also Servers, entschlüsselt und die übermittelte Nachricht anhand der vereinbarten Hash-Funktion ebenfalls zum Hash-Wert gewandelt und verglichen.

Durch dieses Vorgehen kann sich der Client sicher sein, dass das Dokument nur vom erwarteten Server kommen kann. Der Server kann sicher sein, dass nur der Client den versandten Schlüssel dekodieren und somit auch auf das Dokument zugreifen kann. Weiterhin ist durch die symmetrische Verschlüsselung des Dokuments eine bessere Performance des Systems gewährleistet [SB12, ST13].

7.3 De-Perimeterisation

Nachdem die grundlegenden Ansätze der Sicherheitsmechanismen für eine verschlüsselte Kommunikation erläutert wurden, soll die ‚innere' Sicherheit des eSprint-Systems untersucht werden.

Die Idee hinter De-P war ein Grund dafür, dieses Projekt zu entwickeln. Das hier vorgestellte eSprint-Projekt baut auf seinem Vorgänger, dem SPIDER-Projekt von der Cardiff University, auf und versucht, konsequent alle Schwächen des Vorgängers zu verbessern. Die grundlegenden Ideen unterscheiden sich dabei nur wenig vom SPIDER-Projekt. Sie umfassen etwa die Einführung einer Live-URL, die es dem Eigentümer einer Ressource ermöglicht, selbst nachträglich Änderungen an Zugriffsrechten vorzunehmen, auch wenn dies über Unternehmensgrenzen hinausgehen sollte. Es wird also effektiv eine Continous-Access-Control erzielt.

Informationen werden innerhalb des Dokumentes geschützt. Es ist somit möglich, in ein und demselben Dokument verschiedene Zugriffsarten mit jeweils separaten Verschlüsselungen und dadurch unterschiedliche Informationen unterzubringen. So können der erstellte Text individuell an die gegebenen Sicherheits- und Geheimhaltungsvoraussetzungen angepasst und eine de-perimeterisierte Informationsquelle erreicht werden. Diese Maßnahme führt vom ursprünglichen SPIDER-Konzept einer Verschlüsselung durch einen einzigen Key für alle Sicherheits- und Zugriffslevel weg und hin zur Einführung einer – wie in Kapitel 7.4.2 definierten – Multiple-Key-Access-Control. Dieser Mechanismus ist eine der maßgeblichen Verbesserungen im Vergleich zu seinem Vorgänger. Typische Anwendungsgebiete für die Umsetzung könnten etwa ein unternehmensweites verbessertes Content Management, kooperative CSCW-Plattformen, Service-orientierte Architekturen und die Bereitstellung von Daten über die Cloud sein.

7.4 Security-Management und Continous-Access-Control

In diesem Kapitel werden die gefundenen Lösungsansätze für die Implementierung aller notwendigen Sicherheitsmechanismen veranschaulicht. Es umfasst die Beschreibung der wichtigsten Technologien und Konzepte, die für die Realisierung des eSprint-Systems notwendig waren. Hierfür wurden nicht nur bestehende Mechanismen angewandt, sondern auch neue entwickelt.

7.4.1 Identitätsmanagement – Was ist x.509?

Die Public-Key-Verschlüsselung beruht auf dem Prinzip, dass der Public Key der Kommunikationspartner öffentlich zugänglich ist. Dies bringt allerdings die Problematik mit sich, dass bei den herkömmlichen Public-Key-Verfahren, wie auch RSA, ein Angreifer die Möglichkeit hat, sich den Public Key eines anderen Teilnehmers zunutze zu machen und sich als dieser auszugeben. Das Konzept der Public-Key-Zertifikate löst dieses Problem.

Public-Key-Zertifikate

Public-Key-Zertifikate bestehen aus einem Public Key und einer Empfänger-ID des Schlüsselbesitzers, wobei dieser Block als Ganzes von einer dritten vertrauten Partei signiert wurde. Das Zertifikat inkludiert auch Informationen über diese vertraute Partei und darüber, wie lange das ausgestellte Zertifikat gültig ist. Solch eine verwendete Authentifizierungsstelle wird meist als Client-Authority (CA) bezeichnet, der alle teilnehmenden Kommunikationspartner vertrauen. Ein Benutzer übermittelt seinen Public Key an eine CA und erhält von dieser ein signiertes Zertifikat. Dieses kann ein Benutzer dann publizieren. Bei einer Kommunikation zwischen zwei Teilnehmern können beide über die vertraute angehängte Signatur die Identität des anderen sicherstellen. Die Abbildung 32 zeigt die Verwendung eines Public-Key-Zertifikats [SB12, MA03, ST05].

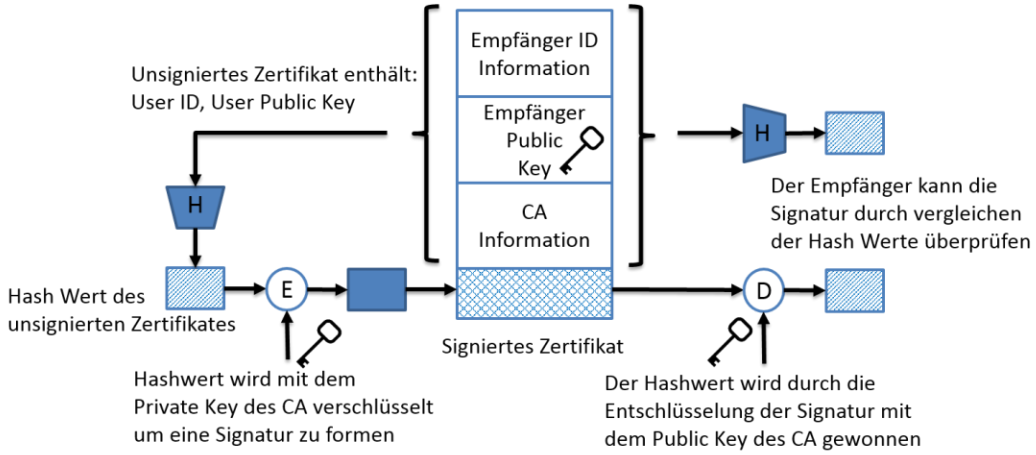

Abbildung 32: Verwendung eines Public-Key-Zertifikats

x.509-Zertifikate

Ein universell akzeptiertes Schema für die Public-Key-Verschlüsselung ist der x.509-Standard. Das x.509-Format wird heutzutage in den meisten Netzwerk-Sicherheitsapplikationen wie SSL[9], SET[10] oder auch S/MIME[11] verwendet.

[9] Secure Sockets Layer ist die alte Bezeichnung für Transport Layer Security, ein Netzwerkprotokoll zur sicheren Übertragung von Daten (Kapitel 5.4).

[10] Secure Electronic Transaction ist ein Sicherheitsprotokoll für den elektronischen Zahlungsverkehr mit Kreditkarten, im Besonderen über das Internet.

[11] Secure/Multipurpose Internet Mail Extensions ist ein Standard für die Verschlüsselung und das Signieren von MIME-gekapselter E-Mail durch ein hybrides Kryptosystem.

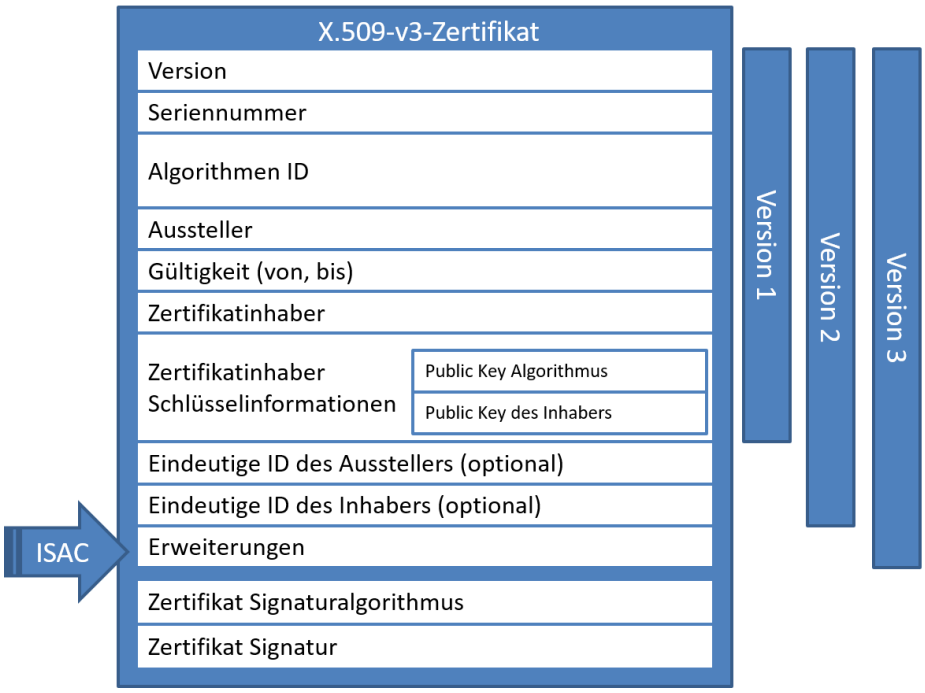

Abbildung 33: Struktur eines x.509-Standardzertifikates

In der oben dargestellten Abbildung 33 wird die Struktur eines x.509-Zertifikates exemplarisch dargestellt und darüber hinaus gezeigt, welche Felder in welcher Version des x.509-Zertifikates hinzugekommen sind [SB12, RFC5280].

- *Version*

  Mit Version 2 kamen eindeutige Identifikations-Felder des Ausstellers und Inhabers hinzu. Die Erweiterungsfelder kamen mit der Version 3 des x.509-Zertifikates und dienten, wie auch in der vorliegenden Ausarbeitung (7.5 Das ISAC-Zertifikat), der Steigerung der Flexibilität und der Ergänzung benötigter Informationen.
- *Seriennummer*

  Diese ist eine beim CA eindeutige Integer-Zahl, die mit dem Zertifikat assoziiert wird.
- *Algorithmen-ID*

  Diese ID dient der Identifikation des Signatur-Algorithmus.

- *Aussteller*

  Der X.500[12] Name der Zertifizierungsstelle (CA), die dieses Zertifikat kreiert und signiert hat.
- *Gültigkeit*

  Sie setzt sich aus einem ‚Gültig ab-' und einem ‚Gültig bis-'Datum zusammen.
- *Zertifikatinhaber*

  Dies ist der Name des Zertifikatnutzers, der den dazu passenden Private Key besitzt.
- *Zertifikatinhaber-Schlüsselinformationen*

  Der Public Key des Zertifikatinhabers sowie eine ID und Parameter des verwendeten Algorithmus stellen die Zertifikatinhaber-Schlüsselinformationen dar.
- *ID des Ausstellers*

  Sie ist ein optionaler bit-String, der verwendet wird, falls der X.500 Name des Ausstellers nicht eindeutig ist.
- *ID des Inhabers*

  Diese ist ein optionaler bit-String, der verwendet wird, falls der X.500 Name des Inhabers nicht eindeutig ist.
- *Erweiterungen*

  Der in Version 3 hinzugekommene Bereich ermöglicht das Ergänzen einer Vielzahl an zusätzlichen Feldern je nach Nutzen. Welche Felder in dem hier angedachten ISAC-Zertifikat hinzugefügt und genutzt werden, beschreiben die anschließenden Kapitel.
- *Signatur*

  Die Signatur enthält den Hash oder *message digest* (Kapitel 5.2), die mit dem Private Key des CA verschlüsselte Signatur der restlichen Felder und einen Identifikator des dazu verwendeten Algorithmus.

Abbildung 34: Beispiel x.509-Zertifikat durch eine exemplarische Textdarstellung eines x.509v3 aufgebauten digitalen Zertifikats.

[12] X.500 Standard, umfangreicher übergreifender Verzeichnisdienst

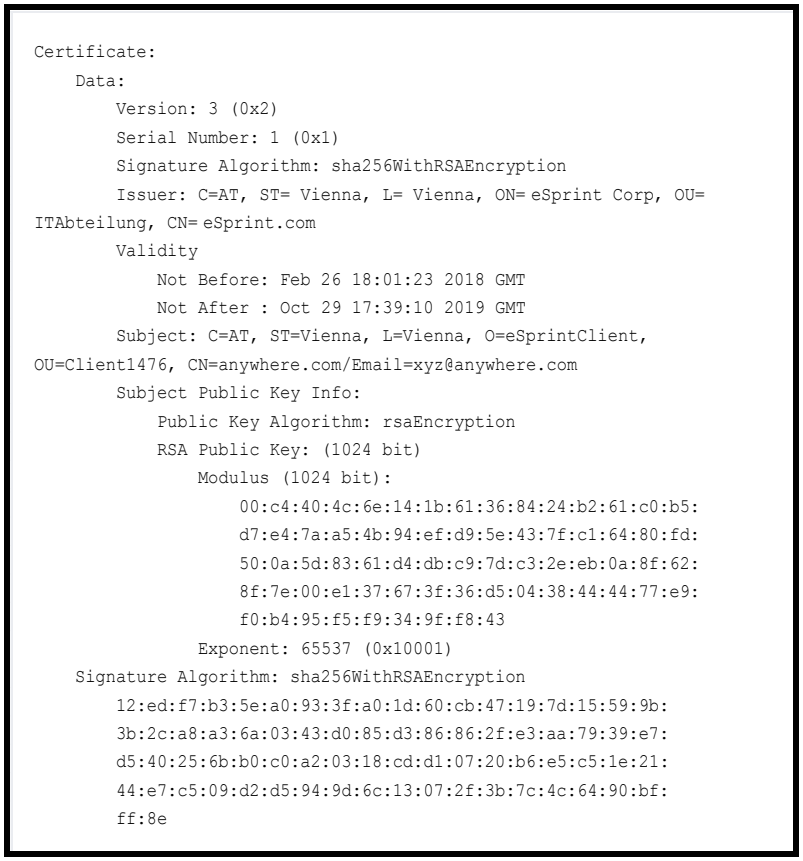

Abbildung 34: Beispiel x.509-Zertifikat

Die Zertifikatserstellung

CSR steht für Certificate Signing Request, also die gestellte Anfrage an eine CA, um ein SSL-Zertifikat zu erhalten. Dieser Block von kodiertem Text beinhaltet Informationen, die später in ein zu erstellendes Zertifikat (CRT) übernommen werden sollen. Dies beinhaltet mindestens Felder, die Auskunft über die Identität des Antragstellers geben, sowie einen generierten Public Key, der in das zu erstellende Zertifikat übernommen wird. Der dazugehörige Private Key wird zeitgleich beim Antragsteller entworfen, bleibt aber sicher verwahrt bei diesem. Codiert ist ein CSR gewöhnlich unter Verwendung der Datenstruktur-Beschreibungssprache ASN.1 gemäß der PKCS #10 Spezifikation [CSR00]. Tabelle 5 gibt eine Übersicht über die darin enthaltenen Felder.

Name	Beschreibung	Beispiel
Common Name (CN)	Name des Antragstellers, für Internetanwendungen meist der Domain-Name	*.eSprint.com mail.eSprint.com
Organization Name (ON)	rechtlicher Name der Organisation	eSprint Corp.
Organizational Unit (OU)	für das Zertifikat verantwortliche Abteilung	IT-Abteilung
City/Locality	Heimatstadt des Unternehmens	Vienna
State/County/Region	Heimatstaat oder Region des Unternehmens (ungekürzt)	Vienna
Country	zweistelliger ISO-Code	AT
Email address	E-Mail-Adresse	info@esprint.com
Public Key	zu übernehmender Public Key	automatisch generiert

Tabelle 5: Inhalt eines Certificate Signing Requests

Die meisten CSRs werden in einem Base64-kodiertem PEM-Format erstellt. Das Format inkludiert "-----BEGIN CERTIFICATE REQUEST-----" am Beginn und "-----END CERTIFICATE REQUEST-----" am Ende der .csr-Datei. Abbildung 35 zeigt eine Beispielansicht, wie diese .csr-Datei in einem Texteditor aussieht:

```
-----BEGIN CERTIFICATE REQUEST-----
MIIFFDCCAvwCAQAwgZoxCzAJBgNVBAYTAkFUMQ8wDQYDVQQIDAZWaWVubmExDzAN
BgNVBAcMBlZpZW5uYTEYMBYGA1UECgwPU3BpZGVyQ2xpZW50MDAxMRgwFgYDVQQL
DA9TcGlkZXJDbGllbnQwMDExGDAWBgNVBAMMD1NwaWRlckNsaWVudDAwMTEbMBkG
s42THwceYXnwhwST8Cs3G26NOBCY3t2QQ3okh/hjbDdZxI1tULq6Sg/0g6IarABT
2QMhWlmClGUrI8H37J1HUDJAzbQt5lXxQImaGKDq4F6yUDELtlufHLcVazL+qUYx
PM2iKzJhnbjzx5tCvIPkaPYAY0FCwU42ApWn9wgCIxTZfCnm7VEJ/z7LY6qkdJ8J
3HVUnOA4HjkSPyx7k6/U6dxKQGhQdYzJoAIj5PWqNBd6AXe9f/EDXUYBIaDOaNQX
DZOHiQkVozF0BlpQ7PqyBNWZUAI2s7Dofpi2+BZ89+mNKXUh9Yix4ugShUkZEpM8
lLba9c8/ztxCOez8JV5urrbDMsODeF5j0dHI/xa1N6dn15DLCHxa5MV9Z6oqYTUu
b4wZ5dkveMA/88736tIi00huRRc+knYk0rsECT2Yuz8gE/1ZT76du94PdHGYj8KO
xS1qj3lgWFg=
-----END CERTIFICATE REQUEST-----
```

Abbildung 35: Beispiel für ein Certificate Signing Request

Ist der CSR erstellt, wird er an die gewünschte CA gesendet und erhält von dieser ein signiertes Zertifikat (CRT) zurück. Das CRT bildet ein wie in den Abbildung 33 und Abbildung 34 beschriebenes Zertifikat und beinhaltet mindestens alle vom CSR angeforderten Felder.

In weiterer Folge werden Zertifikate hier oftmals als CRT bezeichnet. .CRT ist eine von den Autoren der Arbeit verwendete File-Extension eines Zertifikates, die sowohl im binären DER[13]-Format als auch im ASCII PEM[14]-Format kodiert sein kann.

7.4.2 Multiple-Key-Management mittels Extended x.509

Nach der Vorstellung der grundlegenden Funktionsweise von x.509-Zertifikaten wird im Folgenden auf die Adaption dieser eingegangen. Die Fragestellung in diesem Zusammenhang war, wie ein effektiver Schutz der Ressource bei gleichzeitiger individueller Anpassbarkeit der geschützten Elemente erreicht werden kann. Das Ziel war eine Lösung, bei der die jeweils unterschiedlich berechtigten Betrachter innerhalb der Ressource beziehungsweise des Texts mittels verschiedener Verschlüsselungen eine individuelle Einsicht in die präsentierten Informationen erhalten.

Für das eSprint-System war es notwendig, den x.509-Standard mithilfe seiner regulär vorgesehenen Erweiterungen (Extensions) um einen weiteren Sicherheitsmechanismus auszubauen. Die Beschreibung der x.509-Erweiterung und der Transport dieser Erweiterungen zwischen dem Client und dem Server wird Teil des Kapitels 7.5 sein und somit einen der wichtigsten Aspekte dieser Arbeit behandeln. Im nachfolgenden Kapitel geht es um die Verschlüsselung der Ressource. Dies stellt zugleich den Hauptunterschied von eSprint im Vergleich zu seinem Vorgänger SPIDER dar.

Bei dieser von Frau Alam Aldeen und den Autoren erdachten Idee handelt es sich um das sogenannte Multiple-Key-Management. Hierbei werden für eine Ressource beziehungsweise einen Text immer fünf symmetrische Schlüssel – im vorliegenden Fall AES256-Schlüssel – erstellt. Diese dienen dem Ersteller einer Ressource dazu, später den Text anhand der vier zur Auswahl stehenden Zugriffsarten (auch Access-Arten) klassifizieren zu können. Diese Klassifizierungs-Schemata umfassen folgende vier Optionen:

[13] .DER, DER-kodiertes Zertifikat
[14] .PEM, Base64-kodiertes Zertifikat

- *Open Access*
- *Community Access*
- *Organisation Access*
- *Restricted Access*

Mittels der definierten Textstellen kann die Ressource anhand der AES-Schlüssel entsprechend verschlüsselt werden. Diese Schlüssel werden in der x.509-Erweiterung gespeichert und an den Nutzer übermittelt. Die Textstellen werden durch den Benutzer über einen Button gekennzeichnet, der wiederum unsichtbar für den Nutzer ist und eigene XML-Tags in den Text einfügt. Wie dies exemplarisch aussieht, ist in Abbildung 36 zu erkennen.

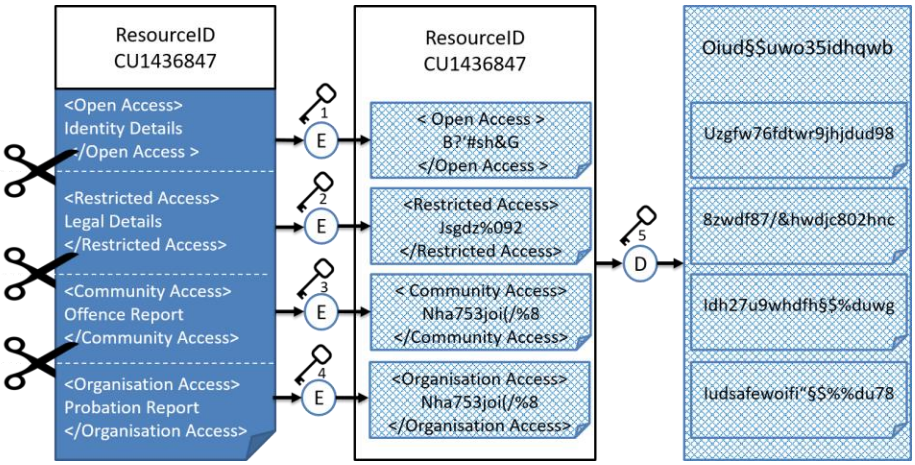

Abbildung 36: Multiple-Keys-Verschlüsselung

Zunächst existiert die vom Eigentümer erstellte Ressource, die aus mehreren Textpassagen besteht, die jeweils durch eine der vier Access-Arten gekennzeichnet wurden. Im ersten Schritt des Speichervorganges wird jeder der Abschnitte mittels eines der vier AES-Keys verschlüsselt. Danach und somit direkt vor der Übertragung zum Server findet eine weitere Verschlüsselung durch den fünften AES-Key statt. Dieser verschlüsselt nochmals die gesamte Ressource, wodurch eine höchstmögliche Sicherheit der übertragenen Daten gewährleistet wird.

Die Zugriffsarten sind *nicht* hierarchisch aufeinander aufbauend. Aus diesem Grund obliegt es letztlich dem Ersteller und/oder Nutzer einer Ressource, wie die vorhandenen Access-Arten eingesetzt werden. Diese dienen lediglich der Abgrenzung untereinander und die Bezeichnung soll eine logische Zuordnung der Textpassagen erleichtern. Die ‚Übereinanderlegung' von verschiedenen Zugriffsarten ist jedoch möglich. Dies läuft wie folgt ab: Der Nutzer kann über den Editor im Client einen Text erzeugen, bei dem er einen beliebigen Textblock als Open Access markiert. Danach kann er innerhalb dieses Textblocks einen weiteren (kleineren) Textblock markieren und diesen als Organisation Access kennzeichnen. Das Ziel hierbei ist also, dass die meisten in der Praxis relevanten Szenarien abgebildet werden können. Um diese Möglichkeiten noch granularer gestalten zu können, kann ein Benutzer einer Ressource einer oder mehreren Gruppen zugeordnet werden. Dieses Thema wird im nachfolgenden Kapitel erörtert.

7.4.3 Gruppenrichtlinien und Gültigkeitsrahmen

Um den Sicherheitsmechanismus des Multiple-Key-Managements noch granularer zu gestalten und mit mehr Zusatzfunktionen auszustatten, wurde das eSprint-System um die Möglichkeit von Gruppenrichtlinien und Gültigkeitsrahmen erweitert. Dies ist die sogenannte Continous-Access-Control (CAC).

Diese setzt die Idee eines Klassifikations-Schemas fort und ermöglicht es dem Eigentümer einer Ressource, die anderen Nutzer der Ressource in Gruppen einzuteilen. Somit muss eine Berechtigung zu der jeweiligen Access-Art nicht mehr direkt über einen Benutzer-Account erfolgen, sondern findet eine Ebene darüber auf der Gruppenebene statt. Dies gestattet automatisch mehrere Gestaltungsvarianten der vorliegenden Zugriffsmanagementlandschaft.

Zum einen kann ein Nutzer einer Ressource mehreren Gruppen zugeteilt werden. Somit wird es ermöglicht, dass ein Nutzer beispielsweise gleichzeitig sowohl verschlüsselte Community als auch Restricted-Bereiche eines Textdokuments sehen kann. Dadurch gewinnen Konstrukte wie die zuvor vorgestellte ‚Übereinanderlegung' von Zugriffsbereichen einen Sinn.

Zum anderen wird über Gruppen auch ein automatischer Gültigkeitszeitraum von Zugriffsrechten gesteuert. Es kann also festgelegt werden, bis zu welchem

vordefinierten Zeitpunkt beispielsweise der Gruppe 1 mit den Zugriffsrechten Community Access, Restricted Access und deren zwei zugeordneten Mitgliedern der Zugriff auf die Ressource gestattet ist. Eine kurze Übersicht hierzu ist in Tabelle 6 zu sehen.

Gruppen	Open Access	Community Access	Organisation Access	Restricted Access	Gültigkeit
Gruppe 1		•	•		31.10.2018
Gruppe 2	•				01.01.2019

Tabelle 6: Exemplarische Übersicht zu Gruppenrechten

Nach Ablauf des Gültigkeitszeitraumes haben Mitglieder dieser Gruppe automatisch keinen Zugriff mehr auf die Ressource, es sei denn, der Eigentümer der Datei verändert mithilfe des sogenannten Policy-Editors die Gruppeneigenschaften. Der genaue Vorgang hierzu wird im Kapitel 9.5 und eine grafische Darstellung in Kapitel 11.3 gezeigt.

Wird das Beispiel aus dem letzten Kapitel nochmals aufgegriffen, so ergibt sich für Gruppe 1 das in Abbildung 37 gezeigte Schema.

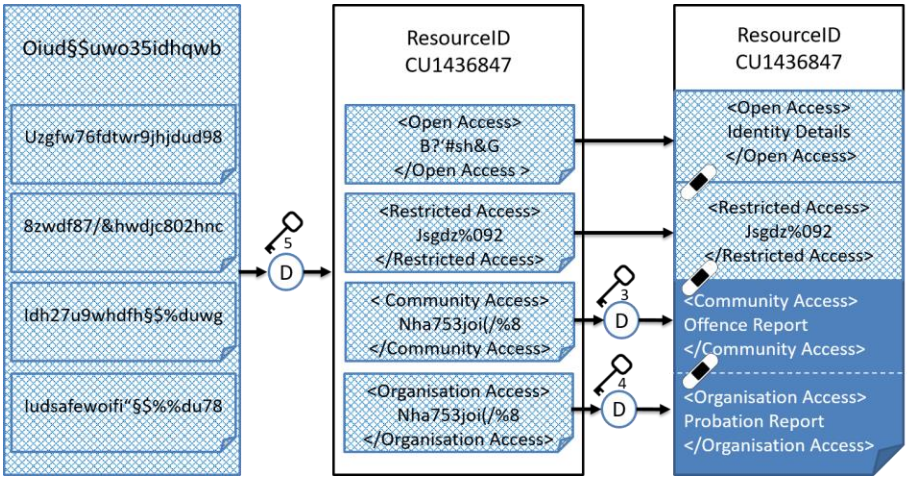

Abbildung 37: Ansicht einer Beispieldatei durch Gruppe 1

Hier ist zu erkennen, dass die Angehörigen der Gruppe 1 zwar die gesamte Ressource in Form eines Textdokuments erhalten, diese jedoch lediglich partiell

dechiffrieren können. Die restlichen Teile verbleiben in dem Dokument als verschlüsselte, nicht bearbeitbare Informationen.

Zusammengefasst dient CAC einem nachhaltigen Schutz, aber auch der Nutzbarkeit einer Ressource. Mit früheren Mechanismen war es meist nur möglich, einmal die entsprechenden Rechte zu vergeben und somit nach Veröffentlichung einer Ressource keine weiteren Anpassungen mehr an der Zugriffssicherheit vornehmen zu können. Dies machte die Kontrolle von digitalen Informationen entweder schlecht oder gar nicht steuerbar und konnte auch aus rechtlicher Sicht unter Umständen zu Problemen führen. CAC stellt also in Zeiten von dezentralen Arbeitsweisen und oftmals zahlreichen beteiligten Akteuren beim Ablauf eines Arbeitsprozesses ein essenzielles, wenn nicht sogar notwendiges, Sicherheitsinstrument dar.

Nachfolgend werden die Security-Aspekte der übergeordneten Sicherheitsmechanismen erläutert.

7.4.4 Single-Session-Security

Beim Prozess der Findung von Lösungsansätzen für die Umsetzung des eSprint-Systems ergab sich auch die Fragestellung einer konsistenten Datenhaltung. Konkret muss zu jedem Zeitpunkt gewährleistet sein, dass sich durch die gleichzeitige Datenbearbeitung von mehreren Nutzern kein Datenschiefstand ergibt. Da dieses Projekt nur auf begrenzte Mittel zurückgreifen kann und dieser Aspekt nicht im Vordergrund stand, musste hierfür eine angemessene, aber auch einfache Lösung gefunden werden. Aus diesem Grund wurde die sogenannte Single-Session-Security umgesetzt.

Die Idee hierbei ist, dass eine Datei zu jedem Zeitpunkt immer nur einen Benutzer haben darf. Solange eine Datei geöffnet ist, kann kein anderer darauf zugreifen. Um dies zu bewerkstelligen, werden zur Bearbeitungszeit durch den Client Keep-alive-Signale an den Server übermittelt. Diese signalisieren anderen Clients, dass die Ressource derzeit nicht verfügbar ist. Sobald die Bearbeitung beendet und die Datei geschlossen wurde, kann sie wieder durch andere Nutzer verwendet werden.

Diese einfache Umsetzung eines Session-Managements erweitert auch die allgemeine Sicherheit von eSprint. Hierdurch wird beispielsweise das Risiko von Man-

in-the-Middle-Attacken verringert. Diese sind oft dadurch gekennzeichnet, dass eine dritte unbefugte Partei sich als jemand anderen ausgibt, um Einsicht in geschützte Informationen zu erlangen. In diesem Fall ist es Angreifern zumindest nicht ohne Weiteres möglich, sich als der Client auszugeben, der derzeit die Datei bearbeitet. Unterstützt wird dieser Sicherheitsaspekt durch weitere implementierte Mechanismen wie etwa den in Kapitel 7.2 vorgestellten Digital Envelope.

7.4.5 Layer-Security

Ein weiterer Teil des erdachten Sicherheitskonzeptes ist die sogenannte Layer-Security. Diese soll eine klare Trennung der Benutzerinteraktionen von der dahinter liegenden technischen Umsetzung gewährleisten und somit die Sicherheit des eSprint-Systems erhöhen.

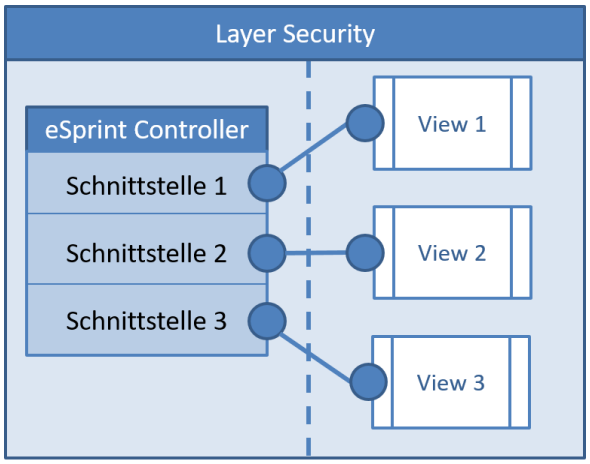

Abbildung 38: Ausschnitt MVC – Layer-Security

Konkret geht es bei diesem Ansatz um eine kleine Erweiterung des üblichen Model-View-Controller-Konzepts. Dieses bedeutet die Trennung zwischen der Präsentations-, der Steuerungs- und der Datenhaltungsschicht, wobei die beiden Letzteren in der Praxis oft auch zusammengefasst oder zumindest nicht klar getrennt werden können. Bei den herkömmlichen Implementationen ist es aber oft nur eine logische Trennung dieser Schichten, die nicht vom Sicherheitsstandpunkt aus betrachtet wird. Das heißt, dass Views bei Stand-alone-Endnutzerprogrammen oft direkt auf Methoden in der

Steuerungsschicht zugreifen können, um dort die benötigten Daten anzufordern oder verarbeiten zu lassen.

Die eSprint-Layer-Security sieht hierfür nochmals eine klare Trennung dieser beiden Ebenen der Präsentation und der Steuerung vor, wie es in Abbildung 38 zu erkennen ist. Hierfür werden in der Präsentationsschicht lediglich Web-View-Elemente der JavaFX-Technologie angewandt, um dem Benutzer eine Oberfläche mittels HTML5 und JavaScript anzuzeigen. Die hierin generierten oder benötigten Daten werden dann durch definierte Schnittstellen vom Controller angefordert oder an diesen geschickt. Hiermit wird eine direkte Kontrolle der Views von dahinter liegenden Methoden oder Daten abgekapselt und unterbunden.

7.5 Das ISAC-Zertifikat

Der zentrale Punkt des eSprint-Systems ist das ISAC-Zertifikat. Dieses stellt eine Kombination vieler der zuvor vorgestellten Technologien und Ideen dar und ist das Kernstück der von Frau Nisreen Alam Aldeen erdachten Weiterentwicklungen des SPIDER-Projektes.

Bei ISAC – das Identification and Secure Access Control Certificate – handelt es sich um eine Weiterentwicklung des x.509-Standards. Die Kernfunktion dieser Weiterentwicklung zielt darauf ab, die Sicherheit und die Flexibilität der Einsatzmöglichkeiten zu erhöhen. Konkret hilft die Funktion dabei, erweiterte Kontrollmechanismen in ein herkömmliches Zertifikat zu injizieren.

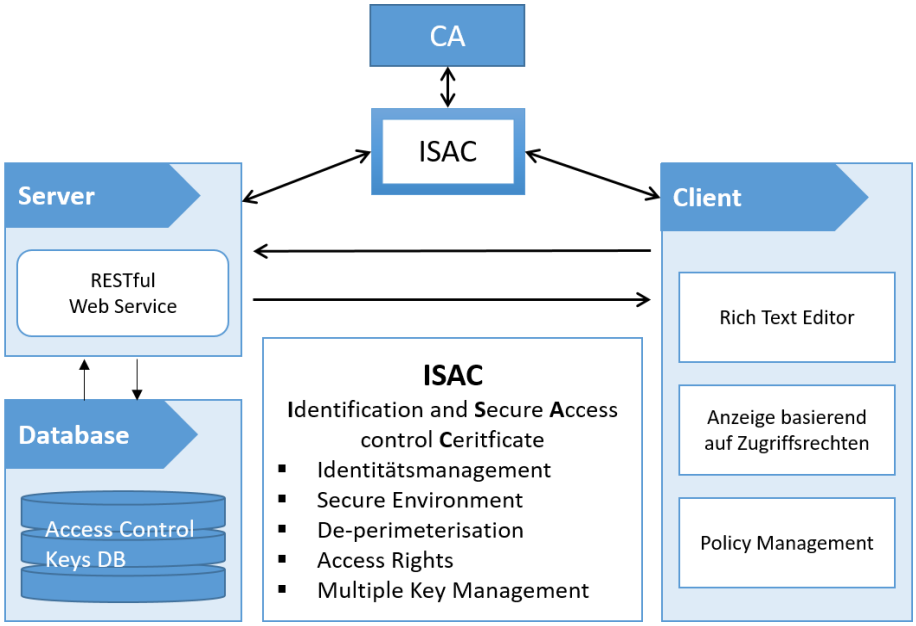

Abbildung 39: Einsatzbereiche von ISAC

In Abbildung 39 sind die Einsatzbereiche des neu entwickelten ISAC-Zertifikats dargestellt. Zunächst gilt es zu beachten, dass das an den Client auszustellende ISAC immer mithilfe einer Server-spezifischen CA erstellt wird. Dies bedeutet, dass der Server die volle Kontrolle über die Neuzulassung von Clients zu bestehenden oder neuen Projekten hat. Hierdurch werden ein hohes Maß an Sicherheit gewährleistet und gleichzeitig eine gewisse Unabhängigkeit von Drittsystemen gewahrt. Doch wofür steht ISAC nun im eigentlichen Sinn und wie trägt es zu einer erweiterten Datensicherheit bei?

Das ISAC-Zertifikat dient dem eSprint-System vorwiegend als Vehikel zum Transfer erweiterbarer Zugriffssteuerung und somit der Schaffung eines sicheren Datenmanipulationsumfeldes. Dies umfasst, einfach ausgedrückt, die Grundaufgabe dieser Technologie. Bei der Betrachtung der geschaffenen Möglichkeiten wird jedoch das gesamte Potenzial von ISAC deutlich. Der prinzipielle Aufbau eines ISAC-Zertifikates – basierend auf dem x.509-Standard – implementiert die üblichen Zertifikatsmechanismen wie etwa ein Identitätsmanagement, mithilfe dessen der Zertifikatsinhaber eindeutig authentifiziert werden kann. Hierzu zählt die Ausstellung

des Zertifikats auf einen eindeutig benannten User mit dessen E-Mail-Adresse, Organisation und Firma. Eine genauere Auflistung dieser Felder befindet sich im Kapitel 7.4.1. Dies befähigt das eSprint-System zu einer lückenlosen Authentifizierungskette und somit der Einhaltung einer hohen Vertraulichkeit beim Speichern, Laden und Bearbeiten von Dateien. Zusätzlich zu dem Basic-Identity-Management beinhaltet ein ISAC-Zertifikat immer eine eindeutig zuzordnende ID zu der Ressource, für die dieses Zertifikat eigens erstellt wurde. Das bedeutet, dass ein ISAC immer nur für eine Ressource zu einem bestimmten Zeitpunkt für einen bestimmten User Gültigkeit besitzt und ohne diese erfüllten Kriterien den Zugriff auf die zu öffnende Datei verhindert.

Ein weiterer Schritt auf dem Weg zur Umsetzung von eSprint und einem sicheren Arbeitsumfeld für die unterschiedlichsten Szenarien war die Implementierung einer De-Perimeterisation, die über das ISAC-Zertifikat gesteuert werden kann. Der Konnex zu ISAC stellt sich bei dem hierfür notwendigen Transport eines Schlüssels zur Chiffrierung einzelner Parts innerhalb eines Textes her. Diese Schlüsselverwaltung für das Zugriffsrechte-Management bewerkstelligt ISAC über die Zertifikatserweiterungen, wie sie nachfolgend erklärt und in Abbildung 40 gezeigt werden. Da ein einzelner Schlüssel für ein effektives feingranulares Klassifikationsschema jedoch nicht ausreicht, wurde auch hier weiterer Aufwand betrieben, um dem Benutzer mehr Möglichkeiten zur Steuerung der Zugriffsberechtigungen zu bieten.

Aus diesem Grund wurde mittels ISAC ein Multiple-Key-Management implementiert. Hierdurch können detailliert innerhalb einer Ressource alle Zugriffe auf sensitive Textparts begrenzt und kontrolliert werden.

In Verbindung mit weiteren Security-Mechanismen wird so ein Datenbearbeitungsumfeld höchster Güte erreicht.

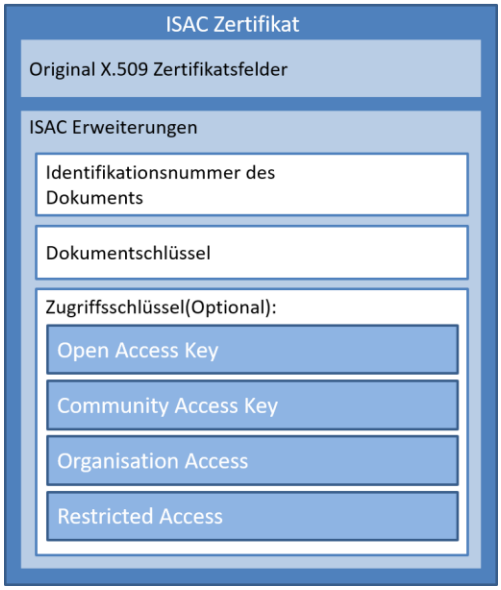

Abbildung 40: ISAC-Zertifikat – Detaildarstellung der Erweiterung

- *Identifikationsnummer des Dokuments*

  Dieses Feld enthält die Identifikationsnummer, auf die sich das ISAC-Zertifikat bezieht. Ein erstelltes ISAC-Zertifikat bezieht sich immer nur auf ein Dokument und ist nur für die Verwendung durch einen Client gedacht.

- *Dokumentschlüssel*

  Dieser enthält einen symmetrischen Verschlüsselungsschlüssel, mit dem das angefragte Dokument entschlüsselt werden kann. Der Dokumentschlüssel ist noch mit dem Public Key des Clients verschlüsselt, sodass nur dieser das Dokument lesen kann.

- *Zugriffschlüssel*

  Je nach Berechtigung des anfragestellenden Clients können optional Zugriffschlüssel an die Erweiterungen des ISAC-Zertifikates angehängt werden. Exemplarisch wurden für das vorliegende Projekt die Access-Arten Open Access, Community Access, Organisation Access und Restricted Access vorgeschlagen. Für jede Access-Art kann ein symmetrischer Schlüssel angehängt werden, der wiederum wie der Dokumentschlüssel mit dem Public Key des Clients verschlüsselt werden soll.

7.6 Konkreter Umsetzungsansatz

An dieser Stelle werden die genannten Ideen und Lösungsansätze zusammengeführt. Im Detail wird nun der Informationsaustausch unter Verwendung der zuvor erläuterten Technologien gezeigt. Um diese Kommunikation zu ermöglichen wird das zuvor vorgestellte ISAC Zertifikat eingesetzt.

In der nachfolgenden Abbildung 41 ist der typische Ablauf einer Ressourcenentschlüsselung dargelegt.

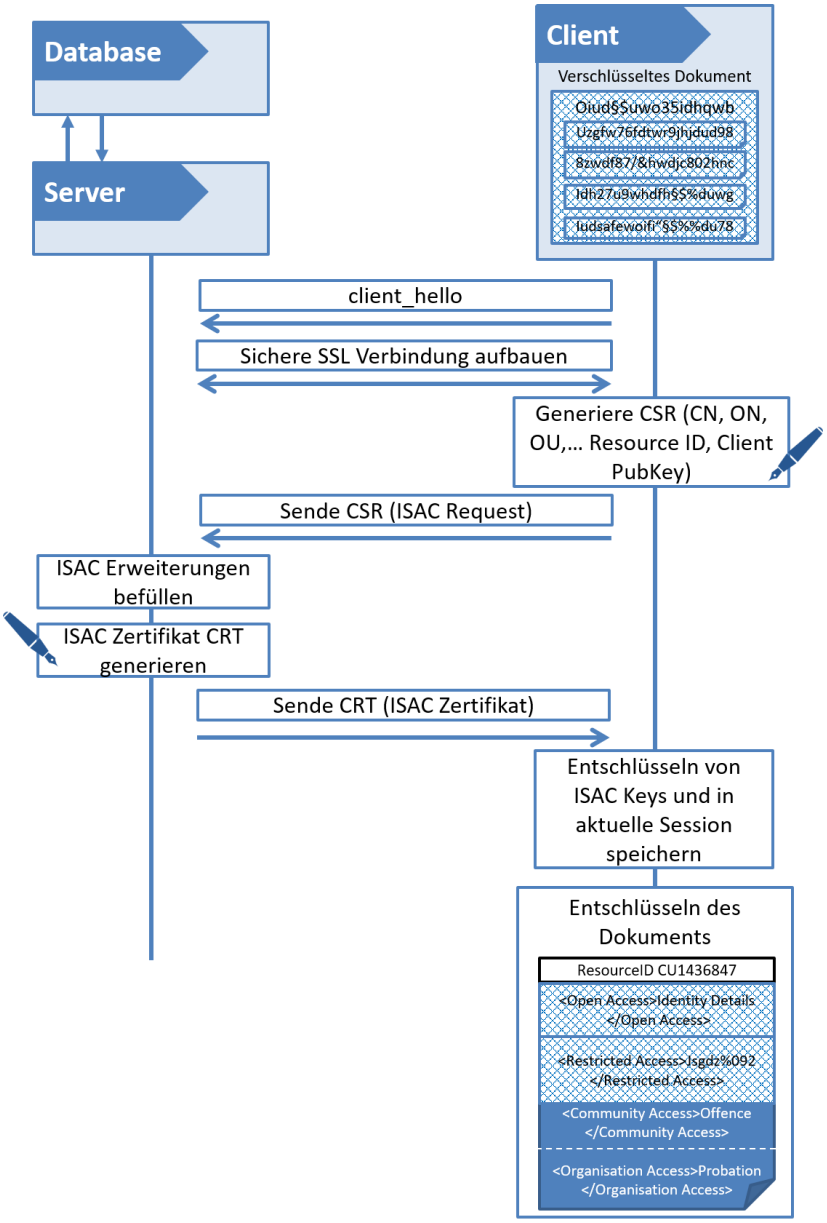

Abbildung 41: Ablaufdiagramm Ressourcenentschlüsselung mittels ISAC

Wie in der Abbildung zu erkennen ist, sind am Entschlüsseln einer Ressource sowohl der Client als auch der Server beteiligt. Wie bereits im Kapitel 7.1 erläutert, basiert der eSprint-Client auf der JavaFX-Technologie und ist dadurch befähigt, eine reichhaltige

Textformatierung inklusive Grafiken darzustellen und zu bearbeiten. Um dies durchführen zu können, muss er jedoch entweder eine Ressource generieren oder eine bestehende Datei öffnen. In dem gezeigten Beispiel handelt es sich um eine bereits bestehende Datei, die geöffnet werden soll. Diese wurde in einem vorhergehenden Schritt bereits zum Client übermittelt und soll nun entschlüsselt werden. Die geplante Umsetzung des Vorganges der Entschlüsselung folgt dem anschließend beschriebenen Muster:

- *Client_hello*

  Zunächst etabliert der Client eine Verbindung zum Server, wie es im Unterkapitel *Handshake-Protocol* im Kapitel 5.4 beschrieben wurde. Zusammengefasst handeln Server und Client in diesem Schritt die Art ihrer Kommunikation miteinander aus, legen fest, welche Algorithmen und Verschlüsselungsarten angewandt werden und stellen danach eine Verbindung zueinander her. Das Besondere an diesem Schritt ist, dass es dem Client überlassen ist, von wo er seine Ressourcen bezieht. Sollte es also aus technischen Gründen notwendig werden, dass die Informationen eines Servers auf einen anderen Zugriffspunkt verschoben werden müssen, würde dies nicht die automatische Unbrauchbarkeit einer Client-Anwendung bedeuten. Es wäre lediglich die Angabe der entsprechenden neuen URL im Client-Interface notwendig.
- *Sichere SSL-Verbindung aufbauen*

  Nachdem sich der Client beim Server ‚vorgestellt' hat, kann eine sichere SSL-Verbindung aufgebaut werden. Dieser Schritt stellt sicher, dass nur zuvor autorisierte und durch den Server ausgestellte eSprint-Clients Zugriff auf den Server erlangen können. Da der Server zugleich Datenverwalter und CA ist, hat er von Beginn an volle Kontrolle über den Prozess.
- *Generiere CSR*

  Nachdem der Client eine sichere Verbindung zum Server hergestellt hat, kann er eine passende Zertifikatsanfrage für die gewünschte Ressource erstellen. Diese ist notwendig, um dem Client ein – nur für diese Ressource und diese Session gültiges – ISAC-Zertifikat ausstellen und schicken zu können. In dieser Anfrage trägt der Client unter anderem die ID der anzufragenden Ressource, die

allgemeinen Daten des Clients selber, die er aus seinem ursprünglichen Zertifikat auslesen kann, sowie seinen eigenen Public Key ein. Abschließend wird dieses CSR noch durch eine digitale Signatur geschützt, welche die Vertraulichkeit der übermittelten Daten garantiert. Die genaue Beschreibung dieses Vorganges wurde bereits im Kapitel 7.4.1 erläutert und kann dort nachgelesen werden.

- *Sende CSR*

  Alsbald der Client seine Zertifikatsanfrage erstellt hat, kann er diese über den standardmäßig etablierten SSL-Tunnel zum Server übermitteln. Dieser Schritt ist zwar ein CSR, stellt aber die erste Stufe zur Übermittlung des ISAC-Zertifikats dar und wird deshalb auch als ISAC-Request bezeichnet.

- *ISAC-Erweiterung befüllen*

  Sobald der Server die Zertifikatsanfrage des Clients erhalten hat, kann er ein entsprechendes CA-signiertes Zertifikat daraus erstellen. Hierfür zieht er die vom Client übermittelte Zertifikatsstruktur mit den dazugehörigen vorbefüllten Feldern heran und hängt an diese die, den Zugriffsrechten des jeweiligen Nutzers entsprechenden, Erweiterungsfelder mit den dazugehörigen AES-Schlüsseln zur Dechiffrierung der Ressource.

- *ISAC CRT generieren*

  Nachdem alle notwendigen Informationen zur Zertifikatsgenerierung zusammengetragen wurden, kann der Server daraus ein gültiges Zertifikat generieren. Hierfür nutzt er das zuvor angesprochene CA, das garantiert, dass alle ausgestellten Zertifikate ausschließlich von dem jeweiligen Server stammen können. Mithilfe dessen und des übermittelten CSR kann er dann das in Kapitel 7.5 beschriebene ISAC-Zertifikat erstellen.

- *Sende CRT*

  Dieser Schritt umfasst lediglich die Rückübermittlung des gerade erstellten ISAC-Zertifikats an den Client über eine SSL-gesicherte Verbindung.

- *Entschlüsseln und Speichern von ISAC-Keys*

  Sobald der Client das ISAC-Zertifikat vom Server erhalten hat, kann er die mitgesendeten Dokumentenschlüssel aus den Erweiterungen auslesen. Diese sind jeweils zur Sicherheit nochmals mit dem Public Key des Clients verschlüsselt und

daher auch nur mit dessen Private Key entschlüsselbar. Hier ist anzumerken, dass die Schlüssel für nicht berechtigte Parts des Dokumentes gar nicht erst an den Client übermittelt werden. Dadurch wird garantiert, dass sich der Client keinen Zugriff auf diese verschaffen kann, um gegebenenfalls unrechtmäßig unzugängliche Teile der Ressource zu entschlüsseln und so einzusehen.

- *Entschlüsseln des Dokuments*
  Abschließend kann der Client mittels des erhaltenen ISAC-Zertifikats und den darin enthaltenen Schlüsseln die entsprechenden Teile des verschlüsselten Dokumentes dechiffrieren. Hierzu liest er die übermittelten ISAC-Keys aus und speichert sie zu der entsprechenden Session des aktuell geöffneten Editor-Fensters. Die Entschlüsselung der ISAC-Keys erfolgt mittels des Private Keys des Clients.

Nachdem die Ressource entschlüsselt wurde, kommen die in Kapitel 7.4.4 und 7.4.5 beschriebenen Sicherheitsmechanismen zum Einsatz, um den durchgängigen Schutz der Ressource weiter zu stärken und mögliche Angriffe auf das Umsetzungskonzept von eSprint zu erschweren.

Weiters ist es dem Ersteller einer Ressource (im weiteren Verlauf der Arbeit auch als Owner bezeichnet) möglich, die Zugriffsrechte der anderen Ressourcen-Nutzer zu modifizieren. Hierbei kommt der Policy-Editor zum Einsatz, der wiederum die Möglichkeit bietet, die in Kapitel 7.4.3 angesprochenen Gruppen zu verwalten.

Es wird also deutlich, dass all die beschriebenen Mechanismen sich gut in ein Gesamtbild einfügen, um das geplante eSprint-Projekt realisieren zu können und die Anforderungen zu erfüllen oder sogar zu übertreffen. Das nachfolgende Kapitel widmet sich deshalb der architektonischen Umsetzung und den technischen Systemkomponenten.

8 Systemarchitektur und -komponenten

In dem Kapitel Systemarchitektur und -komponenten werden zum einen die zum Einsatz kommenden Technologien erläutert und zum anderen ein erster Einblick in die Umsetzung der Client- beziehungsweise Serverstruktur sowie deren Funktionsweise

gegeben. Darüber hinaus soll das Konzept des Datenaustausches zwischen den einzelnen Komponenten definiert und erklärt werden.

8.1 Architektur

Bei der Beschreibung des vorliegenden Systementwurfs soll zunächst die Softwarearchitektur berücksichtigt werden. Diese soll einerseits die komplexen Zusammenhänge innerhalb der Server- beziehungsweise Client-Struktur vereinfacht darstellen und andererseits einen Einblick in die Verbindung sowie die transportierten Informationen zwischen diesen ermöglichen. Außerdem findet sich darin eine kurze Übersicht über die wichtigsten Technologien, die zur Realisierung des Projektes eingesetzt wurden. Es handelt sich bei dem in Abbildung 42 dargestellten Diagramm um eine sogenannte Funktionsarchitektur, welche die Gliederung des Systems in Funktionen, Technologien und Features veranschaulicht.

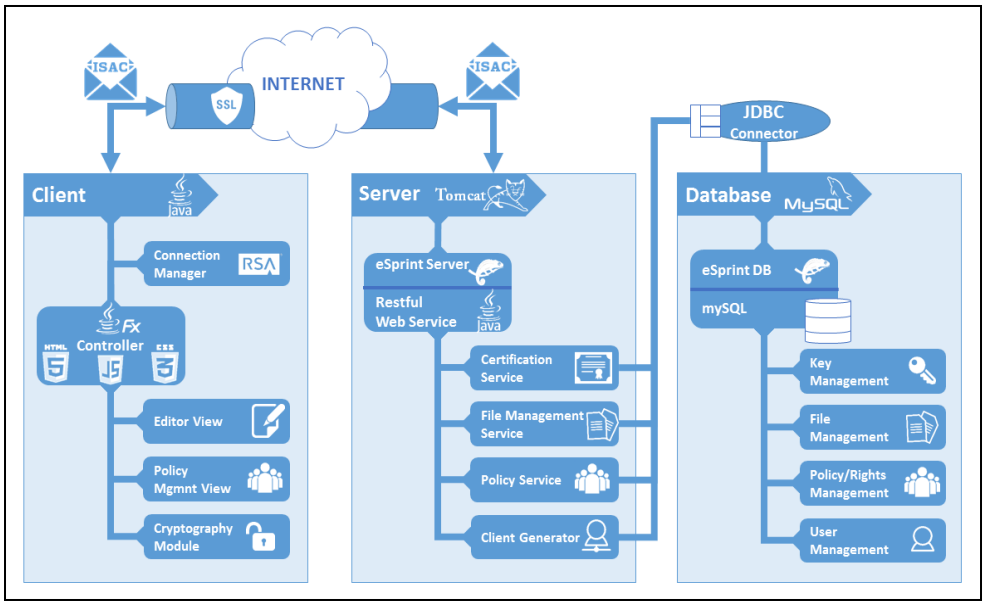

Abbildung 42: Architekturübersicht

Wie bereits in Kapitel 4 angedeutet, stellt das Diagramm in Abbildung 42 eine detailliertere Ansicht der hier verwendeten 3-Tier-Architektur dar. Hierdurch wird ein

grobgranularer Einblick in die angebotenen Funktionen beziehungsweise in die Struktur des eSprint-Systems gewährt.

8.1.1 Client

Hinsichtlich des Clients ist aus der Abbildung zu entnehmen, dass dieser auf Java basiert. Dies ermöglicht einen plattformunabhängigen Betrieb der Software und steht für den modernen Ansatz einer Multiplattform-Anwendung. Um eine Verbindung mit dem Server herstellen zu können, wird das Connection-Manager-Modul benötigt. Dieses etabliert eine gesicherte Verbindung zum eSprint-Server und verwaltet die Versendung sowie den Empfang der ausgetauschten Daten. Die Sicherheit dieser Transfers wird mittels diverser Verschlüsselungstechnologien bewerkstelligt, wovon die RSA-Technologie die Kernaufgaben erfüllt.

Das Frontend und die Steuerung des Connection-Managers erfolgen über ein zentrales Controller-Element. Über dieses werden auch die verschiedenen Funktionen des Clients angesteuert, initialisiert und initiiert. Die wichtigste Technologie ist in diesem Fall JavaFX, das die Erzeugung einer grafischen Benutzeroberfläche ermöglicht und jegliche Interfaceinteraktionen ermöglicht. Hierzu gehört der – für den Endbenutzer essentielle – Texteditor (*EditorView*), worüber sich neue oder bestehende Texte editieren lassen. Die Integration mehrerer JavaFX-Webviews ermöglichen hier auch den Einsatz von HTML5, JavaScript sowie CSS3-Elementen und geben so ein breites Spektrum an Gestaltungs- beziehungsweise Interaktionsmöglichkeiten für den Benutzer. Als Beispiel hierfür kann der TinyMCE-Texteditor angeführt werden, der ein auf JavaScript basierender und auf die vorliegenden Anforderungen angepasster Open-Source-WYSIWYG-Editor für Webanwendungen ist.

Ebenfalls auf den vier Säulen – JavaFX, JavaScript, CSS3 und HTML5 – wurde der *Policy Manager* entworfen. Dieser ermöglicht dem Benutzer die Projektverwaltung, die Rechtevergabe und die Anlegung neuer Benutzer. Diese User-Interface-Funktionalitäten werden in bildlicher Darstellung in Kapitel 12 auf Seite 182 veranschaulicht.

Das *Cryptography Module* nimmt die in dem ISAC-Zertifikat enthaltenen Schlüssel und entschlüsselt hiermit die vom Server übermittelten Nutzdaten und stellt sie dem Benutzer über den Editor View zur Verfügung.

8.1.2 Server

Das eSprint-System setzt auf einem *Apache Tomcat Server* auf und ist ein in Java geschriebener Open-Source-Webserver. Dieser erlaubt es, die vom eSprint-System bereitgestellten Webanwendungen und Services des hier verwendeten Clients zur Verfügung zu stellen.

Die hier bereitgestellten Web-Services basieren auf dem Programmierparadigma REST (Representational State Transfer), das durch seine Einfachheit unter Verwendung von Standard-Operationen des HTTP-Protokolls ideal in der Architektur Platz findet. Die Implementierung dieser *RESTful Services* wurde mittels Java realisiert und unter Berücksichtigung eines möglichst modularen Aufbaus umgesetzt.

Der *Certification-Service* ist für die Erstellung des ISAC-Zertifikats und dessen Übermittlung zuständig. Konkret werden hier zunächst die für das Zertifikat notwendigen RSA-Keys erstellt, die dazugehörigen Attribute dynamisch mittels der Userdatenbank generiert, zu einem Zertifikat vereinigt, abschließend signiert und an den serviceaufrufenden Client übermittelt. Dies geschieht unter der Zuhilfenahme der *Bouncy Castle Crypto API*, die eine Sammlung quelloffener kryptographischer Programmierschnittstellen für Java bereitstellt. Hierdurch werden alle für die hier benötigten Funktionen zur sicheren Kommunikation und Erstellung des ISAC-Zertifikats abgedeckt [BCC14].

Das Modul *File-Management-Service* dient der Übermittlung und Verwaltung der gespeicherten Nutzdaten. Konkret werden auf diese Weise die Prozesse Laden und Speichern einer Datei abgewickelt. Für diese Vorgänge stellt der Service eine Schnittstelle sowohl zum Abruf der vorhandenen Projekte als auch zur Anzeige aller Dateien innerhalb eines Projektes bereit.

Das *Policy-Service-Module* ist für die Verwaltung aller Zugriffs- und Rechtemanagement-Operationen verantwortlich. Zu diesen zählen:

- *Liste aller dem Serviceaufrufer zugehörigen Gruppen*

  Über diese werden alle zu dem aktiven Projekt gehörigen Gruppennamen einer Ressource, denen der Serviceaufrufer angehört, zurückgegeben.
- *Liste aller Gruppen zu einer Ressource*

  Liefert eine Liste aller vorhandenen Projektgruppen der aktiven Ressource.
- *Gruppeninformationen abrufen*

  Über diese Schnittstelle wird dem Servicenutzer der Gruppenname, deren Zugriffsrechte bezüglich der zugehörigen Ressource als auch das Ablaufdatum dieser bereitgestellt. Letzteres dient einer zeitlichen Begrenzung der Zugriffsmöglichkeiten auf die Ressource durch die jeweilige Gruppe.
- *Abfrage der Benutzerrechte innerhalb des aktiven Projektes*

  Diese angebotene Funktion ermöglicht die Ermittlung der Benutzerrechte des Serviceaufrufers innerhalb des aktiven Projektes. Dies dient der Clientanwendung zur Anpassung der dem Benutzer bereitgestellten Programmfunktionen.
- *Gruppeninformationen bearbeiten*

  Diese Schnittstelle ermöglicht es dem Servicenutzer, jegliche Informationen einer Gruppe den gewünschten Spezifikationen anzupassen. Beispielsweise ist es so möglich, den Gruppennamen zu modifizieren, das Gültigkeitsdatum der Gruppe zu verändern, deren Rechte neu zu vergeben oder die gesamte Gruppe samt ihrer Mitglieder aus dem Projekt zu entfernen.
- *Erstellen neuer Gruppen*

  Wie es der Name impliziert, ist es hierüber möglich, neue Gruppen mit Rechten und Gültigkeitsdatum anzulegen.
- *Liste aller Benutzer einer Gruppe bereitstellen*

  Auf diese Weise wird eine vollständige Liste aller einer Gruppe zugewiesenen Benutzer zurückgegeben.
- *Benutzer einer Gruppe zuweisen*

  Hier können bestehende Benutzer einer Gruppe zugewiesen werden.

- *Informationen zu Dateiadministrator*
  Sollte der aktuelle Nutzer einer Ressource nicht der Ersteller dieser Datei sein, so ermöglicht diese Schnittstelle es ihm, Informationen über den Eigentümer zu bekommen.
- *Anlegen eines neuen Benutzers*
  Diese Funktion entspricht der ebenfalls angebotenen Frontend-Funktion des Servers, welche die Erstellung eines Benutzers ermöglicht und in einem weiteren Schritt eine automatisch generierte E-Mail mit einem Downloadlink zu dem individualisierten Client an die angegebene E-Mail-Adresse versendet. Dieses Modul bedient sich des Services *Client-Generator*.

Das letzte Modul der Abbildung beschreibt den *Client-Generator*. Dieser dient der Erstellung neuer Clientprogramme, über die auf das ausstellende eSprint-System zugegriffen werden kann. In diesem Schritt wird ein personalisiertes Zertifikat erstellt, das der übergeordneten CA vertraut und kombiniert mit einer Kopie der Clientsoftware als Download zur Verfügung gestellt wird.

8.1.3 Database

Im Folgenden wird die dritte Stufe der vorliegenden Systemarchitektur, die *Database*, betrachtet. Die im eSprint-System zum Einsatz kommende Datenbank basiert auf der Open-Source-Software MySQL und ist eine der weltweit am meistverbreitetsten Datenbanktechnologien [DBR16].

Die Hauptaufgabe dieses Teils der Architektur ist es, die persistente Datenhaltung zu gewährleisten und performant für die Services zur Verfügung zu stellen. Die relevantesten Bestandteile der Datenbank stellen hierbei dar: das Key-Management zur Verwaltung der verschlüsselungsrelevanten Informationen, das File-Management zur Verwaltung der durch den Benutzer erstellten Textdateien inklusive der vergebenen Rechte innerhalb dieser Datei, das Policy- beziehungsweise Rights-Management zur Verwaltung der Gruppenrichtlinien und deren Mitglieder, inklusive der Rechtevergabe zu jeweiligen Projekten, und schlussendlich das User-Management zur Verwaltung der

erstellten und authentifizierten Clients, die Zugriff auf das eSprint-System haben und somit in Projekten hinzugefügt werden können.

Eine detaillierte Beschreibung des Datenbankaufbaus und der darin abgebildeten Beziehungen der Tabellen und Datensätze untereinander erfolgt im Kapitel 10.2.

8.1.4 Datentransfer

Der letzte Punkt der Systemarchitektur widmet sich den Verbindungen zwischen den drei Stufen des Modells. Zum einen umfasst dies die Datenverbindung zwischen dem eSprint-Server und seiner Datenbank und zum anderen die Verbindung zwischen dem Server und der Clientanwendung, der in diesem Projekt eine besondere Stellung eingeräumt wird.

Abbildung 43: JDBC-Verbindung

Wie in Abbildung 43 dargestellt, wird zunächst die Verbindung zwischen dem Server und der Datenbank betrachtet, so ist ihre Realisierung durch einen sogenannten *JDBC-Connector* durchzuführen. JDBC bedeutet Java-Database-Connectivity und ist eine einheitliche Datenbankschnittstelle zu Datenbanken verschiedener Hersteller. In diesem Projekt bedeutet dies eine Verbindung mit der MySQL-Datenbank, jedoch gewährleistet die Verwendung dieses Connectors auch die Möglichkeit einer nachträglichen Austauschbarkeit der genutzten Datenbanktechnologie. Über die angesprochene API werden Datenbankverbindungen aufgebaut und verwaltet, SQL-Anfragen werden an die Datenbank weitergeleitet und deren Antworten werden in einer für Java nutzbaren Form dem Programm zur Verfügung gestellt.

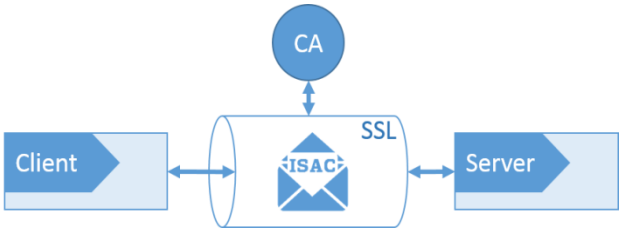

Abbildung 44: Client-Server-Kommunikation

Wird nun der wesentliche Part bei der Verbindungsbeschreibung, nämlich der Datentransfer zwischen dem Server und Client, in Abbildung 44 betrachtet, so lassen sich gleich mehrere Sicherheitstechnologien erkennen, die hier zum Einsatz kommen. Um eine Verbindung zwischen dem Server und Client aufbauen zu können, muss zunächst eine SSL-Verbindung etabliert werden. Der Ablauf und das Konzept hierfür wurden bereits ab Kapitel 5.2 genauer beschrieben. Sowohl der Client als auch der Server vertrauen zum Aufbau der Verbindung derselben CA. Innerhalb dieser gesicherten Verbindung ist es möglich, Daten sicher auszutauschen, weshalb die weiter oben beschriebenen Serviceaufrufe des Clients an den Server geschützt durchgeführt werden können. Diese Art des Informationsaustausches entspricht den aktuellen Standards, wie sie auch bei regulären HTTPS-Anfragen im Internet stattfinden.

Um aber eine gesteigerte Sicherheit innerhalb dieses Kanals einzuführen, wurde im Zuge dieses Projektes das zentrale Element dieser Arbeit – das *Identification-and-Secure-Access-Control*-Zertifikat – entwickelt. Dieses dient der Übermittlung der vom Client benötigten Keys, die das Ver- und Entschlüsseln der ausgetauschten Nutzdaten ermöglichen. Dies wurde bereits im Kapitel 6 genauer erläutert.

Darüber hinaus wird das beschriebene ISAC-Zertifikat durch den sicheren Datencontainer Digital Envelope geschützt. Durch dieses dreiteilige Sicherheitskonzept wird ein optimaler Sicherheitsraum zwischen dem Client und Server geschaffen.

8.2 Systemkomponenten

Die nachfolgenden Komponentendarstellungen gewähren einen tiefergehenden Einblick in den Aufbau der Systemlandschaft und der Komponenten, aus denen sich

die Darstellungen zusammensetzen. Dafür wurden die essentiellen Bestandteile sowohl des Servers als auch des Clients abgebildet, um deren aggregierte Funktion darzustellen. In den Abbildungen sind die diversen Komponenten jeweils zu einer übergeordneten Gruppe zusammengefasst, die die Hauptfunktion der betrachteten Komponente widerspiegeln soll. Wie in Abbildung 45 zu sehen ist, wurde das Diagramm in einen Server- und einen Clientpart unterteilt, was lediglich einer besseren Übersicht dienen soll.

8.2.1 Server

Der Server setzt sich, wie in Abbildung 45 dargestellt, aus acht Hauptgruppierungen zusammen.

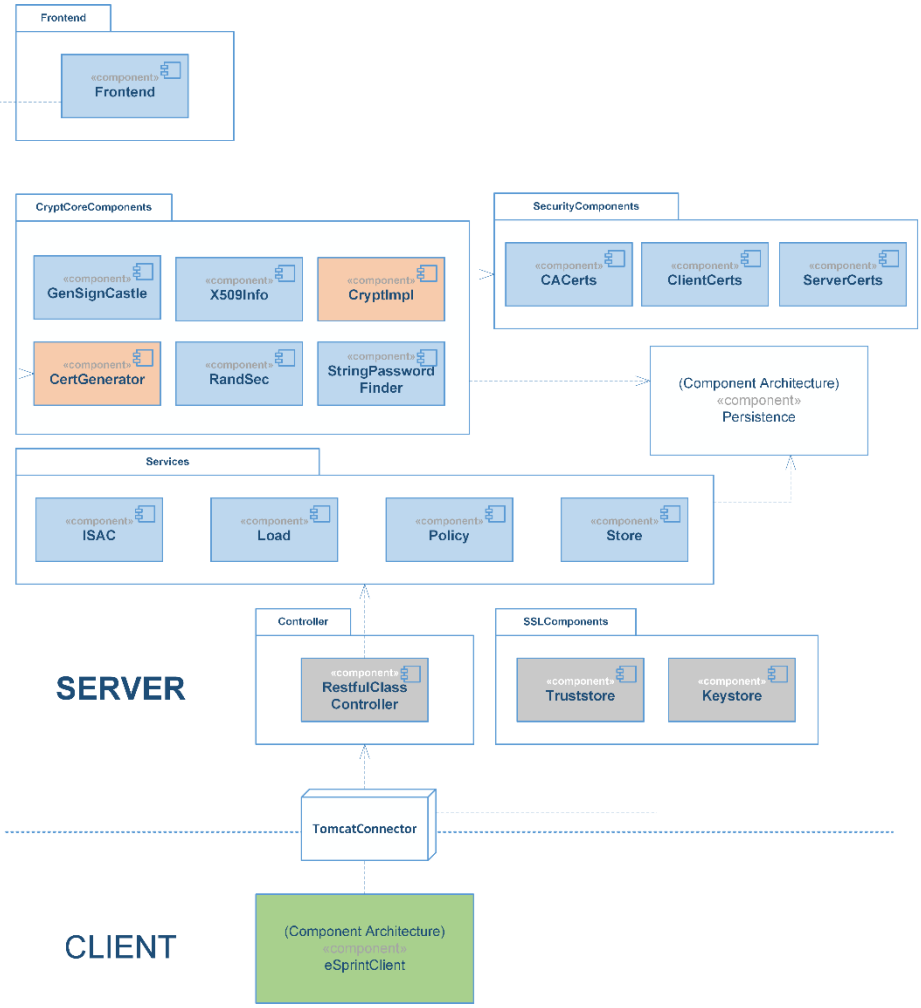

Abbildung 45: Komponentendiagramm – Server

- *Tomcat-Connector*

  Der Tomcat-Connector stellt den zentralen Zugangspunkt des Servers für die servicekonsumierenden Clients dar. Er definiert über seine Konfigurationsdatei *server.xml* einerseits, über welche Ports Requests behandelt werden. Hierzu zählt auch die Festlegung, dass der Service beim Datenaustausch mit den eSprint-Clients ausschließlich über gesicherte SSL-Verbindungen kommunizieren darf. Andererseits werden hier der Key- sowie der Truststore definiert.

- *SSL-Components*

  Hierin werden sowohl der Trust- als auch der Keystore gespeichert. Der Truststore beinhaltet eine Liste von CAs, denen vertraut werden kann. Im Keystore wird das servereigene x.509-Public-Certificate gespeichert, das den Aufbau einer SSL-Instanz für einen Service durch eine Controllerklasse ermöglicht.
- *Controller-Components*

  Diese Gruppe beinhaltet die RESTful-Class-Controller, der für die Verteilung der eintreffenden Serviceanfragen an die jeweiligen Serviceimplementierungen weiterleitet.
- *Services*

  Diese Gruppe beinhaltet die Implementierungen aller angebotenen Services des eSprint-Systems. Dies umfasst die ISAC, Load, Policy und Store-Services.
- *Crypt-Core-Components*

  Hierin werden alle für die Kryptographie relevanten Komponenten zusammengefasst. Diese beinhalten die beiden Hauptkomponenten *CertGenerator* und *CryptImpl* sowie die Hilfskomponenten *GenSignCastle, X509Info, RandSec* und *StringPasswordFinder*. Die Hauptkomponente CryptImpl stellt hierbei alle Kryptographie-Funktionen für die angebotenen Services bereit. Der *CertGenerator* bietet die Funktionen zur dynamischen Erstellung von neuen Clientanwendungen und deren Zertifikaten.
- *Security-Components*

  In diesem Teilbereich des Servers werden alle von der Crypt-Core-Components-Gruppe benötigten Zertifikate bereitgestellt. Diese umfasst die *CACerts, ClientCerts* und *ServerCerts.*
- *Persistence-Components*

  Diese Komponente stellt die im Kapitel 10.2 Datenmodell näher beschriebene Datenbank des eSprint-Servers dar.

- *Frontend*

  Das Frontend bietet dem Benutzer eine Schnittstelle zum Service, um neue Client-Anwendungen generieren und abrufen zu können.

8.2.2 Client

Anschließend wird die schematische Darstellung der Clientanwendungen erläutert. Auf diese Weise lässt sich erkennen, dass die Hauptaufgabe des Clients in der Darstellung und Bearbeitung der durch den Server bereitgestellten Services ist. So stellen die Interface- und Kryptographie-Gruppen dessen Hauptkomponenten dar.

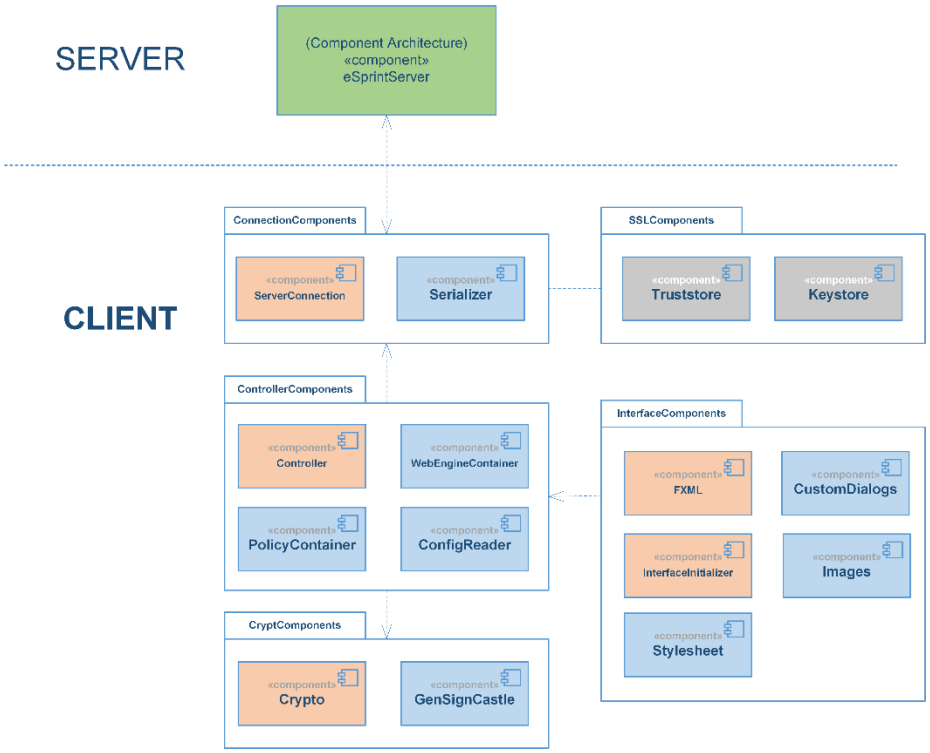

Abbildung 46: Komponentendiagramm – Client

- *Connection-Components*

  In dieser Gruppe befindet sich die Komponente *ServerConnection,* die für die Etablierung einer sicheren Verbindung mit dem Server und der Übermittlung der Daten verantwortlich ist. Zusätzlich beinhaltet die Gruppe die Hilfskomponente *Serializer*, die für die Aufbereitung der zu sendenden und empfangenen Daten sorgt.

- *SSL-Components*

  Wie auch im Server werden auch hier sowohl der Trust- als auch der Keystore gespeichert. Dies dient zur Verwaltung der Zertifikate, zum Aufbau einer sicheren Verbindung zum Server und zur Verschlüsselung der Nutzdaten.

- *Controller-Components*

  Diese Gruppe beinhaltet die Kernkomponente *Controller* der Clientanwendung. Über diese werden alle Interaktionen des Nutzers verarbeitet und mittels der Komponenten *WebEngineController, PolicyContainer, ConfigReader* gesteuert. Diese Gruppe ist demnach maßgeblich für das Zusammenspiel der Interfaceaktionen und der Serverinteraktionen sowie der daraus bedingten Serviceaufrufe verantwortlich. Sowohl der *WebEngineController* als auch der *PolicyContainer* kümmern sich um die Verwaltung der internen Variablen sowie des Keymanagements.

- *Crypt-Components*

  Hier werden alle für die Kryptographie relevanten Komponenten zusammengefasst. Diese bestehen aus den Hauptkomponenten *Crypto* sowie den Hilfskomponenten *GenSignCastle*. Die Hauptkomponente Crypto stellt hierbei alle Kryptographie-Funktionen zur Ver- und Entschlüsselung des Datenverkehrs mit dem Server bereit.

- *Interface-Components*

  Hierin befinden sich die Hauptkomponenten der für eine JavaFX-Anwendung üblichen Interfacedefinition in Form einer *FXML*-Komponente sowie eines *InterfaceInitializer,* der für die Instanziierung der Controllerklasse und der FXML zuständig ist. Weiters befinden sich in der Gruppe die Komponente

CustomDialogs, die zur Darstellung von Dialogfenstern genutzt wird. Außerdem werden die Komponenten *Stylesheets* und *Images* zur Visualisierung benötigt.

9 Prozessbeschreibung

Nach der Darstellung der Systemarchitektur und deren Komponenten erfolgt nun die Beschreibung der einzelnen Prozesse. Diese sollen einerseits die Funktionsweise des gesamten eSprint-Umfeldes anhand von konkreten Anwendungsfällen veranschaulichen und andererseits die für den Benutzer angebotenen, möglichen Operationen aufzeigen. Um diese Prozesse darüber hinaus in einen geeigneten Kontext zu bringen, wird auf den in Kapitel 2 vorgestellten Use-Case zurückgegriffen.

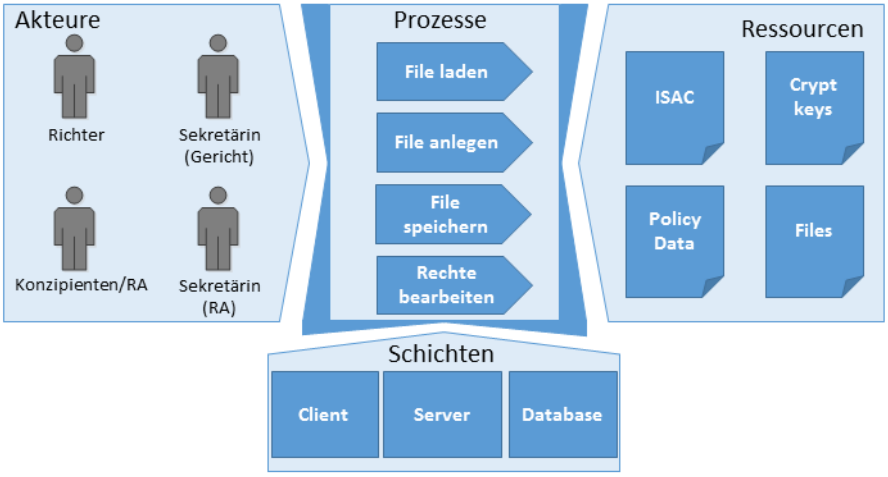

Abbildung 47: Übersicht – Prozesslandschaft

Wie in Abbildung 47 zu erkennen ist, werden nur jene vier Hauptprozesse dargestellt, die auch alle drei Schichten der Architektur nutzen – und zwar *File laden*, *File anlegen*, *File speichern* und *Rechte bearbeiten*. Die diversen Unterfunktionen wie beispielsweise die des Texteditors werden hier ausgespart, da sie lediglich Sub- oder Nebenprozesse zu diesen darstellen. Wie im Anwendungsfall beschrieben, können die nachfolgend dargestellten Prozesse von den vier abgebildeten Akteuren ausgeführt werden. Diese werden durch einen das Urteil schreibenden Richter, die mit Verwaltungsaufgaben betrauten Sekretärinnen sowie die Rechtsanwälte

beziehungsweise deren Konzipienten verkörpert. Die relevanten Ressourcen für die dargestellten Prozesse umfassen dabei die erstellten Nutzdaten des Benutzers, die dazugehörigen Policy-Daten zur Verwaltung der Zugriffsrechte, die Dateischlüssel zur Sicherstellung der Vertraulichkeit der verteilt genutzten Ressource sowie das ISAC-Zertifikat zur Verteilung eben dieser Schlüssel.

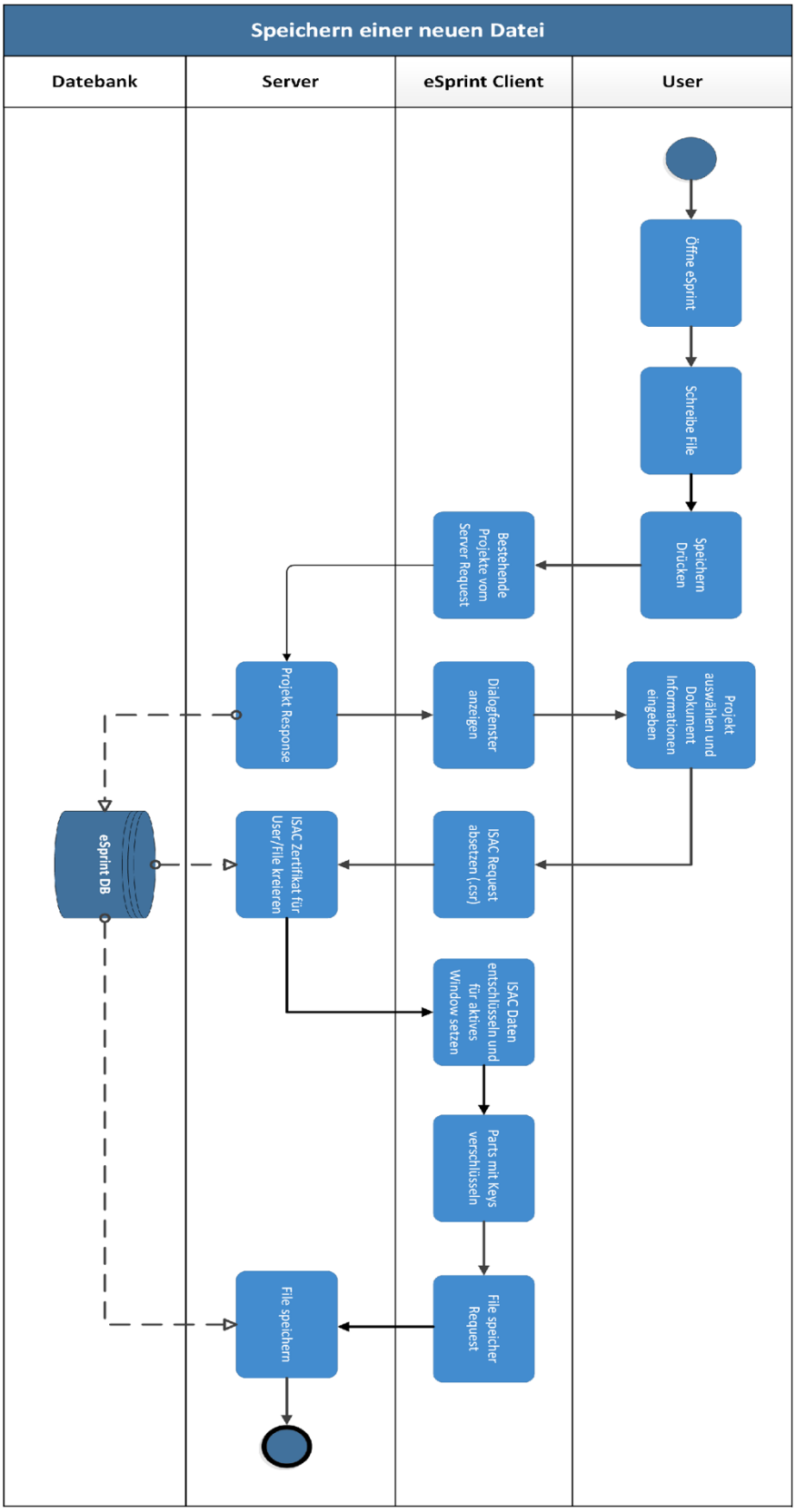

Abbildung 48: Ablaufdiagramm – neue Datei speichern

9.1 Neue Datei speichern

In der oben dargestellten Abbildung 48 wird der erste der vier Hauptprozesse, der Vorgang des Speicherns einer neuen Datei, in seine Teilprozesse untergliedert und näher erläutert. Um ein möglichst vollständiges Bild dieses Ablaufs darstellen zu können, beginnt dieser beim tatsächlichen Start des eSprint-Clients auf Seiten des Benutzers und endet beim Persistieren der Daten auf der Serverseite.

- *eSprint öffnen*
  Dieser Prozess startet mit dem Öffnen des eSprint-Clients durch den Benutzer.
- *File schreiben*
  In diesem Schritt wird eine neue Datei durch den Benutzer mit Informationen gefüllt. Hierzu zählen einerseits die Nutzdaten, aber auch die dazugehörigen Policy-Informationen, wie etwa die Definierung der einzelnen Textabschnitte und deren Access-Arten. Darüber hinaus lassen sich die Dokumenteneigenschaften (Document Properties) und diverse zusätzliche Textformatierungen festlegen.
- *Speichern drücken*
  Hier wählt der Benutzer aus dem Menü „File“ den Punkt „Save“ oder „Save as“. Bei einer neuen Datei führen beide Funktionen zum selben Dialogfenster.
- *Bestehende Projekte vom Server-Request*
  Dieser Schritt bewirkt eine Anfrage durch das Clientprogramm an den Server bezüglich aller existierenden Projekte.
- *Server-Project-Response*
  In diesem Punkt findet der Zugriff des Servers auf die eSprint-Datenbank statt, aus der er eine Liste der bereits existierenden Projekte generiert und an den Client zurücksendet.
- *Dialogfenster anzeigen*
  Das nun automatisch vom eSprint-Client geöffnete Dialogfenster filtert, sortiert und präsentiert dem Benutzer der Clientanwendung die durch den Server übermittelten Daten.

- *Projekt auswählen und Dokumentinformationen eingeben*

  Im dargestellten Dialogfenster kann der Benutzer nun entweder eines der bereits vorhandenen Projekte als Speicherort für die neu erstellte Datei auswählen oder ein eigenes Projekt anlegen. Weiters kann er hier, falls nicht schon zuvor über die Dokumenteneigenschaften festgelegt, die spezifischen Informationen des Dokumentes (Titel, Beschreibung, Keywords) definieren. Um den Vorgang abzuschließen, muss der Benutzer nochmals auf „Speichern" drücken.

- *ISAC-Request absetzen (.csr)*

  Nun erstellt der eSprint-Client zunächst eine ISAC-Zertifikatsanfrage (kurz CSR[15]). In diese Anfrage werden zum einen die Identitätsinformationen des ausstellenden Clients und zum anderen ein eigens hierfür erstelltes 4096-Bit-starkes Schlüsselpaar integriert. Nachdem diese Schritte auf Seiten des Clients abgehandelt sind, wird das erstellte CSR an den Server übermittelt. Hierfür wird dieses nochmals durch den Client mittels eines SHA256withRSA-Algorithmus signiert, um eine zweifelsfreie Identifikation des Clients durch den Server zu ermöglichen. Zur zusätzlichen Sicherstellung eines Secure-Channels werden die Daten mittels des in Kapitel 7.2 erläuterten Digital Envelopes transferiert.

- *ISAC-Zertifikat für User/File kreieren*

  Nachdem der eSprint-Server das CSR des Clients erhalten hat, generiert er daraus das individuell angepasste ISAC-Zertifikat (CRT[16]) für die betreffende Ressource. Bei der Erzeugung einer neuen Datei wird der erstellende Benutzer automatisch als Eigentümer der Datei (Owner) gesetzt und ein neuer Satz Access-Keys werden zur Verschlüsselung der vier möglichen Zugriffsarten Open, Community, Organisation und Restricted sowohl in die Datenbank als auch in das Zertifikat geschrieben. Weiters wird ein Hauptschlüssel zur nochmaligen Verschlüsselung des gesamten Dokumentes erstellt und gespeichert. Abschließend wird durch den Server noch eine fortlaufende Dokumentenidentifikationsnummer (DocID) generiert und eingefügt. Diese ist mitels asynchroner Verschlüsselunurch den Public Key des Servers geschützt und

[15] CSR – Certificate Signing Request
[16] CRT – Certificate

liegt dem Client niemals in lesbarer Form von Klartext vor. Nach der Vollendung dieser Arbeitsschritte wird das Zertifikat signiert und an den eSprint-Client zurückgesendet, wo es zur weiteren Verwendung in der aktiven Session zur Verfügung steht.

- *ISAC Daten entschlüsseln und für aktives Window setzen*

  In diesem Prozessschritt wird das empfange ISAC-Zertifikat zunächst entschlüsselt und danach werden die darin enthaltenen Informationen ausgelesen. Diese werden für den jeweiligen Reiter innerhalb des eSprint-Clientfensters, in dem die Aktionen durch den Benutzer durchgeführt werden, im Hintergrund temporär gespeichert.

- *Parts mit Keys verschlüsseln*

  Nachdem die Clientanwendung das neu erstellte ISAC-Zertifikat erhalten hat, werden mit den darin übermittelten Keys die jeweiligen Inhalte je nach der entsprechenden Access-Art verschlüsselt. Abschließend wird das gesamte Dokument nochmals mithilfe eines weiteren Schlüssels gesichert.

- *File-Speicher-Request*

  Hier wird das soeben verschlüsselte Dokument an den Server übermittelt, um eine Persistierung der neu erstellten Datei auf Serverseite anzustoßen. Die übermittelten Daten umfassen sowohl das Dokument selber als auch dessen, nur vom Server entschlüsselbare, DocID und die allgemeinen Dokumentenmetadaten (Title, Description, Keywords).

- *File speichern*

  Der eSprint-Server liest zunächst die empfangene DocID aus und entschlüsselt diese mithilfe seines internen Private Keys. Danach speichert er die übermittelten Nutzdaten in verschlüsselter Form in die Datenbank, von wo aus sie durch andere Projektteilnehmer abgerufen werden können.

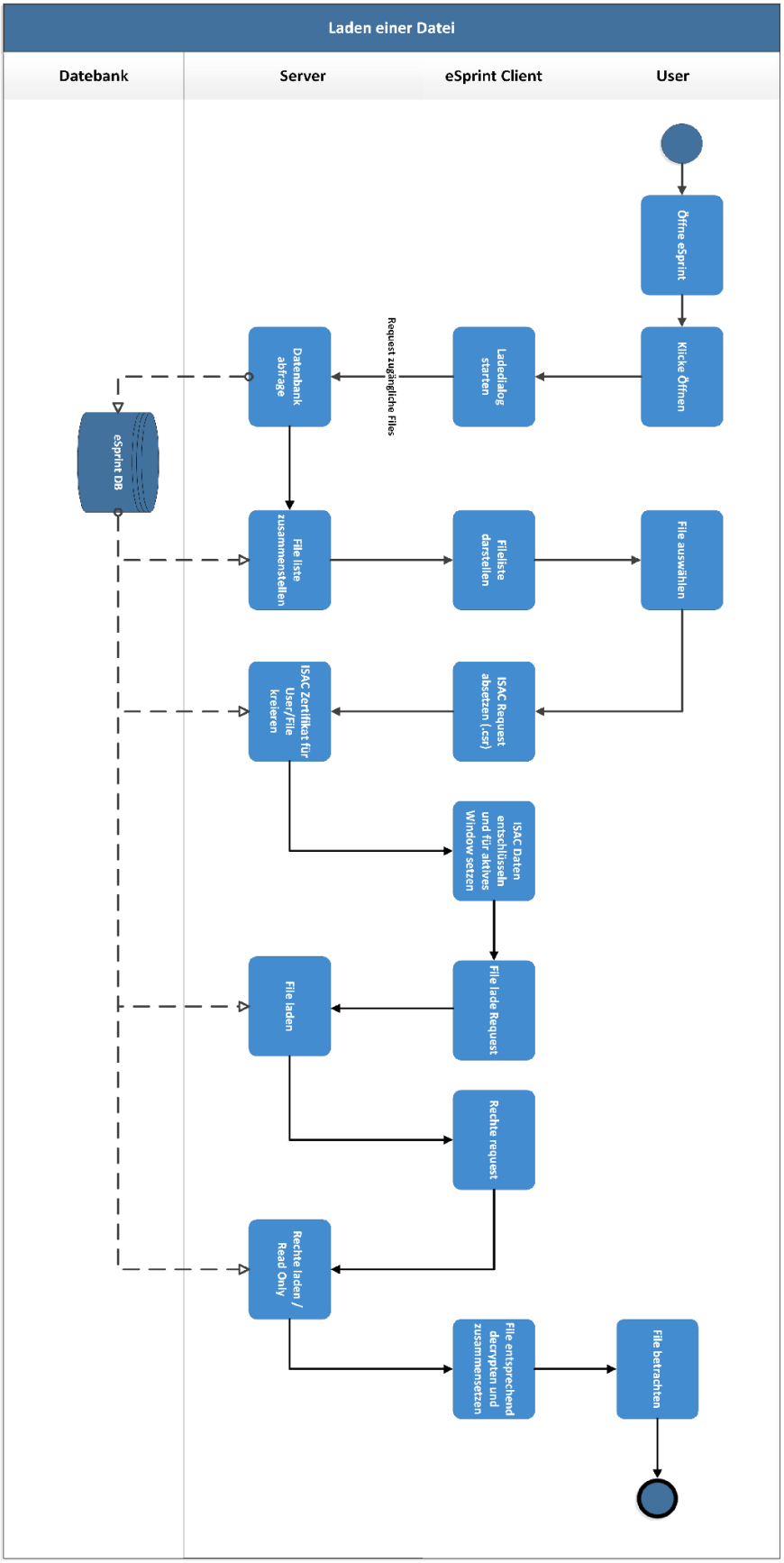

Abbildung 49: Ablaufdiagramm – Datei laden

9.2 Datei laden

Der in Abbildung 49 dargestellte Prozess startet durch das Öffnen der Clientanwendung durch den Benutzer und zeigt den Durchlauf der gesamten Prozesskette bis hin zur Anzeige der entsprechend angeforderten Datei im Editor des eSprint-Clients. Konkret handelt es sich im nachfolgend beschriebenen Prozess um den Vorgang des Ladens einer Datei durch den Benutzer.

- *Öffne eSprint*
  Zunächst muss der eSprint-Client durch den Benutzer gestartet werden.
- *Öffnen klicken*
  Um eine vorhandene Datei laden zu können, muss durch den Benutzer der Punkt „Load…“ aus dem Reiter „File“ ausgewählt werden.
- *Ladedialog starten*
  Durch diese Aktion des Benutzers wird im Hintergrund vom eSprint-Client der Prozess zum Laden einer Datei gestartet. Hierfür sendet er einen Request an den Server, um die für ihn mit dem vorhandenen Zertifikat zugänglichen Dateien als Liste anzufordern.
- *Datenbankabfrage*
  Auf Seiten des Servers wird nun eine Datenbankabfrage bezüglich der zugänglichen Dateien durchgeführt.
- *File-Liste zusammenstellen*
  Wenn der Server alle notwendigen Informationen angefragt hat, wird das Resultat in Form einer Response mit einer Liste an den Client zurückgegeben. Wichtig hierbei ist, dass er jeden Eintrag mit einer eindeutig zuzuordnenden Identifikationsnummer versieht, die wiederum aus der Datenbank ausgelesen wird.
- *File-Liste darstellen*
  Nun kann der Client die angeforderte Liste in einem neuen Ladedialogfenster („Filebrowser“) dem Benutzer präsentieren.

- *File auswählen*

  Wie der Name bereits vermuten lässt, kann der Benutzer nun wählen welche Datei er über den eSprint Client laden will.

- *ISAC-Request absetzen (.csr)*

  In diesem Schritt beginnt nun der Prozess der Zertifikatsgenerierung, wie bereits im Kapitel 9.1 bei dem Punkt „ISAC Request absetzen (.csr)“ beschrieben. Dieser unterscheidet sich einzig dadurch, dass beim Laden einer bestehenden Datei die zugehörige verschlüsselte Ressource-ID oder Identifikationsnummer der ausgewählten Datei bereits im CSR mitgesendet wird. Diese kann nur durch den Server mittels seines Public Keys entschlüsselt und der Datei zugeordnet werden. Auf diese Weise wird eine eindeutige und ausschließliche Nutzung der jeweils gewählten Ressource mithilfe des generierten Zertifikats gewährleistet.

- *ISAC-Zertifikat für User/File kreieren*

  Nachdem der eSprint-Server das CSR des Clients erhalten hat, generiert er daraus das individuell angepasste ISAC-Zertifikat (CRT) für die betreffende Ressource. Beim Laden einer Datei wird der aufrufende Benutzer zunächst durch den Server und dessen Datenbank identifiziert und dessen Zugriffsrechte (Open, Community, Organisation und Restricted) werden ausgelesen. Danach wird die jeweilige DocID in das ISAC-Zertifikat eingefügt. Diese ist mittels asynchroner Verschlüsselung durch den Public Key des Servers geschützt und liegt dem Client niemals in lesbarer Form von Klartext vor. Nach der Vollendung dieser Arbeitsschritte wird das Zertifikat signiert und an den eSprint-Client zurückgesendet, wo es zur weiteren Verwendung in der aktiven Session zur Verfügung steht.

- *ISAC-Daten entschlüsseln und für aktives Window setzen*

  In diesem Prozessschritt wird das empfangene ISAC-Zertifikat zunächst entschlüsselt und danach werden dessen enthaltenen Informationen ausgelesen. Diese werden dann für den jeweiligen Reiter innerhalb des eSprint-Clientfensters, in dem die Aktionen durch den Benutzer durchgeführt werden, im Hintergrund temporär gespeichert.

- *File-Lade-Request*

  Bei diesem Schritt wird lediglich die verschlüsselte DocID vom Client an den Server übermittelt. Dieser überprüft nochmals die Identität des anfragenden Clients und übermittelt anschließend die verschlüsselte Ressource.

- *File laden*

  Nachdem der Client die Ressource erhalten hat, wird zunächst die Herkunft der Datei mittels der Signatur überprüft. Stammt diese zweifelsfrei vom Server, wird das Dokument mittels des übergeordneten Dokumentenschlüssels entschlüsselt. Hierdurch erhält der Client den Titel, die Beschreibung, die Keywords des Dokuments sowie die einzeln nochmals verschlüsselten Parts der Ressource.

- *Rechte-Request*

  Die Request dient dazu, die Rechte des Benutzers nochmals anzufordern. Dies bezweckt primär die Bereitstellung der Zusatzfunktionalität eines reinen Lesemodus ohne jegliche Bearbeitungsmöglichkeit. Hierdurch können einzelne, besonders sensible Passagen eines Textes vor unberechtigten Veränderungen durch andere beteiligte Parteien geschützt werden. Ein Beispiel dafür sind Daten zu Fristen, die eingehalten und nicht abgeändert werden dürfen.

- *Rechte laden/Read only*

  Der Server überprüft nun nochmals die Rechte des Benutzers und übergibt diese samt den allfälligen Read-Only-Einschränkungen zurück an den Client.

- *File entsprechend decrypten und zusammensetzen*

  Nachdem der Client alle für ihn relevanten Informationen zu der betreffenden Ressource erhalten hat, kann er diese entsprechend seiner Rechte entschlüsseln und dem Editormodul zur Darstellung übergeben.

- *File betrachten*

  Nun wird dem Benutzer das Dokument mittels des Texteditors präsentiert und zur weiteren Bearbeitung zur Verfügung gestellt. Hierbei werden aufgrund von fehlenden Berechtigungen nicht einsehbare Teile des Textes mittels schwarzen Platzhaltern in Form von Balken dargestellt und Read-Only-Parts durch ausgegraute Bereiche abgebildet.

9.3 Bestehende Datei speichern

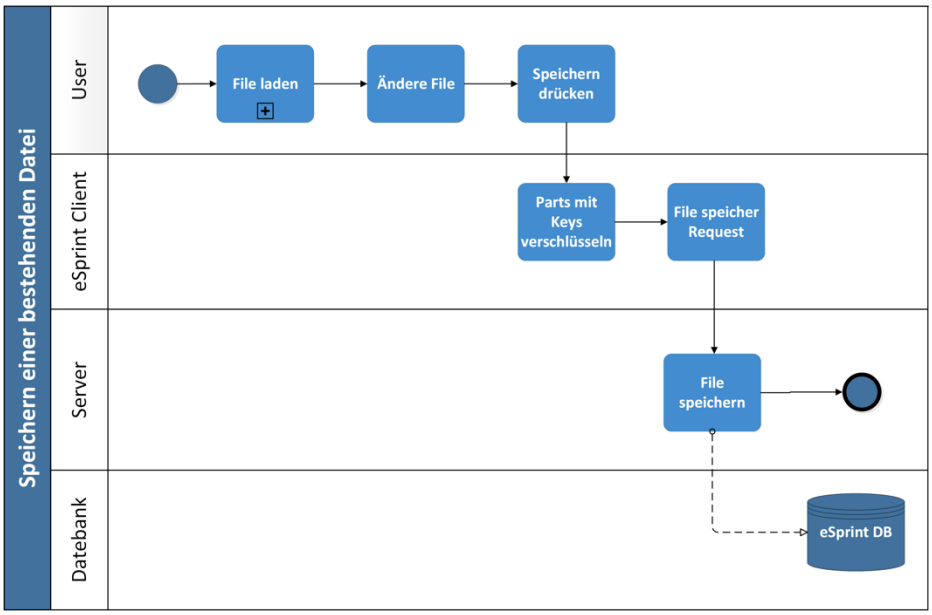

Abbildung 50: Ablaufdiagramm – Speichern einer bestehenden Datei

- *Datei laden*

  Der Prozess beginnt zunächst – wie im Kapitel 9.2 beschrieben – mit dem Durchlaufen des Subprozesses Laden einer Datei.

- *File ändern*

  Bei diesem Schritt kann der Benutzer den vorliegenden Text entsprechend den ihm zugeteilten Rechten manipulieren. Konkret bedeutet dies, dass er neue Passagen hinzufügen und bestehende Abschnitte verändern sowie löschen kann.

- *Speichern drücken*

  Nachdem alle gewünschten Änderungen an der Datei durchgeführt wurden, kann der Benutzer diese mittels Drücken des Unterpunktes „Save“ vom Menüpunkt „File“ speichern. Dadurch wird die bestehende Datei überspeichert. Sollte dies nicht gewünscht sein, kann anhand alternativer Betätigung des Menüpunktes „Save as…“ eine neue Datei erzeugt werden.

  Nach der Interaktion des Benutzers erfolgt eine Sicherheitsprüfung durch den Client, ob alle relevanten Informationen durch den Benutzer gesetzt wurden.

Diese Metadaten der Datei umfassen den Titel, die Beschreibung (Description) sowie die Schlüsselwörter (Keywords), die zur eindeutigen Identifizierung dienen.

Diesem Schritt folgt die erneute Zusammenführung des gesamten Textes. Zur besseren Lesbarkeit durch den Benutzer werden jene Textabschnitte durch schwarze Balken ersetzt, die nicht mittels der vorhandenen Rechte eingesehen werden können. Die in dieser Session nicht entschlüsselbaren Parts wurden zwischenzeitlich im Hintergrund gesichert, um jetzt wieder in den Text integriert werden zu können. Um die Unveränderbarkeit der Teile selbst als auch deren Position innerhalb des Textes zu gewährleisten, wurden eigene Sicherungen programmiert, die eine permanente Überwachung eben dieser durchführen.

- *Parts mit Keys verschlüsseln*

  Nachdem das gesamte Dokument im Hintergrund zusammengestellt wurde, kann sich das System der Verschlüsselung widmen. Zunächst wird hierfür analysiert, welche (neuen) Access-Rights in das Dokument eingefügt wurden. Der Benutzer kann hierbei lediglich jene Access-Arten setzen, für die er auch die passenden AES-Schlüssel besitzt, die er mittels des ISAC-Zertifikats durch den Server übermittelt bekommen hat. Hat ein Client beispielsweise nur Community-Rechte, so kann er keine neuen Restricted-Access-Rechte innerhalb des Textes setzen.

  Nachdem die Analyse und die Zusammensetzung des gesamten Textes vollendet sind, werden alle neuen Access-Parts mit den entsprechenden AES-Schlüsseln aus dem ISAC-Zertifikat verschlüsselt und das Dokument nochmals mittels des Haupt-AES-Schlüssels gesichert. Danach kann die Datei zurück an den Server übermittelt werden.

- *File-Speicher-Request*

  Die Übermittlung der Datei geschieht mithilfe des Speicher-Requests an den Server. Hier wird das soeben verschlüsselte Dokument an den Server übermittelt, um eine Persistierung der neu erstellten Datei auf Serverseite anzustoßen. Die übermittelten Daten umfassen sowohl das Dokument selber als auch dessen, nur die vom Server entschlüsselbare, DocID und die allgemeinen Dokumentmetadaten (Title, Description, Keywords).

- *File speichern*

  Der eSprint-Server liest zunächst die empfangene DocID aus und entschlüsselt diese mithilfe seines internen Private Keys. Danach speichert er die übermittelten Nutzdaten in verschlüsselter Form in die Datenbank, von wo aus sie für andere Projektteilnehmer abrufbar sind.

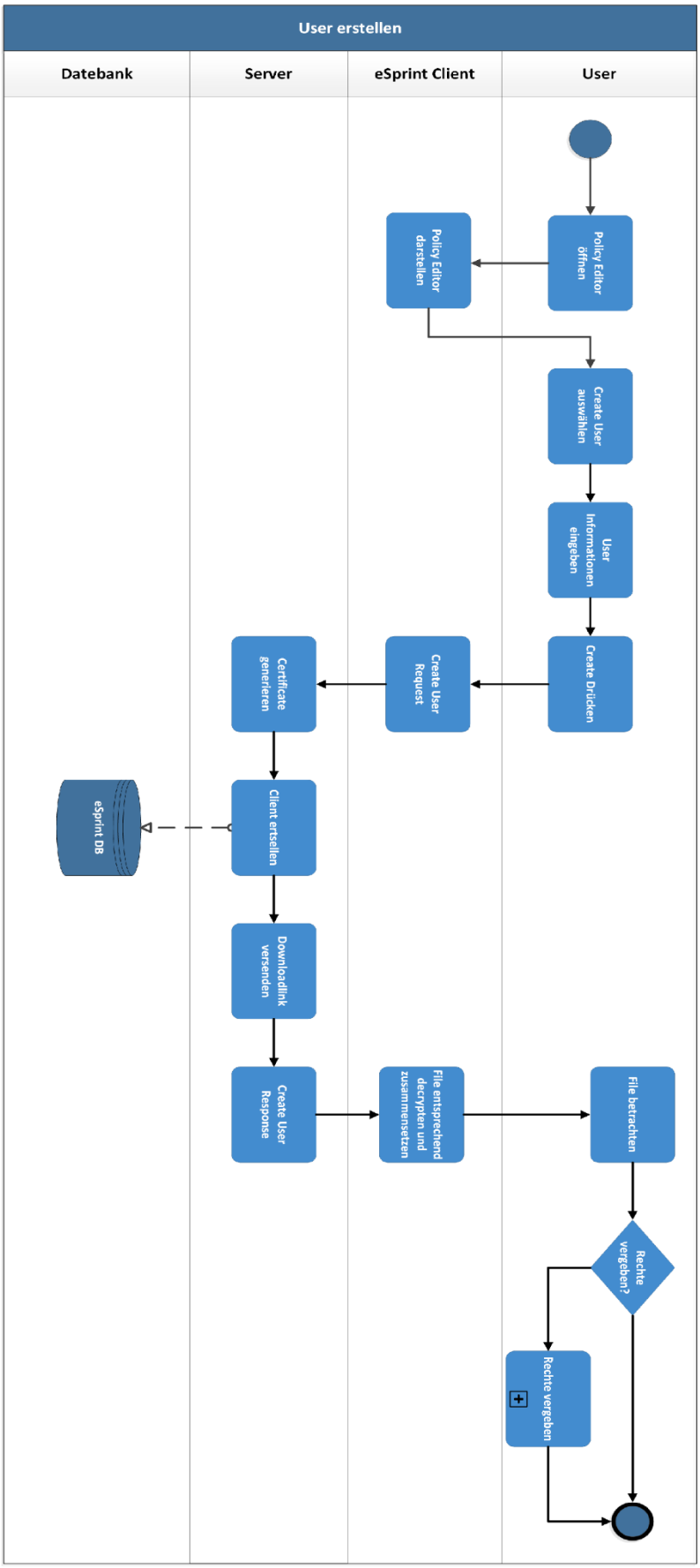

Abbildung 51: Ablaufdiagramm – User erstellen

9.4 User erstellen

Ebenfalls ein wichtiger Teilprozess ist die Erstellung eines neuen Benutzers. So lassen sich neue Mitarbeiter im System erzeugen und den jeweiligen Projekten zuordnen.

- *Policy-Editor öffnen*

  Zunächst muss der Benutzer den Policy-Editor öffnen. Dies geschieht direkt über den Text-Editor, in dem sich das dazugehörige Dokument befindet. Für das Öffnen des Policy-Editors ist ein vorheriges Speichern des Dokumentes am Server erforderlich und weiters muss der ausführende Benutzer der Ersteller beziehungsweise der Eigentümer (Owner) der geöffneten Datei sein. Hierdurch soll eine Veränderung der Zugriffsrechte durch andere Projektteilnehmer unterbunden werden.
- *Policy-Editor darstellen*

  Sollten die oben angeführten Voraussetzungen erfüllt sein, so kann der Editor durch den eSprint-Client angezeigt werden. Dies geschieht in Form eines neuen seitlichen Tabs unterhalb des dazugehörigen Datei-Tabs.
- *Create User auswählen*

  Nachdem sich das neue Fenster geöffnet hat, kann über den Punkt „Create User“ aus dem linksbündigen Menü die User-Erstellung geöffnet werden.
- *User-Information eingeben*

  In dem jetzt sichtbaren Eingabebereich müssen die notwendigen Informationen für den neuen Nutzer bereitgestellt werden. Diese umfassen den Vornamen, den Nachnamen, die E-Mail-Adresse sowie die Organisation beziehungsweise die Abteilung innerhalb des Unternehmens, für die der neue Nutzer tätig ist. Diese Informationen ermöglichen es den Dateierstellern, zu einem späteren Zeitpunkt die jeweiligen Benutzer leichter aufzufinden, um sie einem oder mehreren Projekten zuordnen zu können.
- *Create drücken*

  Nachdem alle benötigten Informationen angegeben wurden, kann der Vorgang der User-Erstellung mittels Drücken des Buttons „Create“ gestartet werden.

- *Create User-Request*

  Wird der Request zum Generieren eines neuen Benutzers abgesetzt, so werden zunächst auf Seiten des Clients die Informationen zur Übermittlung vorbereitet (Stichwort: Serialisierung), die angezeigte Eingabemaske ausgeblendet und auf eine Rückmeldung durch den Server gewartet. Die serialisierten Informationen werden in AES-verschlüsselter und signierter Form an den Server übermittelt, wo sie dann weiterverarbeitet werden können.

- *Certificate generieren*

  Am Server wird zunächst überprüft, ob sich in der Datenbank bereits ein Benutzer mit gleicher E-Mail-Adresse befindet. Ist dies der Fall, so muss kein neuer Client generiert, sondern lediglich ein Downloadlink zu einem bestehenden Client verschickt werden. Ist der Benutzer jedoch noch nicht vorhanden, folgt der nächste Schritt im Prozessdurchlauf. Bei diesem wird anhand der übermittelten Daten ein neuer Datenbankeintrag angelegt. Ebenfalls zu diesem Datensatz wird eine zufällig generierte Identifikationsnummer gespeichert, die dem Dateinamen des eSprint-Client-Zip-Archives entspricht, das vom Benutzer heruntergeladen werden kann. Nach diesem Schritt wird dieses Archiv samt allen benötigten Programmbestandteilen zusammengestellt. Hierfür wird ebenfalls ein neues Zertifikat mittels der übergeordneten CA (die sich am Server befindet) generiert. Dieser Schritt ist ähnlich der Erzeugung des ISAC-Zertifikates, jedoch wird hier das neue Zertifikat in einen (in Java üblichen) Keystore umgewandelt. Dieser wird vom Clients verwendet, um Verbindungen zu dem zugehörigen Server aufzubauen. Auch der Truststore wird im Archiv gespeichert. Dieser ist bei allen Clients desselben Servers ident und kann daher einmal generiert immer wieder verwendet werden.

- *Client erstellen*

  In dieser Phase erfolgt das Zusammenführen aller notwendigen Dateien in ein komprimiertes Archiv. Dieses muss deshalb komprimiert werden, weil der Client samt einer eigenständig lauffähigen Java-Umgebung ausgeliefert wird. Dies dient der plattform- und versionsunabhängigen garantierten Lauffähigkeit des

Programms, führt jedoch auch zu einem verhältnismäßig größeren Speicherbedarf.

- *Downloadlink versenden*

  Die bereits angesprochene, zufällig generierte ID des neu erzeugten Clients dient der eindeutigen Zuordnung zu einem Benutzer. Dieser erhält nach der Generierung des individualisierten Clients eine E-Mail vom Server, die einen Link zum Download von eben diesementhält. Ein Beispiel hierfür wird in Tabelle 7 angeführt.

From:	**eSprint Ltd**. <download@esprint.com>
Date:	2016-05-23 13:21 GMT+02:00
Subject:	eSprint Client
To:	max.mustermann@unet.univie.ac.at

Welcome to the eSprint Network.
Please download your Client here: *eSprint Download*

Tabelle 7: E-Mail mit Downloadlink zum Client

- *Create User-Response*

  Nachdem der neue Benutzer erfolgreich in der Datenbank angelegt und die zugehörige E-Mail versandt wurde, wird vom Server eine verschlüsselte Antwort über den positiven (oder auch negativen) Abschluss des Vorganges an den Ersteller des neuen Benutzers geschickt.

- *File entsprechend decrypten und zusammensetzen*

  Die Antwort des Servers wird auf Seiten des Clients nun entschlüsselt (Stichwort: Digital Envelope) und dem Benutzer über das Interface des Policy-Editors angezeigt.

- *File betrachten*

  Nachdem der neue Client generiert wurde, bekommt der Ersteller des neuen Benutzers eine positive Rückmeldung über die Anlegung der Daten. Diese wird im Policy-Editor mittels der Statusmeldung „User created successfully!“

ausgegeben. Unmittelbar danach kann bei Bedarf der nächste Benutzer angelegt werden.

- *Rechte vergeben*

  Abschließend kann der Benutzer an dieser Stelle in den Prozess „Rechte vergeben“ wechseln, um dort den eben erstellten Benutzer einem Projekt zuzuordnen und ihm seine individuellen Access-Rechte zu der Ressource zu erteilen.

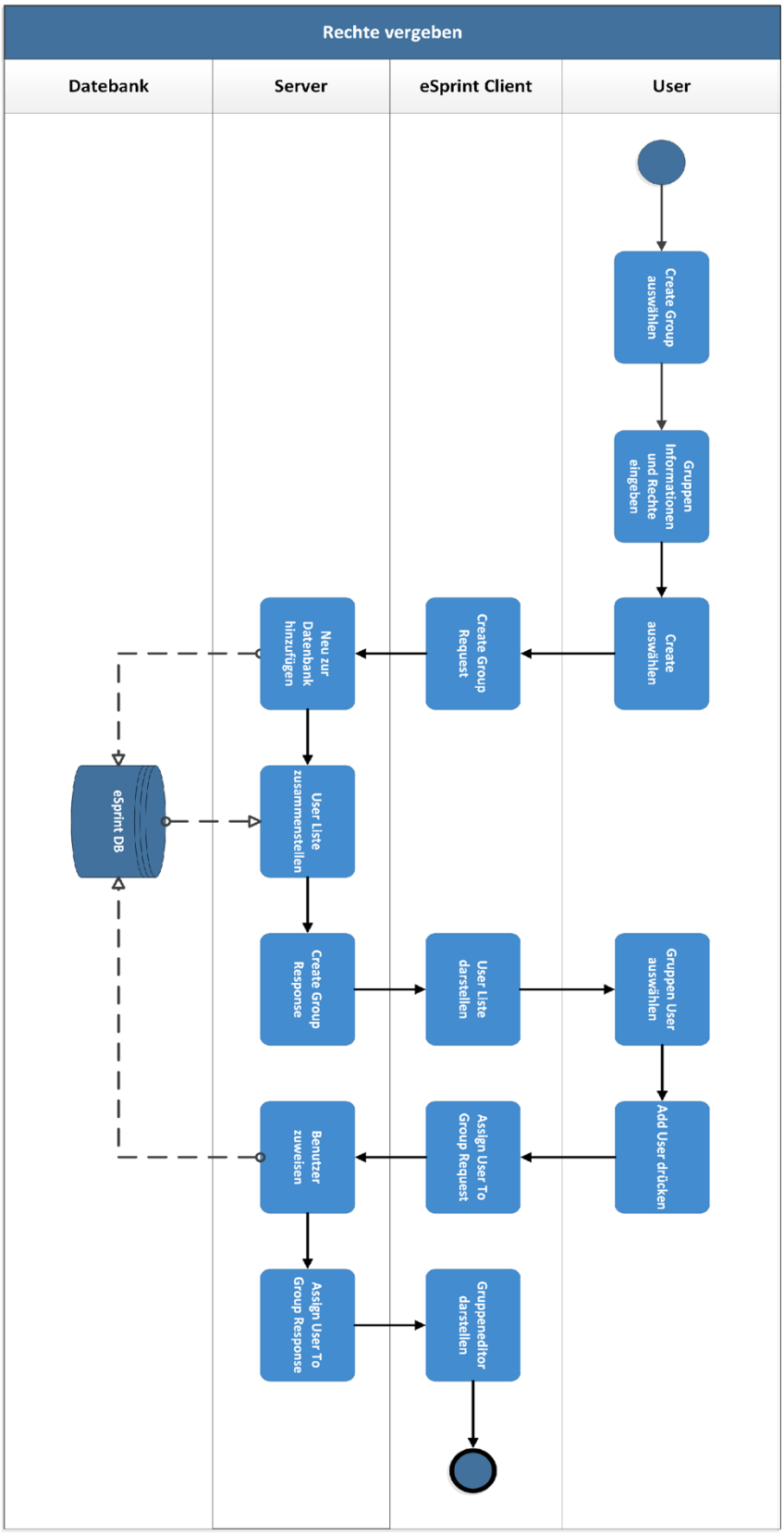

Abbildung 52: Ablaufdiagramm – Rechte vergeben

9.5 Rechte vergeben und verwalten

Der letzte relevante Hauptprozess des eSprint-Systems, den es zu erläutern gilt, umfasst die Vergabe und die Verwaltung von Benutzerrechten eines Projektes. Dieser Vorgang geschieht ebenfalls innerhalb des Policy-Editors und kann daher, wie bereits im Prozess 9.4 „User erstellen“, lediglich vom Owner einer Ressource initiiert werden.

- *Create Group auswählen*

  Im ersten Schritt des Prozesses wird aus dem linken Menü zunächst der Punkt „Create Group“ ausgewählt. Hierdurch wird im mittleren Anzeigebereich das Eingabeinterface generiert und angezeigt.
- *Gruppeninformationen und Rechte eingeben*

  Als Nächstes müssen durch den Benutzer alle relevanten Informationen eingegeben werden. Diese umfassen neben dem obligatorischen Gruppennamen auch dessen individuell kombinierbaren Access-Arten. Diese können aus jeder beliebigen Kombination der Zugriffsarten Open Access, Community Access, Organisation Access und Restricted Access bestehen.

  Weiters muss ein Gültigkeitsdatum für die jeweilige Gruppe definiert werden, worüber deren Zugriff zeitlich auf die Projektdauer begrenzt werden kann.

  Zuletzt kann noch festgelegt werden, ob die Mitglieder der neu erstellten Gruppe lediglich Leserechte an der Ressource besitzen sollen oder nicht.
- *Create auswählen*

  Nachdem alle Informationen durch den Benutzer eingegeben wurden, kann das Formular abgeschickt werden.
- *Create Group-Request*

  Nun wird die Anfrage samt allen angegebenen Daten zusammengestellt und zum Server übermittelt. Dies geschieht, wie jeder Datenaustausch zwischen dem Server und Client, mittels des Digital Envelopes über einen SSL-Tunnel.

- *Neu zur Datenbank hinzufügen*

  Auf Serverseite wird die Gruppe in der Datenbank angelegt. Sollte hierbei ein Fehler auftreten, so wird dies dem Client als Antwort zurückgegeben. Bei positiver Durchführung geht der Server zum nächsten Schritt über.

- *User-Liste zusammenstellen*

  Bei diesem Schritt wird vom Server eine User-Liste zusammengestellt, die alle in der Datenbank vorhandenen Benutzer umfasst. Die Informationen beinhalten – wie bereits in Kapitel 9.4 angegeben – den Vornamen, den Nachnamen, die E-Mail-Adresse sowie die Abteilung oder die Organisation.

- *Create Group-Response*

  Die oben ausgelesenen Informationen werden im Anschluss wiederum in Form von verschlüsselten Byte-Arrays an den Client zurückgesendet.

- *User-Liste darstellen*

  Nachdem der Client die Liste vom Server erhalten, entschlüsselt und umgewandelt hat, kann er sie dem Benutzer im Policy-Editor anzeigen. Die angezeigte Liste kann mithilfe eines Suchfeldes durch den Benutzer leicht und effizient eingeschränkt werden. Hierdurch wird es ermöglicht, selbst lange Listen von Benutzern zielführend zu filtern. Das Suchfeld untersucht durch eine Echtzeitanalyse die Eingabe des Benutzers und schränkt dadurch die Liste immer weiter ein. Hierbei spielt es keine Rolle, in welchem Feld sich die vom Benutzer eingegebenen Informationen befinden, da die Suche über alle Tabellenfelder läuft. Dies ermöglicht dem Benutzer beispielsweise, auch alle Benutzer derselben Abteilung schnell zu identifizieren und hinzufügen zu können.

- *Gruppen-User auswählen*

  Hat der Benutzer einen oder mehrere Benutzer gefiltert, die er dem aktuellen Projekt hinzufügen möchte, kann er in der rechten Spalte neben den jeweiligen Userinformationen mittels Anhaken des „Add“-Feldes diesen zum Hinzufügen markieren.

- *Add User drücken*

  Wurden alle Benutzer markiert, können diese durch Betätigung des Knopfes „Add Users“ hinzugefügt werden.

- *Assign User to Group-Request*
  Wie schon zuvor kann der Client nun eine Nachricht an den Server zusammenstellen und abschicken.
- *Benutzer zuweisen*
  Der Server empfängt die Anfrage des Clients und schreibt die neu zugeordneten Benutzer in die Datenbank.
- *Assign User to Group-Response*
  Nachdem die Benutzer einer Gruppe zugeordnet wurden, sendet der Server eine Antwort an den Client. Dieser zeigt das Resultat des Vorganges dem Benutzer an und leitet ihn automatisch zum Gruppeneditor weiter.
- *Gruppeneditor darstellen*
  Im Gruppeneditor kann der Benutzer alle der Ressource zugeordneten Gruppen einsehen und verwalten. Hierzu zählen wiederum die Änderung der Zugriffsrechte sowie das Hinzufügen oder Entfernen von Gruppenmitgliedern oder ganzen Gruppen.

10 Realisierung

Das Ziel dieses Kapitels ist es, mittels eines Top-Down-Ansatzes im Laufe der nächsten Kapitel die Funktionsweise des gesamten Systems genauer darzustellen und zu erläutern. Dabei soll dem Leser die Möglichkeit gegeben werden, sich bei Bedarf einen schnellen Überblick über die gesamte Infrastruktur zu verschaffen, aber bei Interesse auch einen detaillierteren Einblick zu bieten. Zu diesem Zweck werden ein Überblick über die vorhandenen Klassen der Server-Client-Struktur gegeben und ein Modell samt Beschreibung der dahinterstehenden Datenbank ausgeführt.

10.1 Klassendiagramm Überblick

In den nachfolgenden Ausführungen werden alle für die Realisierung des eSprint-Systems essentiellen Elemente und deren Funktion genauer erläutert.

10.1.1 Server

Der Server wurde als herkömmliches Java-Web-Service-Projekt innerhalb der Netbeans IDE erstellt. Als lauffähiger Server dient ein Apache Tomcat. Zusätzlich wurden benötigte „Jersey, JAX-RS“[17] Libraries für das Betreiben des angestrebten RESTful-WebServices eingebunden. Tabelle 8 gibt einen Überblick über die im Klassendiagramm in Abbildung 53 dargestellten Komponenten.

[17] Jersey, JAX-RS: Das Jersey RESTfull Framework ist ein Open-Source-Framework zur Entwicklung von RESTfull WebServices in Java.

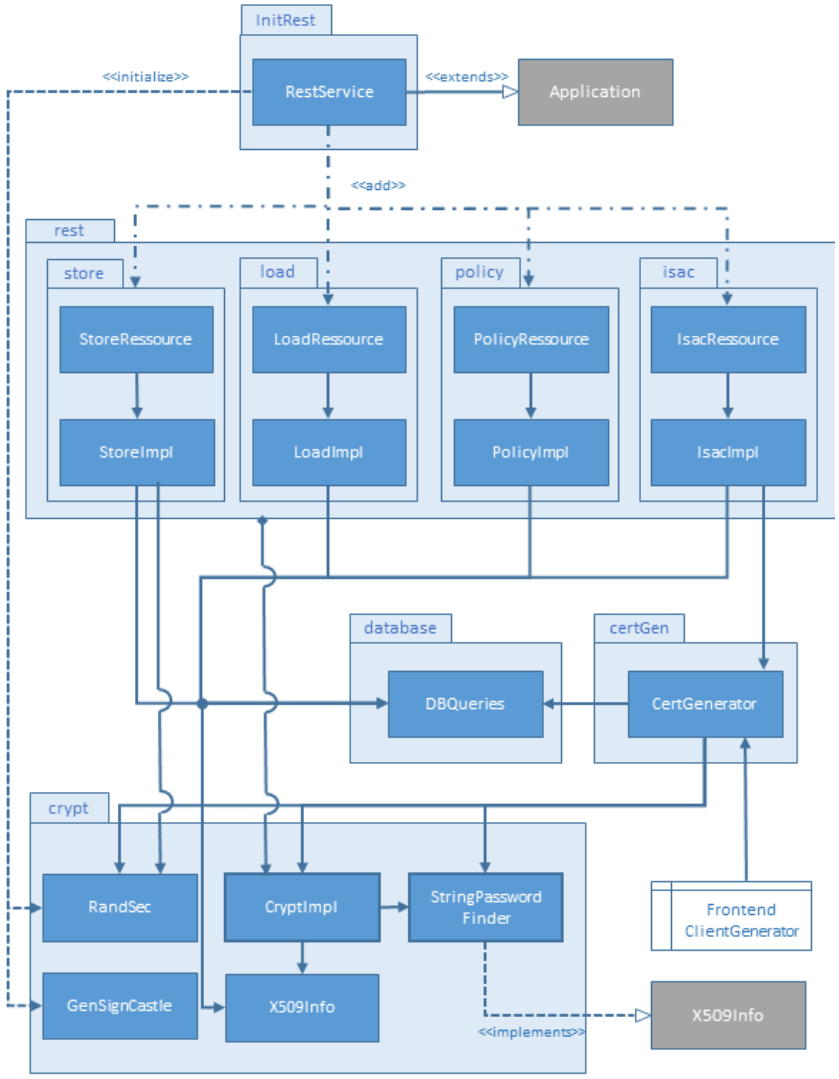

Abbildung 53: Klassendiagramm Übersicht – Server

Die implementierten Klassen wurden entsprechend ihrer Aufgaben in einzelne Packages zusammengefasst.

Package/Klasse	Eigenschaften
Webpages	Enthält alle benötigten JSP-, HTML-, CSS-Dateien für das Frontend. Das Frontend bietet dem Benutzer eine Schnittstelle zum Service, um neue Client-Anwendungen generieren und abrufen zu können.
initRest	Dient zur Initialisierung der RESTful-Webservices und wird bei Start der Applikation ausgeführt.
rest.load	Beinhaltet Registrierung und Implementierungen des Service-Endpunktes „/load“. Der Name des Endpunktes „Load“ impliziert, dass es sich hierbei um den Ansprechpunkt beim Laden einer bereits vorhandenen Datei handelt.
rest.policy	Umfasst Registrierung und Implementierungen des Service-Endpunktes „/policy“. Über den Endpunkt Policy lassen sich alle Rechtemanagement-Operationen durchführen.
rest.store	Beinhaltet Registrierung und Implementierungen des Service-Endpunktes „/store“. Über diese können die Speicheroperationen durchgeführt werden kann.
rest.isac	Umfasst Registrierung und Implementierungen des Service-Endpunktes „/isac“. Über den Endpunkt ISAC werden alle zertifikatsbezogenen Operationen durchgeführt. Dies umfasst fast jede Interaktion, die am Client durchgeführt wird und eine Kommunikation mit dem Server erfordert.
database	In diesem Package wurden alle datenbankbezogenen Operationen, also die Klassen/Funktionen zum Schreiben und Lesen auf die Datenbank zusammengefasst.

crypt	Hierin werden alle für die Kryptographie relevanten Klassen zusammengefasst. Die Klasse CryptImpl.java stellt hierbei alle Kryptographie-Funktionen für die angebotenen Services bereit. Der CertGenerator.java bietet die Funktion zum dynamischen Erstellen von neuen Clientanwendungen und deren Zertifizierung.
certGen	Enthält die Klasse CertGenerator.java, welche die entsprechenden Methoden zur Erstellung von Zertifikaten benötigt. Über die dort zur Verfügung gestellten Methoden können auch neue Clientanwendungen generiert werden.

Tabelle 8: Komponentenbeschreibung Implementierung Server

10.1.2 Client

Zur Erstellung des Clients wurde ein JavaFX-Projekt mittels der NetBeans IDE angelegt. Als ergänzendes Werkzeug wurde zur Generierung des Frontend-Grundgerüstes der in Kapitel 11.2 vorgestellte sogenannte JavaFX Scene Builder eingesetzt.

Zusätzlich wurden die zur Kommunikation mit dem RESTful WebService benötigten „Jersey, JAX-RS“ Libraries eingebunden.

Nachfolgend wird in Abbildung 54 ein Gesamtüberblick über alle vorhandenen Klassen und Packages des Clients gegeben.

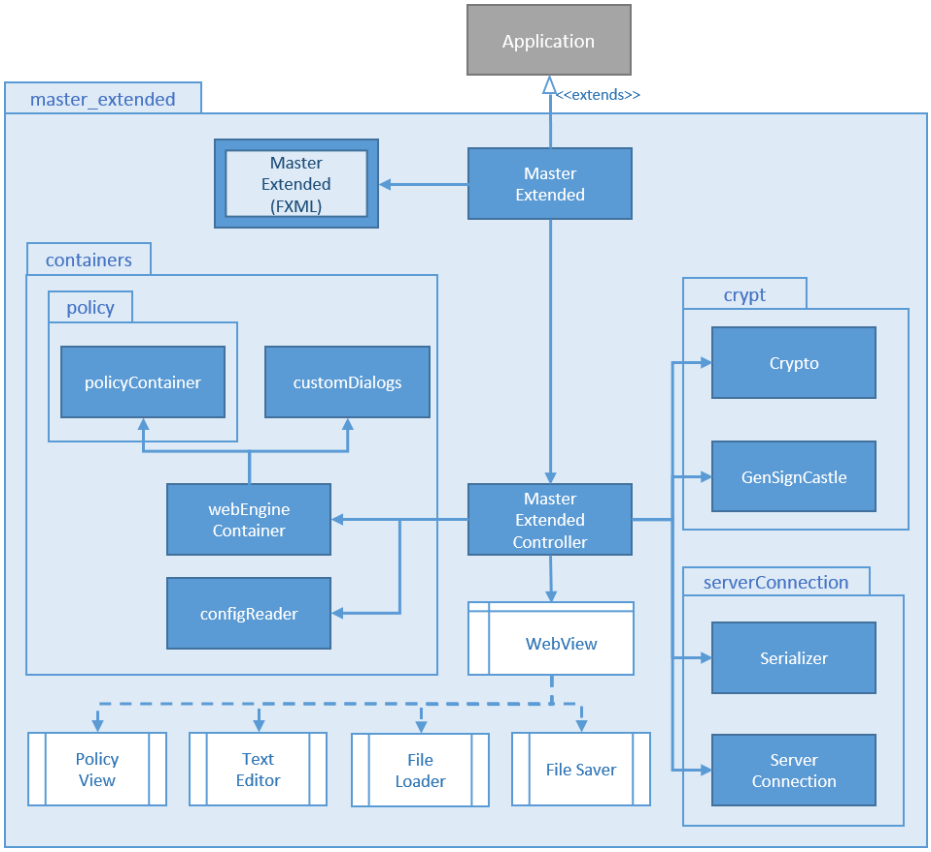

Abbildung 54: Klassendiagramm Übersicht – Client

Aus der obigen Darstellung lässt sich eine klare Trennung der einzelnen Umsetzungsmodule in „Steuerung“, „Anzeige“ und „Connection“ erkennen. Die wichtigsten Teile sollen in der nachfolgenden Tabelle 9 genauer erläutert werden.

Package/Klasse	Eigenschaften
Images	Dieses Package beinhaltet alle zur Darstellung des Clients relevanten Bilddateien. Hierzu zählen etwa das Anwendungsicon als auch diverse Hintergrundbilder.
Master_Extended.fxml	FXML-Dateien sind im Prinzip einfache XML-Dateien mit JavaFX-spezifischer Syntax. Diese Datei beinhaltet das Grundgerüst zur Darstellung des eSprint-Clients.
Master_Extended.java	Beinhaltet den zentralen Einstiegspunkt zur eSprint-Anwendung. Hier werden Basiseinstellungen zur Bereitstellung und Anzeige des Clients geschrieben.
Master_Extended Controller	Umfasst die zentralen Steuerelemente zur Interaktion mit der eSprint-Anwendung durch den Benutzer. Hierin wird die Interaktion der eigentlichen Businesslogik mit den Frontend-Elementen verwaltet.
CustomDialogs	Hat die Darstellungslogiken zur Anzeige von Dialog-fenstern innerhalb des eSprint-Clients zum Inhalt.
WebEngineContainer	Beinhaltet das jeweils durch den Benutzer ausgewählte, geöffnete Objekt (also das entschlüsselte und dargestellte Dokument) und alle dazugehörigen Elemente wie etwa die individuelle Live-URL der Datei.
PolicyContainer	Umfasst alle Informationen zur Darstellung des zu einem Dokument gehörenden Policy Editors. Über diesen kann der Benutzer Gruppenrechte und Benutzerzuordnungen verwalten.

Crypto	Hat die zur Zertifikatsanfrage-Generierung (CSR) notwendigen Implementierungen zum Inhalt. Hierüber werden neue ISAC-Zertifikate beim Server beantragt.
ServerConnection	Beinhaltet die Kommunikationslogik zur Interaktion mit dem RESTful-Webservice. Hierüber werden Verbindungen zum Server aufgebaut, um anschließend Lade- und Speichervorgänge oder einfache Abfragen durchführen zu können.

Tabelle 9: Komponentenbeschreibung Implementierung Client

10.2 Datenmodell

Im Zentrum des eSprint-Systems stehen die auf das jeweilige Dokument angewandten Policies. Nachfolgend soll das zugrunde liegende Datenmodell hierzu vorgestellt und erläutert werden.

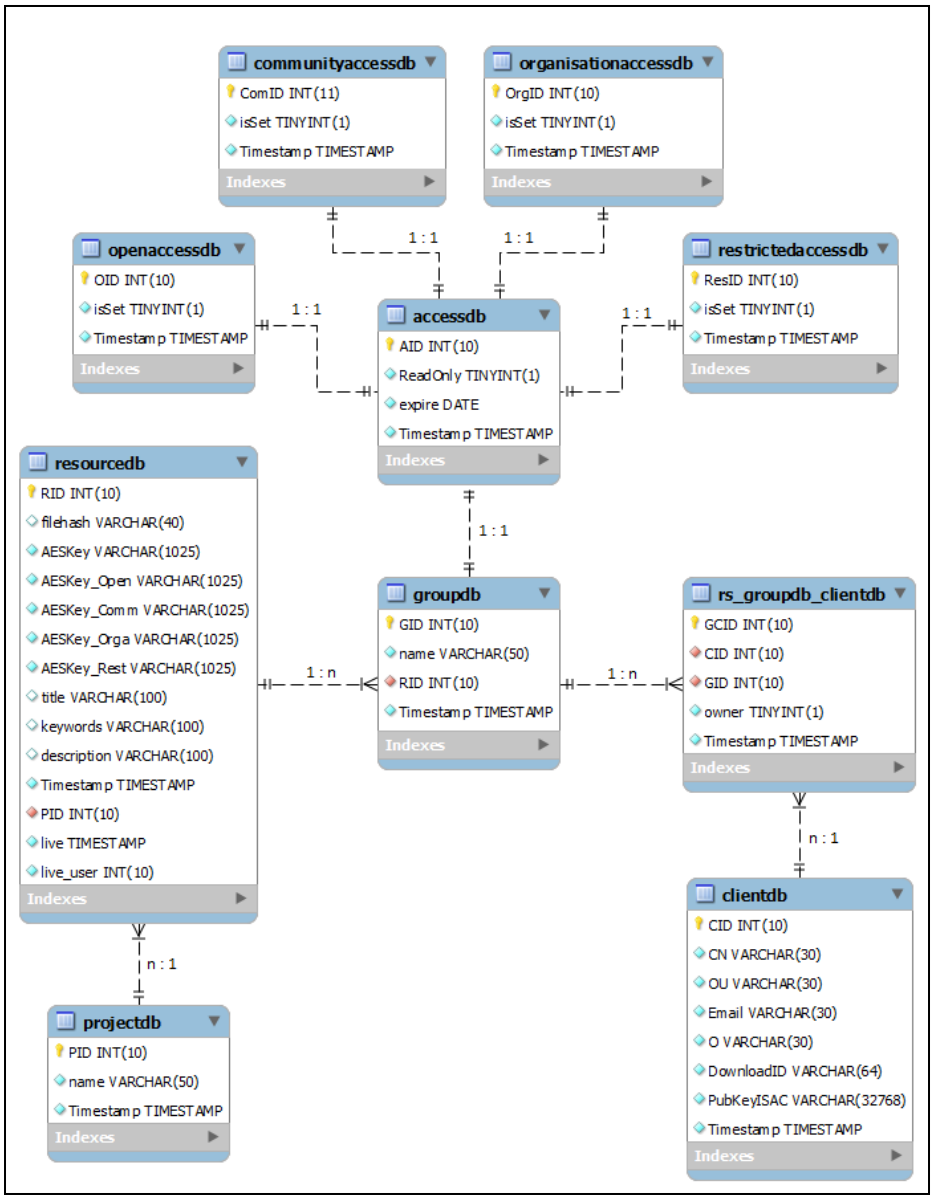

Abbildung 55: Datenbankmodell

Zunächst lässt sich eine grobe Unterteilung der wichtigsten Tabellen innerhalb der Datenbank vornehmen. Zu diesen zählen die ClientDB, die GroupDB, die ResourceDB, die ProjectDB sowie die AccessDB.

Wird zunächst die für die Darstellung der Benutzer unerlässliche ClientDB betrachtet, so ist festzustellen, dass diese die für die Identifikation notwendigen Felder den zugehörigen Client-Zertifikaten entnimmt. Hierzu gehören die Felder „CN“, „OU“, „Email“ und „O“. Diese stellen die Standardfelder eines herkömmlichen x.509-Zertifikates dar und stehen ursprünglich für „Common Name (CN)“, „Organizational Unit (OU)“, „Email“ und „Organization (O)“. Im Fall von eSprint wurden diese Felder zweckentfremdet, wobei CN den Vornamen, OU den Nachnamen, Email die E-Mail-Adresse und O die Abteilung innerhalb des Unternehmens enthalten. Außerdem enthält die Tabelle das Feld „DownloadID“, worüber eine eindeutige Zuordnung der generierten Clientanwendungen ermöglicht wird. Diese gewährleistet, dass ein Benutzer seine Anwendungen jederzeit erneut herunterladen und immer mit dem ihm zugeordneten Zertifikat und individuellen Zugriffsrechten weiterarbeiten kann. Zuletzt wird hier noch der verschlüsselte Public Key des dazugehörigen ISAC-Zertifikates gespeichert.

Die Steuerung der Zugriffsrechte erfolgt über Gruppen („GroupDB“). Diese werden durch ihren Primärschlüssel eindeutig identifizierbar und können für den Benutzer mittels des Feldes „name“ einfacher unterschieden werden. Clients werden diesen Gruppen zugeordnet und erhalten hierdurch die individuellen Rechte. Clients und Gruppen haben eine „n:m“-Kardinalität zueinander, jeder Client kann somit ein Teil mehrerer Gruppen sein und jede Gruppe kann mehrere Clients haben. Realisiert wurde dies über eine Tabelle, welche die Primärschlüssel aus beiden Tabellen enthält. Die „n:m“-Beziehung wurde hierdurch aufgelöst und über zwei „1:n“-Beziehungen mit der Tabelle „rs_groupdb_clientdb“ realisiert.

Die diversen Gruppen werden jeweils einer Ressource zugewiesen. Für jede Ressource besteht genau eine „Owner“-Gruppe, welcher der Ersteller der Ressource automatisch zugeordnet wird. Diese hält alle Rechte an der Ressource und kann somit auch neue Gruppen mit einer oder mehreren Zugriffsrechten auf die betreffende Ressource erstellen. Während eine Ressource von „n“ Gruppen bearbeitet werden kann, kann eine Gruppe jeweils nur einer Ressource zugeordnet sein. Es ergibt sich somit eine „n:1“-Beziehung zwischen diesen beiden Tabellen.

Die „resourcedb“-Tabelle enthält alle individuell relevanten Informationen wie etwa den Titel („title“), die Schlagwörter („Keywords“) und die Beschreibung

(„Description“). Darüber hinaus werden in dieser Tabelle die dateizugehörigen AES-Keys abgespeichert. Diese umfassen den allgemeinen AES-Key zum Ver- beziehungsweise Entschlüsseln der gesamten Datei und die einzelnen Keys der vier Zugriffsrechte. Außerdem enthält die Tabelle die Felder „live“ und „live_user“, die den gleichzeitigen Zugriff auf die jeweilige Datei beschränken und somit Dateikonfliktsituationen vermeiden.

Da es innerhalb des eSprint-Systems ebenfalls möglich ist, die generierten und vorhandenen Ressourcen jeweils einem Projekt zuzuordnen, wurde hierfür die Tabelle „projectdb“ angelegt. Diese hat neben einer eindeutigen ProjectID („PID“) einen Namen zur Identifizierung.

Für die Rechteverwaltung durch die jeweiligen Gruppen ist eine Gruppe stets einem eindeutigen Eintrag der „accessdb“-Tabelle zugeordnet. In dieser werden wiederum die Verknüpfungen zu den einzelnen Zugriffsrechte-Tabellen geführt. Neben den Fremdschlüsseln ist hier besonders das Feld „expire“ vom Datentyp „Date“ hervorzuheben. Dieses dient zur determinierten Steuerung des Ablaufdatums auf die Zugriffsrechte. Ist das vorgegebene Datum erreicht, wird der Gruppe durch einen automatischen Sicherungsmechanismus der Zugang zur Ressource verwehrt.

Die Zugriffsrechte umfassen die Tabellen „openaccessdb“, „communityaccessdb“, „organisationaccessdb“ und „restrictedaccessdb“, welche die Pendants zu den in dieser Arbeit definierten Access-Arten darstellen. Die Zugriffsrechte-Tabellen enthalten einerseits eine eindeutige ID als Primärschlüssel, der auch der Zuweisung zur „accessdb“-Tabelle dient, sowie einen Boolean-Wert, der anzeigt, ob diese Zugriffsart gesetzt wurde.

10.3 Schnittstellendefinition und –beschreibung

Beim Entwurf der Datenaustauschschnittstelle war es entscheidend, die Anforderungen einer sicheren Austauschmöglichkeit zu erfüllen und zugleich eine größtmögliche Flexibilität für die Art des auszutauschenden Objektes zu ermöglichen.

10.3.1 Datentransfer-Format

Für die Übermittlung der Daten wurde das Datenformat MIME Multipart gewählt. Dieses ist eine Erweiterung des Internetstandards RFC822 und spielt eine zentrale Rolle bei der Übermittlung von Dateien oder Nutzdaten über das HTTP-Protokoll [RFC822].

Im Falle dieser Arbeit wurde der spezielle Multipart-Subtyp mit der Bezeichnung „Form-Data“ gewählt. Dieser wurde ursprünglich zur Übermittlung von Formulardaten eingeführt, stellt aber mittlerweile den gebräuchlichsten Subtyp zur Übermittlung von Dateien dar. Aus semantischer Sicht wäre an dieser Stelle wohl der Subtyp „Mixed“ noch passender, jedoch gab es in der Vergangenheit immer wieder Systeme und Programmiersprachen, die mit deren Umsetzung Probleme hatten. Aus diesem Grund wurde der – semantisch nicht ganz korrekte, jedoch eher universell einsetzbare – Typ Form-Data gewählt [MBU14].

MultiPart-/Form-Data ermöglicht die Unterteilung der gesendeten Daten in mehrere, separat auslesbare, „Bodyparts“. Hierdurch lässt sich eine einfach handhabbare Trennung der übermittelten Daten in Meta- als auch Nutzdaten erzielen. Dies begründet, warum die Entscheidungsfindung zu diesem Übermittlungsformat geführt hat. Es ermöglicht eine wenig aufwendige und zugleich zweckdienliche Trennung der Steuerungs- und Nutzdaten. Hierzu zählt etwa die Signatur einer Nachricht oder der enkodierte „Header“, also die Metadaten, der Nachricht (Steuerung) als auch der tatsächliche „Content“ (Nutzdaten). Zusammengefasst können mittels MIME MultiPart-/Form-Data also einfach an die Anforderungen angepasste Datentransfers von Objekten vorgenommen werden.

10.3.2 Datenstruktur

An dieser Stelle wird die gewählte Datenstruktur zur Übermittlung der tatsächlichen Nutzdaten untersucht. Hierfür wurde eine möglichst einfache Strukturierung entworfen, die durch einen Standard abgebildet werden kann. Aus diesem Grund wurde eine XML-Darstellung gewählt, die danach als verschlüsseltes Byte-Array übertragen werden kann. In diesem XML werden zum einen separate Datei-Informationen gespeichert, wie etwa Titel, Beschreibung und Stichwörter, zum anderen die (HTML-)

Informationen, die sich aus dem Editor mit den eingegebenen Daten des Benutzers ergeben. Dies läuft folgendermaßen ab:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <head>
  <title>Testdokument 1</title>
  <keywords>Stichwort 1, Stichwort 2, Stichwort 3</keywords>
  <description>Dieses Dokument dient zu Testzwecken</description>
  </head>
  <content>
  <![CDATA[
  <html>
    <head>
     <title>Testdokument 1</title>
     <meta name="description" content="Dieses Dokument dient zu Testzwecken"/>
     <meta name="keywords" content="Stichwort 1, Stichwort 2, Stichwort 3" />
    </head>
    <body>
        Lorem ipsum dolor sit amet, consetetur sadipscing elitr.
    </body>
  </html>
  ]]>
  </content>
</document>
```

Abbildung 56: Datentransfer Content-Struktur

Diese Daten werden anschließend zu einem Byte-Array umgewandelt, mittels AES verschlüsselt und als Bodypart angehängt.

10.3.3 Service-Endpunkte und Ressourcen

Der Server verfügt über mehrere, für den Client zugängliche, Service-Endpunkte, die angesprochen werden können. Hinter diesen Endpunkten befinden sich mehrere Ressourcen, über die der Client Operationen am Server ansprechen kann. Für dieses Projekt wurde nach längerer Überlegung festgelegt, dass die Kommunikation zwischen dem Client und Server stets über die HTTP-Methode „Post“ stattfinden sollte. Dieser Überlegung geht der semantisch korrekte Gedanke voran, dass Serveranfragen, die Daten zur Verarbeitung mit sich führen, stets ein POST-Request sind, bevor die Antwort durch den Server erfolgen kann. Im Falle dieses Projektes müssen zu jeder

Anfrage Zertifikate, Schlüssel und die tatsächliche Anfrage übermittelt werden. Konkret handelt es sich in unserem Fall um vier ansprechbare Endpunkte, die über mehrere Ressourcen oder Objekte verfügen. Wie in Abbildung 57 zu erkennen ist, umfasst dies auf der Serverseite vier Endpunkte:

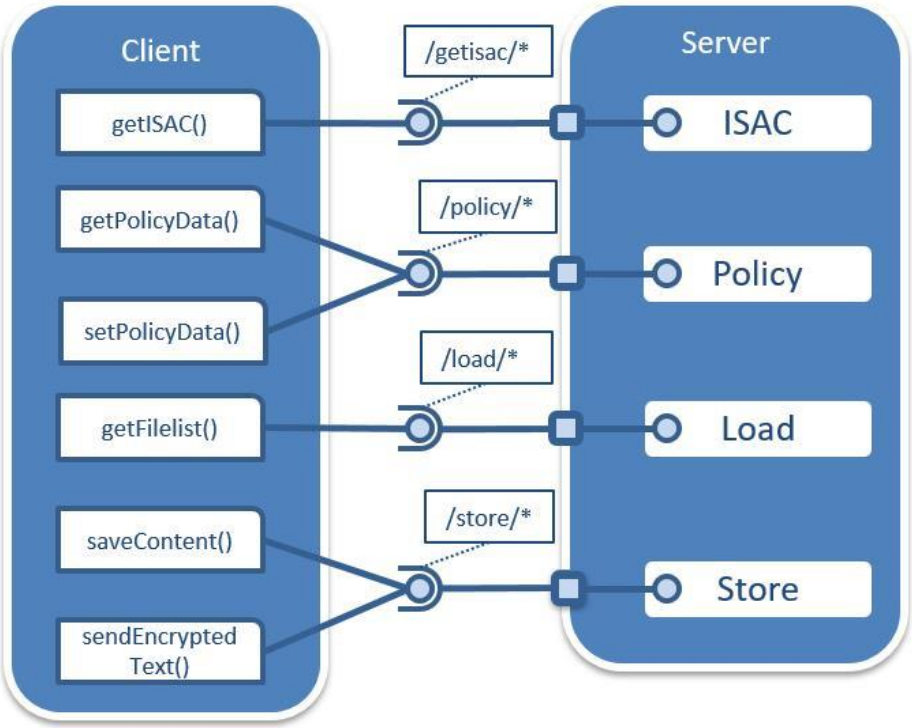

Abbildung 57: Schnittstellendefinition

ISAC

Über den Endpunkt ISAC werden alle zertifikatsbezogenen Operationen durchgeführt. Dies umfasst fast jede Interaktion, die am Client durchgeführt wird und eine Kommunikation mit dem Server erfordert. Hierzu zählen beispielsweise das Laden und Speichern von Dateien und Projekten. Aber auch die Bearbeitung der gesetzten Policies, also das Setzen von Gruppenberechtigungen, Hinzufügen von neuen Mitarbeitern zu einem Projekt oder das Löschen von vorhandenen Gruppen und/oder deren Mitgliedern, beherbergt das Ansprechen dieses Endpunktes. Konkret umfasst dieser Endpunkt folgende Ressourcen:

- */save :* POST<MultiPart>

  Um ein Dokument speichern zu können, benötigt ein Client ein ISAC-Zertifikat. Dieses fordert der Client von der Ressource getisac/save des Servers an. Er übermittelt die Identifikationsnummer des zu speichernden Dokuments, einen Server Public Key verschlüsselten AES-Key und ein damit verschlüsseltes Certificate-Signing-Request- (.csr) Dokument, das Informationen für die ISAC-Zertifikatserstellung enthält. Der Server befüllt das ISAC-Zertifikat entsprechend der Client-Rechte mit den AES-Keys der einzelnen Access-Arten, signiert es und sendet es zurück an den Client.

getisac/save : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	encryptedAesKey	AES-256	PublicKey verschlüsselter AES Key
Input	encryptedcsrbytes	Certificate Signing Request	ISAC Zertifikat Request
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	isac	ISAC Certificate	Befülltes ISAC Zertifikat

Tabelle 10: Schnittstellendefinition getisac/save

- */load :* POST<MultiPart>

  Auch für den Ladeprozess benötigt der Client ein ISAC-Zertifikat mit den zum Dokument passenden Schlüsseln, um dieses danach auch entsprechend entschlüsseln zu können. Hierzu bedient er sich der Server-Ressource getisac/load. Der Client übermittelt die zu ladende Identifikationsnummer des Dokuments mit einem verschlüsselten CSR und bekommt ein signiertes ISAC-Zertifikat zurück.

getisac/load : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	encryptedAesKey	AES-256	PublicKey verschlüsselter AES Key
Input	encryptedcsrbytes	Certificate Signing Request	ISAC Zertifikat Request
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	isac	ISAC Certificate	Befülltes ISAC Zertifikat

Tabelle 11: Schnittstellendefinition getisac/load

Policy

Über die Endpunkt-Policy lassen sich alle Rechtemanagement-Operationen durchführen. Hierzu zählen die Erstellung von neuen Gruppen innerhalb eines Projektes, die Festlegung der jeweiligen Gruppenberechtigungen, inklusive eines Gültigkeitszeitraumes der Zugriffsrechte für diese Gruppe, die Zuordnung neuer Gruppenmitglieder, die Anzeige der eigenen Berechtigung und nicht zuletzt das Anlegen neuer Unternehmensmitarbeiter, inklusive eines automatischen Mail-Versandes mit einem personalisierten Client für den neuen Projektmitarbeiter. Um die Möglichkeiten des Policy-Managements nutzen zu können, muss zuvor ein vollständiger Speichervorgang durchgeführt werden. Ist dieser erfolgt, stehen die folgenden Ressourcen zur Verfügung:

- */getgroupname :* POST<MultiPart>

  Die Ressource policy/getgroupname bekommt die ID eines Dokuments, ermittelt den dazugehörigen Gruppennamen anhand des aufrufenden Users und retourniert diesen an den Client.

policy/getgroupname : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	gid	Integer ID der Gruppe	Identifikationsnummer der Gruppe

Tabelle 12: Schnittstellendefinition getisac/getgroupname

- */getgroups :* POST<MultiPart>

  Über die Ressource policy/getgroups können alle einem Dokument zugeordneten Gruppen angefordert werden. Der Client übermittelt die Identifikationsnummer des Dokuments und der Server antwortet mit den dazu passenden Gruppen.

policy/getgroups : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	gids	Integer IDs der Gruppen	Serialisierte Identifikationsnummern der Gruppen

Tabelle 13: Schnittstellendefinition getisac/getgroups

- */getgroupinfo :* POST<MultiPart>

  Die Ressource policy/getgroupsinfo ermittelt Zugriffs-Informationen für die Gruppen eines Dokuments. Über die gesendete Dokumentidentifikationsnummer werden für jede dazu passende Gruppe der Name, das Ablaufdatum, die Zugriffsrechte und die Schreibberechtigung ermittelt.

policy/getgroupsinfo : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	groupsinfo	Integer IDs und Namen der Gruppen	Serialisierte Identifikationsnummern und Namen der Gruppen

Tabelle 14: Schnittstellendefinition getisac/getgroupsinfo

- */getownrights :* POST<MultiPart

  Über policy/getownrights fordert der Client seine Rechte auf ein Dokument an. Der Server identifiziert den Client über sein SSL-Zertifikat und sendet ihm Informationen, über welche Access-Arten er für das abgefragte Dokument verfügt.

policy/getownrights : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	rightssinfo	String mit Gruppenrechten	Serialisierte Information über Gruppenrechte

Tabelle 15: Schnittstellendefinition getisac/getownrights

- */creategroup :* POST<MultiPart>

  Die Ressource policy/creategroup dient zur Anlage neuer Gruppen. Übermittelt wird die Identifikationsnummer des betreffenden Dokumentes sowie serialisierte Information über den Namen und die Gruppenrechte der zu erstellenden Gruppe. Der Server überprüft die Rechte des Clients, erstellt die neue Gruppe und retourniert deren Gruppen-Identifikationsnummer.

policy/creategroup: POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	rightssinfo	String mit Gruppenrechten	Serialisierte Information über die zu erstellende Gruppe
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	gid	Integer ID der Gruppe	Identifikationsnummer der erstellten Gruppe

Tabelle 16: Schnittstellendefinition getisac/creategroup

- */getusers :* POST<MultiPart>

  Alle verfügbaren User können über policy/getusers angefordert werden. Der Server befragt die Datenbank und übermittelt eine Liste aller zuweisbaren Benutzer in serialisierter Form an den Client.

policy/getusers : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	users	String mit Userliste	Serialisierte Information über alle Benutzer

Tabelle 17: Schnittstellendefinition getisac/getusers

- */assignuserstogroup :* POST<MultiPart>

  Der Client fordert über die Ressource policy/assignuserstogroup die Zuweisung eines Users zu einer Gruppe an. Für diese Anfrage übermittelt der Client eine Liste an Benutzeridentifikationsnummern sowie die Gruppenidentifikationsnummer. Die Benutzer werden am Server dieser Gruppe zugewiesen. Abschließend wird die erfolgreiche Durchführung über den Parameter „confirm“ zurückgemeldet.

policy/assignuserstogroup: POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	gid	Integer ID der Gruppe	Identifikationsnummer der zuzuweisenden Gruppe
Input	cids	Integer IDs von Clients	Identifikationsnummer der zuzuweisenden Benutzer
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	confirm	Boolean Bestätigung	Information über die erfolgreiche Verarbeitung

Tabelle 18: Schnittstellendefinition getisac/assignuserstogroup

- */createusers :* POST<MultiPart>

  Der Client fordert über die Ressource policy/createusers die Erzeugung neuer Benutzer an. Dier Client überträgt Informationen über die anzulegenden Benutzer an den Server. Der Server kreiert die notwendigen Zertifikate und übermittelt diese gemeinsam mit neu erzeugten Clients an die hinzugefügten Benutzer.

policy/createusers : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	users	String mit User Informationen	Serialisierte Informationen über die anzulegenden Benutzer
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	confirm	Boolean Bestätigung	Information über die erfolgreiche Verarbeitung

Tabelle 19: Schnittstellendefinition getisac/createusers

- */getgroupmembers :* POST<MultiPart>

  Über die Ressource policy/getgroupmembers können Detail-Informationen zu Gruppenmitgliedern abgerufen werden. Für die übermittelte Identifikationsnummer des Dokuments ermittelt der Server alle Gruppennamen und ihre Benutzer.

policy/getgroupmembers : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	members	String mit Gruppenmitglieder-Informationen	Gruppennamen und ihre Benutzer

Tabelle 20: Schnittstellendefinition getisac/getgroupmembers

- */getowner :* POST<MultiPart>

  Die Ressource policy/getowner gibt Auskunft über den Besitzer eines Dokuments. Der Server gibt wertvolle Informationen wie E-Mail und Name über den Besitzer eines Dokuments zurück. Mithilfe dieser Informationen kann ein Benutzer dann weitere Rechte an einem Dokument beantragen.

policy/getowner : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	ownerinfo	String mit Client-Informationen	Informationen über den Besitzer eines Dokuments

Tabelle 21: Schnittstellendefinition getisac/getowner

Load

Der Name des Endpunktes „Load“ lässt erahnen, dass es sich hierbei um den Ansprechpunkt beim Laden einer bereits vorhandenen Datei handelt. Er beinhaltet auch eine gleichheißende Ressource „*load*“, über die eine Datei vom Client angefordert werden kann. Hierbei wurde ein Sicherungsmechanismus eingebaut, sodass eine Datei immer nur von einer Person gleichzeitig genutzt und bearbeitet werden darf. Dieser Zeitraum ist – für den Sonderfall, dass ein Benutzer das Programm unvorhergesehen beenden muss, wie es etwa bei einem Computerabsturz der Fall wäre – auf fünf Minuten beschränkt. Erfolgen innerhalb dieser fünf Minuten normale Interaktionen des Benutzers, verlängert sich der Zeitraum wieder um 5 Minuten (und so weiter). Diese Vorkehrung mit dem Namen „Single Session Security“ (kurz: S3) garantiert eine einfache Implementierung von konsistenter Datenhaltung und ist somit Teil des entwickelten Continous-Access-Control-Mechanismus. Ebenfalls Teil dieser S3-Funktion ist die Ressource „*sendKill*“. Dieser Name stammt von Linux und bedeutet, dass der Prozess beendet werden soll. In Falle von eSprint heißt es also, dass der Prozess der Dateibearbeitung beendet ist und somit ein anderer Benutzer bei Bedarf Zugriff darauf hat.

Die Ressourcen dieses Endpunktes umfassen die Möglichkeit des Ladens einer Datei. Um dies durchführen zu können, muss dem Benutzer zunächst eine Auswahl aller vorhandenen Dateien, auf die er Zugriffsrechte hat, gegeben werden. Dies geschieht über die Ressource des „*filechoosers*“.

- */load :* POST<MultiPart>

  Über die Ressource /load kann ein Dokument entsprechend der Rechte des Anforderers geladen werden. Hierzu übermittelt der Client die Identifikationsnummer des zu ladenden Dokuments an den Server. Dieser verschlüsselt die einzelnen Dokumentteile sowie das Dokument als Ganzes und übermittelt es an den Client. Der Client bekommt die dazu passenden Schlüssel über sein ISAC-Zertifikat.

/load : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	encryptedRid	AES-256	Verschlüsselte Dokumenten-identifikationsnummer
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	document	Byte Array	Verschlüsseltes Dokument
Output	doucmentInfo	String	Weitere Dokumentinformationen

Tabelle 22: Schnittstellendefinition /load

- */filechooser :* POST<MultiPart>

  Die Ressource filechooser dient dazu, alle relevanten und verfügbaren Informationen für eine Dokumentladeauswahl dem Client zur Verfügung zu stellen. Dies inkludiert Projekt und Dokumentinformationen.

load/filechooser : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	clientInformation	String	Informationen über den aufrufenden Client
Output	signature	SHA512withRSA Signatur	Signatur des Servers
Output	projectDocInfo	String	Projekt und Dokumentinformationen

Tabelle 23: Schnittstellendefinition load/filechooser

- */keepalive :* POST<MultiPart>

  Über diese Ressource teilt der Client dem Server mit, dass er ein bestimmtes Dokument gerade in Verwendung hat. Diese Information wird vom Server benötigt, um sicherstellen zu können, dass immer nur ein Nutzer ein Dokument bearbeitet. Der Server sendet keine weiteren Daten an den Client.

load/keepalive : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments

Tabelle 24: Schnittstellendefinition load/keepalive

- */sendkill :* POST<MultiPart>

  Mit /sendkill teilt der Client dem Server mit, dass er ein Dokument nicht mehr in Verwendung hat. Somit ist es wieder für andere Benutzer zum Bearbeiten verfügbar. Der Server sendet keine weiteren Daten an den Client.

/load/sendkill : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	signature	SHA512withRSA Signatur	Signatur des Clients
Input	rid	Integer ID der Ressource	Identifikationsnummer des Dokuments

Tabelle 25: Schnittstellendefinition load/sendkill

Store

Zuletzt wird das Speichern bestehender und neuer Dateien betrachtet. Der hierfür zuständige Endpunkt ist unter der Bezeichnung „Store“ zu finden und beinhaltet wiederum mehrere Ressourcen. Hierzu zählt die bereits angesprochene Möglichkeit, neue und durch den Benutzer erstellte Dateien anzulegen oder aber zuvor geladene und bearbeitete bestehende Dateien wieder zu speichern. Hier greift der zuvor beschriebene Single-Session-Security-Mechanismus zur Gewährleistung der Konsistenz.

Der Endpunkt „Store“ beinhaltet wiederum die gleichnamige Ressource „store“, worüber die Speicheroperation durchgeführt werden kann. Außerdem gibt es noch die

Ressource „projects“, die dem Benutzer beim Speichervorgang die Möglichkeit bietet, das neu erstellte File einem bereits bestehenden Projekt zuzuordnen. Über diese Ressource werden demnach alle im System vorhandenen Projekte zurückgegeben, auf die der Benutzer aktuell Zugriff hat und somit auch als Speicherort nutzen kann.

- */store :* POST<MultiPart>

  Der Client übermittelt Meta-Informationen zu einem Dokument und das entsprechend der ISAC-Informationen verschlüsselte Dokument an den Server-Service-Endpunkt „/Store“. Der Server speichert das Dokument und bestätigt dies über den Parameter „confirm“.

/store : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	Signature	SHA512withRSA Signatur	Signatur des Clients
Input	Rid	Integer ID der Ressource	Identifikationsnummer des Dokuments
Input	documentInfo	String	Weitere Dokumentinformationen
Input	Document	Byte Array	Verschlüsseltes Dokument
Output	Confirm	Boolean Bestätigung	Information über die erfolgreiche Verarbeitung

Tabelle 26: Schnittstellendefinition /store

- *store/projects :* POST<MultiPart>

  Während des Speicherprozesses benötigt der Client Projektdaten, die er über die Ressource store/projects anfordern kann. Hierfür übermittelt der Client einfach Informationen über den aufrufenden Client.

store/projects : POST<MultiPart>			
Typ	**Name**	**Details**	**Beschreibung**
Input	Signature	SHA512withRSA Signatur	Signatur des Clients
Input	clientInformation	String	Informationen über den aufrufenden Client
Output	Signature	SHA512withRSA Signatur	Signatur des Servers
Output	projectInfo	String	Projekt und Dokumentinformationen

Tabelle 27: Schnittstellendefinition store/projects

11 Implementierungsbeschreibung

Dieses Kapitel befasst sich mit der Implementierung der zuvor beschriebenen Ideen und Techniken. Es werden – getrennt in Server- und Client-Seite – verschiedenste Codefragmente und deren Rolle im Gesamtprojekt vorgestellt. Dabei ist das Ziel, einen genaueren Einblick in die technische Funktionsweise von eSprint zu gewähren und durch Beispiele und Erklärungen den Sinn der gezeigten Codebeispiele (auch: *Snippets*) zu erläutern. Die Darstellungsform der Snippets kann variieren, da es hierbei das vornehmliche Ziel war, eine höchstmögliche Lesbarkeit zu gewährleisten.

11.1 Entwicklungsumgebung

Zunächst wird die eingesetzte Entwicklungsumgebung beschrieben. Hierbei haben sich die Autoren der Arbeit für die IDE NetBeans der Firma Oracle entschieden. Konkret musste vor der Umsetzung des eSprint-Projektes eine IDE gefunden werden, die den nachfolgenden Vorgaben entspricht:

- einfache Handhabung
- Ausrichtung auf Java-Projekte
- gute Integration von Web-Service-Architekturen
- Unterstützung von RESTful Webservices
- Kompatibilität mit Tomcat Webservern
- einfache Integration von MySQL-Datenbanken

Da Oracle zugleich die Entwicklerfirma der Programmiersprache Java ist, stellt die NetBeans IDE das empfohlene Standardtool zur Java-Entwicklung dar. Dementsprechend war eine nahtlose und unkomplizierte Handhabung bei der Erstellung eines umfangreichen Java-Projektes anzunehmen, was sich zu einem späteren Zeitpunkt auch bewahrheitet hat [NET18].

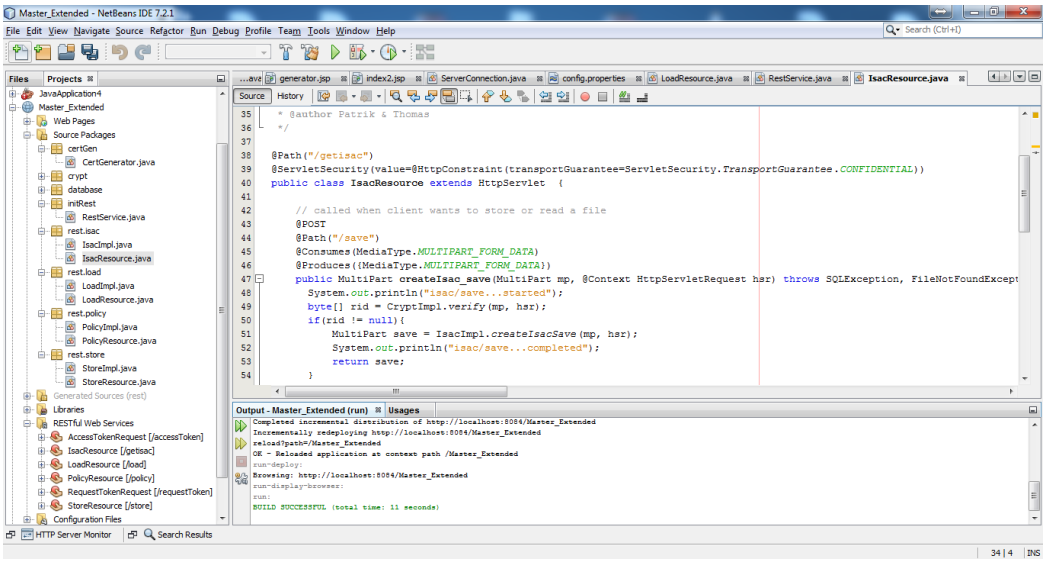

Abbildung 58: NetBeans IDE

Zusammengefasst erfüllte diese IDE zum damaligen Zeitpunkt alle Anforderungen und stellte somit die geeignetste Grundlage für die Umsetzung des eSprint-Systems dar.

11.2 Implementierung des Layouts

Ein weiteres Werkzeug zur Realisierung von eSprint war der JavaFX Scene Builder. Damals noch von Oracle selbst herausgegeben, ermöglichte er die einfache Generierung einer JavaFX FXML-Datei, basierend auf einer grafischen Oberfläche. Hierdurch ließen sich schnell und übersichtlich individuelle Programmoberflächen erstellen und/oder bestehende FXML-Dateien ändern [CD11].

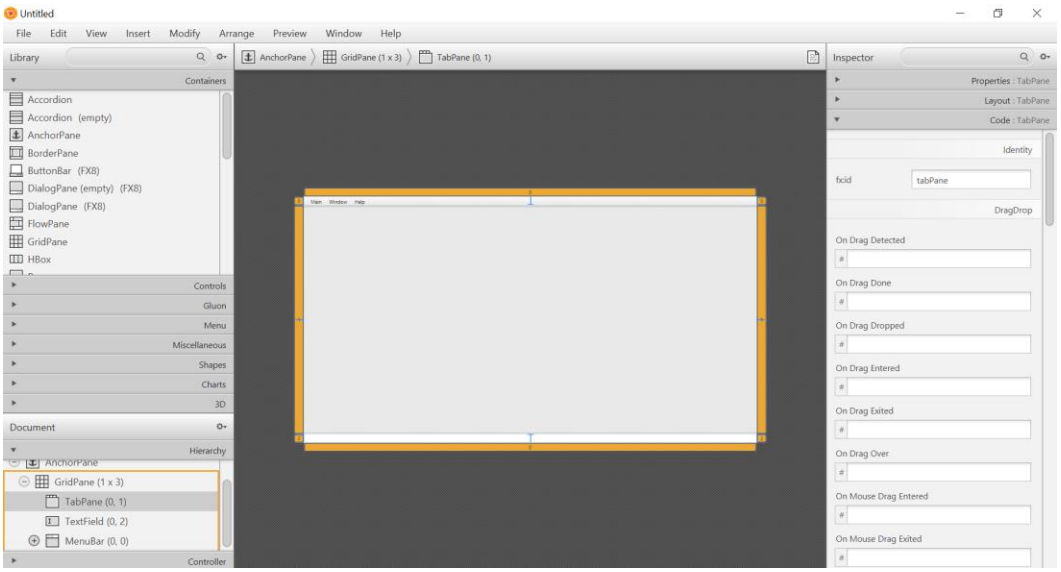

Abbildung 59: JavaFX Scene Builder

Mittels dieses Werkzeuges wurde die in Abbildung 60 dargestellte FXML-Datei erzeugt, durch programmatische Erweiterungen in Java-Steuerklassen noch feingranularer definiert und mit „Business“-Logik hinterlegt.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<!-- //… -->
<AnchorPane fx:id="AnchorPane" prefHeight="600.0" prefWidth="1100.0"
  xmlns:fx="http://javafx.com/fxml" fx:controller="master_extended.Master_ExtendedController">
    <children>
        <GridPane fx:id="gridPane1" alignment="CENTER" prefHeight="400.0"
          prefWidth="1025.0" styleClass="scene" AnchorPane.bottomAnchor="0.0"
          AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
            <children>
                <TabPane fx:id="tabPane" tabClosingPolicy="UNAVAILABLE"
                  prefHeight="379.0" prefWidth="1025.0" side="LEFT" GridPane.columnIndex="0"
                  GridPane.rowIndex="1" />
                <TextField fx:id="statusBar" editable="false" focusTraversable="false"
                  prefWidth="200.0" GridPane.columnIndex="0" GridPane.rowIndex="2"
                  styleClass="statusBar" />
                <MenuBar GridPane.columnIndex="0" GridPane.rowIndex="0">
                    <menus>
                        <Menu mnemonicParsing="false" text="Main">
                            <items>
                              <MenuItem mnemonicParsing="false" text="Close" />
                            </items>
                        </Menu>
                        <Menu mnemonicParsing="false" text="Window">
                            <items>
                              <MenuItem mnemonicParsing="false" onAction="#addEditorView"
                                text="Add editor" />
                              <MenuItem mnemonicParsing="false" onAction="#addPolicyView"
                                text="Add policy view" />
                            </items>
                        </Menu>
                        <Menu mnemonicParsing="false" text="Help">
                            <items>
                              <MenuItem mnemonicParsing="false" onAction="#debugTabs" text="About"/>
                            </items>
                        </Menu>
                    </menus>
                </MenuBar>
            </children>
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="1025.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints maxHeight="21.0" minHeight="21.0" prefHeight="21.0" valignment="TOP"
                  vgrow="SOMETIMES" />
                <RowConstraints vgrow="SOMETIMES" />
                <RowConstraints fillHeight="false" maxHeight="21.0" minHeight="21.0"
                  prefHeight="21.0" valignment="BOTTOM" vgrow="NEVER" />
            </rowConstraints>
        </GridPane></children></AnchorPane>
```

Abbildung 60: Java FXML-Benutzeroberfläche (Frontend)

Nachdem die relevanten Elemente der benötigten Entwicklungsumgebung vorgestellt wurden, kann die eigentliche Implementierung stattfinden. Hierfür wird im anschließenden Kapitel anhand einer exemplarischen Aktivität ein kurzes Codebeispiel der Umsetzung gezeigt.

11.3 Implementierung der Zertifikatsgenerierung

In diesem Abschnitt werden kurze Codeteile von Client und Server präsentiert. Hierzu wird die zugrunde liegende Implementierung der Aktivitäten *ISAC Request absetzen (.csr)* und *ISAC Zertifikat für User/File kreieren* des in Kapitel 9.2 Datei laden beschriebenen Prozesses gezeigt und erläutert. Bei diesen Code-Ausschnitten handelt es sich lediglich um die wesentlichen Teile der gezeigten Methoden, weshalb zur leichteren Lesbarkeit manche Programmcodes weggelassen wurden. Der vollständig erhaltene Programmcode kann im angeschlossenen Annex dieser Arbeit nachgeschlagen werden.

Hierbei wird die Aktivität *ISAC Request absetzen (.csr)* vom Client und die Aktivität *ISAC Zertifikat für User/File kreieren* vom Server ausgeführt.

11.3.1 Client – ISAC Request absetzen (.csr)

Im Kapitel 9.2 „Datei laden“ können die zuvor durchgeführten Schritte nochmals im Detail betrachtet werden. An dieser Stelle wird kurz die Ausgangslage vor der Ausführung des gezeigten Codes angeführt.

Der Benutzer sieht innerhalb seiner Clientanwendung eine vom Server erhaltene File-Liste, welche die Identifikationsnummern der zu ladenden Dokumente enthält. Dort wählt er nun die Datei, die er über den eSprint-Client laden möchte.

Nun erstellt der eSprint-Client zunächst eine ISAC-Zertifikatsanfrage. In dieser Anfrage werden zum einen die Identitätsinformationen des ausstellenden Clients und zum anderen ein eigens hierfür erstelltes 4096-Bit-starkes Schlüsselpaar integriert. Nachdem diese Schritte auf Seiten des Clients abgeschlossen sind, wird das erstellte CSR an den Server übermittelt. Hierfür wird das CSR nochmals durch den Client mittels eines SHA256withRSA Algorithmus signiert, um eine zweifelsfreie

Identifikation des Clients durch den Server zu ermöglichen. Danach wird die Zertifikatsanfrage an den Server übermittelt.

```
                public class Master_ExtendedController implements Initializable
//...
 1  private void loadContent(String encId) {
 2      engineList.get(getActiveWindowIndex()).getDialog().closeFileLoaderStage();
 3      if (request_Isac(encId)) {
 4         byte[] encryptedText = request_File();
 5         if (encryptedText != null) {
 6             //...
 7             String textRead = convertContent(new String(encryptedText));
 8             textRead = extractSetContentHeader(textRead);
 9             engineList.get(getActiveWindowIndex()).setOriginalContent(textRead);
 10            //...
 11            setTabName(engineList.get(getActiveWindowIndex()).getPolicyContainer().getResTitle());
 12            statusUpdates("File loaded (" +
 13              engineList.get(getActiveWindowIndex()).getPolicyContainer()
 14              .getResTitle() + ")", false);
 15         }
 16     } else {
 17        Dialogs.showErrorDialog(stage, "ISAC Request failed!\n Please contact your Administrator! \n
 18        Server Adress: " + engineList.get(getActiveWindowIndex()).getUrlIsac_load() + "",
 19        "ISAC Request Error", "Error");
 20     }
 21  }
```

Abbildung 61: Ausschnitt – Methode: loadContent(..)

Der in Abbildung 61 abgebildete Programmausschnitt zeigt den Einstieg in den anfangs beschriebenen „Datei laden“-Prozess.

Sobald der Benutzer über die Oberfläche des eSprint-Clients eine Datei ausgewählt und den Ladeprozess gestartet hat, wechselt das Programm in die Methode „loadContent“. Als Erstes wird der zuvor dargestellte „Filebrowser“ durch die Aktion geschlossen (Zeile 2). Danach folgt in Zeile 3 bereits die Abfrage nach einem passenden ISAC-Zertifikat für die ausgewählte Datei. Hierfür wird die sogenannte „encId“ verwendet, die zuvor in verschlüsselter Form passend zu jedem Element in der dargestellten Dateiliste an den Client übermittelt wurde. Die „encId“ ist in diesem Fall mit dem Public Key des Servers chiffriert und daher auch nur von diesem entschlüsselbar. Dieser ISAC-Request wird in Abbildung 62 genauer erläutert. Danach

folgt in Zeile 4 die eigentliche Anforderung des Dokuments mithilfe des zuvor erhaltenen ISAC-Zertifikates. Nachdem das Dokument mittels der im ISAC-Zertifikat enthaltenen AES-Schlüssel dechiffriert wurde (Zeile 7), wird es zur Darstellung im Editor aufbereitet und dem Benutzer angezeigt (Zeilen 8 bis 14).

Für den Fall, dass es bei der Übermittlung der angeforderten Datei zu einem Fehler kommt, wird dies dem Benutzer über ein Dialogfenster angezeigt (Zeilen 16 bis 20).

```
public class Master_ExtendedController implements Initializable
//...
 1 private boolean request_Isac(String encId_load) {
 2     String requestUrl;
 3     if (encId_load != null) {
 4         requestUrl = engineList.get(getActiveWindowIndex()).getUrlIsac_load();
 5     } else {
 6         requestUrl = engineList.get(getActiveWindowIndex()).getUrlIsac_save();
 7     }
 8     ServerConnection isacRequest = new ServerConnection(requestUrl);
 9     Crypto crypt = new Crypto(stage);
10     if (isacRequest.connect(boolean.class)) {
11         byte[] csr;
12         if (encId_load != null) {
13             csr = crypt.csr_generator(encId_load);
14         } else {
15             csr = crypt.csr_generator();
16         }
17
18         byte[] encryptedCsr = encryptContentAES(csr, null);
19         byte[] encryptedCSRAesKey = encryptKeyRSA(isacRequest.getServerPublicKey(),
20             engineList.get(getActiveWindowIndex()).getEncryptedContent_AesKey());
21         byte[] csrAesKeySignature = crypt.createSignature(encryptedCSRAesKey,
22             isacRequest.getOwnPrivateKey());
23         MultiPart encMultiPart_AES_ISAC = isacRequest.getISAC(MultiPart.class,
24         csrAesKeySignature, encryptedCSRAesKey, encryptedCsr);
25         byte[] AES_ISAC = crypt.verifySignature(encMultiPart_AES_ISAC,
26             isacRequest.getServerPublicKey());
27         if (AES_ISAC != null) {
28             return decryptSetISAC(AES_ISAC);
29         } else {
30             return false;
31         }
32     } else {
33         statusUpdates("Could not connect to Server!", true);
34         isacRequest.close();
35         return false;
36     }
37 }
```

Abbildung 62: Ausschnitt – Methode: request_Isac(..)

Diese Methode behandelt nun den Einstiegspunkt zur Anforderung des ISAC-Zertifikats vom Server. Das Programm startet hier zunächst mit der Identifizierung, ob es sich um einen Speicher- oder Ladevorgang handelt (Zeile 3 bis 7). Danach wird eine entsprechende Verbindung zum Server initialisiert (Zeile 8) und anschließend etabliert

(Zeile 10). Wurde diese Verbindung erfolgreich hergestellt, kann der Client eine entsprechende Zertifikatsanfrage erstellen (CSR), die in der Abbildung 63 genauer erläutert wird. Hierfür wird erneut zwischen einem *Ladeprozess* (Kapitel 9.2) und einem *Speicherprozess* (Kapitel 9.1) unterschieden (Zeilen 12 bis 16).

Danach folgen die Schritte zur Generierung eines Digital Envelopes inklusive der Erstellung einer dazu passenden Signatur zur Gewährleistung der Authentizität der versendeten Nachricht. Hierfür wird zunächst ein neuer symmetrischer AES-Schlüssel erzeugt, um das CSR damit zu verschlüsseln (Zeile 18 – Variable „*encryptedCsr*"). Nachfolgend wird der erzeugte AES-Schlüssel mittels des asymmetrischen RSA-Public-Keys des Servers chiffriert (Zeile 19 – Variable „*encryptedCSRAesKey*"). In Zeile 21 (Variable „*csrAesKeySignature*") erfolgt die Erstellung einer Signatur für den zuvor genannten verschlüsselten AES-Key. Dieser Schritt dient dem Server zur Verifizierung eines unveränderten Schlüssels.

Sind diese Schritte abgearbeitet, kann der Client das erzeugte CSR an den Server übermitteln. Dies geschieht in Zeile 23 (Variable „*encMultiPart_AES_ISAC*"), wobei der Vorgang der CSR-Übermittlung in der nachfolgenden Abbildung 64 genauer erläutert wird. Als Rückgabe erhält der Client hier ein MultiPart-Objekt, welches das endgültige ISAC-Zertifikat enthält. In den Zeilen 25 bis 28 wird das erhaltene ISAC noch auf die Echtheit seiner Herkunft hin verifiziert, um danach entschlüsselt und für das aktuell geöffnete Fenster abgespeichert zu werden.

Falls der Client keine Verbindung zum Server etablieren konnte, wird dieser Fehlerfall in den Zeilen 32 bis 36 als Dialogausgabe an den Benutzer gemeldet.

public class Crypto extends Master_ExtendedController

```
1  import java.security.cert.X509Certificate;
2  import java.security.KeyPairGenerator;
3  import org.bouncycastle.pkcs.PKCS10CertificationRequestBuilder;
4  import org.bouncycastle.pkcs.PKCS10CertificationRequest;
5  import org.bouncycastle.operator.jcajce.JcaContentSignerBuilder;
6  import org.bouncycastle.operator.ContentSigner;
7  //...
8  public byte[] csr_generator(String enc_rid) {
9      ServerConnection server = new ServerConnection("newCertificate");
10     java.security.cert.Certificate cert = server.getCertificate();
11     if (cert instanceof X509Certificate) {
12         X509Certificate x509cert = (X509Certificate) cert;
13         ASN1ObjectIdentifier asndocid = new ASN1ObjectIdentifier("1.3.3.6");
14         String subjectDn = x509cert.getSubjectDN().toString();
15         String[] credentials = subjectDn.split(",");
16         String mail[] = credentials[0].split("=");
17         String cn[] = credentials[1].split("=");
18         String ou[] = credentials[2].split("=");
19         String o[] = credentials[3].split("=");
20
21         KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
22         generator.initialize(4096, new SecureRandom());
23         KeyPair pair = generator.generateKeyPair();
24         PKCS10CertificationRequestBuilder p10Builder = new JcaPKCS10CertificationRequestBuilder
25             (new X500Principal(
26                 "Email=" + mail[1] +
27                 ", CN=" + cn[1] +
28                 ", OU=" + ou[1] +
29                 ", O=" + o[1] +
30                 ""),
31                 pair.getPublic());
32         ExtensionsGenerator extensionsGenerator = new ExtensionsGenerator();
33
34         extensionsGenerator.addExtension(asndocid, false, enc_rid.getBytes());
35
36         p10Builder.addAttribute(PKCSObjectIdentifiers.pkcs_9_at_extensionRequest,
37             extensionsGenerator.generate());
38
39         JcaContentSignerBuilder csBuilder = new JcaContentSignerBuilder("SHA256withRSA");
40         ContentSigner signer = csBuilder.build(pair.getPrivate());
41         PKCS10CertificationRequest csr2 = p10Builder.build(signer);
42         //...
43         return csr2.getEncoded();
44     }
45     //...
46 }
```

Abbildung 63: Ausschnitt – Methode: csr_generator(..)

Der in Abbildung 63 dargestellte Programmcode-Ausschnitt zeigt die Erstellung des zuvor erwähnten CSR. Die Erzeugung beginnt mit der Generierung eines leeren Zertifikat-Containers (Zeilen 9 bis 12 – Variable „x509cert“). Nach der erfolgreichen Erstellung von eben diesem, wird eine neue Certificate-Extension (siehe hierfür Kapitel 7.5) mit der Kennung „*1.3.3.6*“ erzeugt (Zeile 13). Über diese Kennung kann der Server die angehängten Daten zu einem späteren Zeitpunkt wieder auslesen.

Anschließend werden alle essentiellen Informationen – die Identität des anfragenden Clients betreffend – aus dem bestehenden Zertifikats-Container, der das vorhandene Client-Zertifikat beinhaltet, ausgelesen und in temporäre Variablen geschrieben (Zeilen 14 bis 19). Das bestehende Zertifikat ist jenes, das bei der Erstellung und Auslieferung des genutzten Clients durch den Server erstellt wurde.

Sobald diese grundlegenden Schritte erledigt sind, kann ein neues asymmetrisches Schlüsselpaar erzeugt werden. In den Zeilen 21 bis 23 wird hierfür ein 4096-Bit-langer RSA-Schlüssel erzeugt und in die Variable *pair* gespeichert. In den Zeilen 24 bis 31 wird aus diesen zuvor gesammelten Daten nun ein sogenannter *CSR Builder* geschrieben. Dieser hat den Zweck, dass hierin alle relevanten Daten zur Erstellung eines CSR zwischengespeichert werden, bevor daraus das endgültige CSR generiert wird. Ebenfalls an den CSR-Builder angehängt wird die oben beschriebene *Certificate-Extension* mit der Kennung „*1.3.3.6*“. In dieser wird nun die vom Server übermittelte chiffrierte *encId* abgespeichert (Zeilen 32 bis 37).

Abschließend wird das CSR noch durch den Private Key des neu erzeugten Schlüsselpaares mittels SHA256withRSA-Algorithmus signiert. bevor es an den Server übermittelt wird (Zeilen 39 bis 41).

public class ServerConnection extends Master_ExtendedController

```
1  import com.sun.jersey.multipart.MultiPart;
2  import javax.ws.rs.core.MediaType;
3
4  //...
5
6  public <T> T getISAC(Class<T> responseType, byte[] signature, byte[] aesKey, byte[] csr) {
7     MultiPart multiPart = new MultiPart(MediaType.MULTIPART_FORM_DATA_TYPE);
8     multiPart
9              .bodyPart(signature, MediaType.APPLICATION_OCTET_STREAM_TYPE)
10             .bodyPart(aesKey, MediaType.APPLICATION_OCTET_STREAM_TYPE)
11             .bodyPart(csr, MediaType.APPLICATION_OCTET_STREAM_TYPE);
12    return webResource.type(MediaType.MULTIPART_FORM_DATA)
13             .post(responseType, multiPart);
14 }
```

Abbildung 64: Ausschnitt – Methode: getISAC(..)

Der in Abbildung 64 dargestellte Programmcode-Ausschnitt zeigt – entsprechend dem Gedanken des Digital Envelopes – die Versendung des CSR inklusive der zugehörigen Signatur und den mit dem Server-Public-Key-verschlüsselten AES-Key.

Der Client bereitet die zu sendende Nachricht auf. Mittels einer Hash-Funktion wird das zu übermittelnde Dokument vom Client zu einem Hash-Wert gewandelt, mit seinem Private Key verschlüsselt und dem Dokument als Signatur angehängt.

Anschließend generiert der Client einen neuen zufälligen symmetrischen Schlüssel, der nur im Zuge der Erstellung des Digital Envelopes verwendet wird und danach verworfen werden sollte.

Der Client nutzt diesen Schlüssel und verschlüsselt die zuvor aufbereitete Nachricht samt der Signatur.

Nachfolgend wird der zufällige symmetrische Schlüssel mit dem Public Key des Empfängers, also des Dokuments erwartenden Servers, verschlüsselt.

Abschließend wird der Public-Key-verschlüsselte zufällige symmetrische Schlüssel der verschlüsselten Nachricht angehängt und an den Server übermittelt.

11.3.2 Server – ISAC-Zertifikat für User/File kreieren

Nachdem die oben beschriebenen Schritte auf Seiten des Clients abgehandelt sind, wird das erstellte CSR an den Server mittels des in Kapitel 6.2 erläuterten Digital Envelopes transferiert.

Nachdem der eSprint-Server das CSR des Clients erhalten hat, generiert er daraus das individuelle angepasste ISAC-Zertifikat (CRT) für die betreffende Ressource. Beim Laden einer Datei wird der aufrufende Benutzer zunächst durch den Server und dessen Datenbank identifiziert und dessen Zugriffsrechte (Open, Community, Organisation und Restricted) ausgelesen. Danach wird die jeweilige DocID in das ISAC-Zertifikat eingefügt. Diese ist mittels asynchroner Verschlüsselung durch den Public Key des Servers geschützt und liegt dem Client niemals in lesbarer Form von Klartext vor. Nach der Vollendung dieser Arbeitsschritte wird das Zertifikat signiert und an den eSprint-Client zurückgesendet, wo es zur weiteren Verwendung in der aktiven Session zur Verfügung steht.

```
1  package rest.isac;
2
3  import com.sun.jersey.multipart.MultiPart;
4  import javax.servlet.http.HttpServletRequest;
5
6  @Path("/getisac")
7  @ServletSecurity(value=@HttpConstraint
8      (transportGuarantee=ServletSecurity.TransportGuarantee.CONFIDENTIAL))
9  public class IsacResource extends HttpServlet {
10    @POST
11    @Path("/load")
12    @Consumes(MediaType.MULTIPART_FORM_DATA)
13    @Produces({MediaType.MULTIPART_FORM_DATA})
14    public MultiPart createIsac_load(MultiPart mp, @Context HttpServletRequest hsr) {
15        boolean verified = CryptImpl.verify(mp, hsr);
16        if(verified){
17            MultiPart load = IsacImpl.createIsacLoad(mp, hsr);
18            return load;
19        } else {
20        //...
21        }
22    }
23 }
```

Abbildung 65: Ausschnitt IsacResource.java

Als Ressource wird ein Zugangspunkt bezeichnet, unter dem eine Funktion eines Webservices angesprochen werden kann. Die Hintergründe hierzu finden sich im Kapitel 4.1 Uniform-Resource-Identifier. Bei Übermittlung des CSR spricht der Client direkt die „Resource“ /getisac/load des Webservice an. Abbildung 65 zeigt einen Code-Ausschnitt der Datei IsacResource.java.

Mit der Annotation „@path“, zu sehen in Zeile 6 und 11, wird die Ressource innerhalb des Webservice entsprechend registriert. Der Client Request, der das CSR mitsendet, kann somit durch die in Zeile 14 ersichtliche Methode creatIsac_load behandelt werden. Gleich im ersten Schritt dieser Methode (Zeile 15) wird die Signatur in der übertragenen Nachricht überprüft. Die aufgerufene Methode ist in Abbildung 66 dargestellt. Nach positiver Überprüfung wird das CSR innerhalb der Methode createIsacLoad (Aufruf: Zeile 17, Implementation: Abbildung 67) befüllt und zum fertigen ISAC-Zertifikat (CRT) gewandelt. Das Ergebnis wird dann an den aufrufenden Client retourniert.

```
1  package crypt;
2  import com.sun.jersey.multipart.MultiPart;
3  import javax.servlet.http.HttpServletRequest;
4  import java.security.Signature;
5  import java.security.cert.X509Certificate;
6
7  public class CryptImpl {
8  public static boolean verify(MultiPart mp, HttpServletRequest hsr) {
9     byte[] signature = mp.getBodyParts().get(0).getEntityAs(byte[].class);
10    byte[] content = mp.getBodyParts().get(1).getEntityAs(byte[].class);
11   Signature sHA512withRSA = Signature.getInstance("SHA512withRSA");
12   X509info x509Info = new X509info(
13       (X509Certificate[]) hsr.getAttribute("javax.servlet.request.X509Certificate")
14   );
15   sHA512withRSA.initVerify(x509Info.getPublicKey());
16   sHA512withRSA.update(content);
17   return sHA512withRSA.verify(signature);
18 }
```

Abbildung 66: Ausschnitt CryptImpl.java

Die Methode *verify* der Klasse *CryptImpl* überprüft die Signatur der übermittelten Nachricht. Dies dient der Umsetzung der zuvor in Kapitel 5.2 behandelten Nachrichten-Authentifizierung bei der Public-Key-Verschlüsselung bzw. des in Kapitel 7.2 behandelten Digital Envelopes. Eine Veranschaulichung der durchgeführten Schritte gibt es in Abbildung 25.

Innerhalb der CryptImpl.java stellt Zeile 10 die Nachricht und Zeile 9 die dazu angehängte Signatur dar. Die verschlüsselte Signatur wird mittels des Public Keys des Senders, also des Clients, entschlüsselt und die übermittelte Nachricht mittels der vereinbarten Hash-Funktion ebenfalls zum Hash-Wert gewandelt. Nun können der selbst erzeugte und der übermittelte Hash-Wert verglichen werden. Stimmen diese überein, kann der Empfänger davon ausgehen, dass die Nachricht unverfälscht geblieben ist und vom erwarteten Empfänger gekommen ist.

Im Code wird dies durch ein Objekt der Klasse java.security.Signature abgehandelt. Für dieses wird, zuerst bei der Instanziierung des Objektes, in Zeile 11 der verwendete Public-Key-Algorithmus definiert. In Zeile 15 und 16 werden der Public Key des Senders sowie der signierte Inhalt gesetzt. Mittels der verify-Methode des Signature-Objektes werden dann die beschriebenen Schritte durchgeführt und der Boolsche Wert „true“ bei erfolgreicher Verifizierung zurückgegeben.

```
1  package rest.isac;
2
3  import certGen.CertGenerator;
4  import com.sun.jersey.multipart.MultiPart;
5  import crypt.CryptImpl;
6  import crypt.X509info;
7  import database.DBqueries;
8  import java.security.cert.X509Certificate;
9  import javax.crypto.spec.SecretKeySpec;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.ws.rs.core.MediaType;
12 import org.bouncycastle.pkcs.PKCS10CertificationRequest;
13
14  public class IsacImpl {
15      public static MultiPart createIsacLoad(MultiPart multiPart, HttpServletRequest hsr) {
16
17          byte[] encryptedcsrbytes = multiPart.getBodyParts().get(2)
18              .getEntityAs(byte[].class);
19          byte[] encryptedAesKey = multiPart.getBodyParts().get(1)
20             .getEntityAs(byte[].class);
21         byte[] aesKey = CryptImpl.decryptRsa(encryptedAesKey);
22         SecretKeySpec Key = new SecretKeySpec(aesKey, "AES");
23         byte[] csrbytes = CryptImpl.decryptAes(encryptedcsrbytes, Key);
24         PKCS10CertificationRequest csr = new PKCS10CertificationRequest(csrbytes);
25         X509info x509Info = new X509info((X509Certificate[]) hsr
26             .getAttribute("javax.servlet.request.X509Certificate"));
27         Integer cid = DBqueries.getUserId(x509Info.getMail(), x509Info.getCn(), x509Info.getOu());
28         byte[] isac = CertGenerator.createISAC_load(csr, cid);
29         byte[] ret = CryptImpl.encrypt(isac, x509Info.getPublicKey());
30
31         MultiPart multi = new MultiPart(MediaType.MULTIPART_FORM_DATA_TYPE);
32         multi
33                 .bodyPart(CryptImpl.getSignature(ret), MediaType.APPLICATION_OCTET_STREAM_TYPE)
34                 .bodyPart(ret, MediaType.APPLICATION_OCTET_STREAM_TYPE);
35
36         return multi;
37     }
38  }
```

Abbildung 67: Ausschnitt IsacImpl.java

Abbildung 67 zeigt die Methode *createIsacLoad* der Klasse *IsacImpl*. Die Klasse stellt die Methoden zur Implementierung der in IsacResource.java definierten Zugangspunkte bereit. In der createIsacLoad-Methode wird zuerst entsprechend dem Gedanken des Digital Envelopes (Abbildung 31) vorgegangen.

Zuerst wird der vom Client, mit dem Public Key des Servers, verschlüsselte symmetrische Schlüssel entnommen (Zeile 17). Dieser wird dann mit dem eigenen Private Key entschlüsselt (Zeile 21). Der daraus gewonnene symmetrische Schlüssel wird verwendet, um die verschlüsselte Nachricht zu entschlüsseln (Zeile 22, 23). Der Server erhält so das CSR (Zeile 24). Die Überprüfung der Signatur wurde zuvor abgehandelt. Durch dieses Vorgehen kann sich der Server sicher sein, dass das Dokument nur vom erwarteten Client kommen kann und der Client kann sich sicher sein, dass nur der Server den versandten Schlüssel dekodieren und somit auch auf die übermittelte Nachricht zugreifen kann. Zudem ist durch die symmetrische Verschlüsselung des Dokuments eine bessere Performance des Systems gewährleistet.

Nach Auslesen des CSR wird der Client noch anhand seines SSL-Zertifikats identifiziert (Zeile 27) und die Client-Identifikationsnummer gemeinsam mit dem CSR an die Methode CertGenerator.createISAC_Load übergeben. Diese übernimmt dann das Erstellen des ISAC-Zertifikates. Der Rückgabewert wird im Anschluss entsprechend signiert und zurückgegeben.

```
1  package certGen;
2
3  import java.io.*;
4  import java.net.*;
5  import java.security.*;
6  import java.sql.SQLException;
7  import java.util.Date;
8  import org.bouncycastle.*;
9
10  public class CertGenerator {
11    public static byte[] createISAC_load(PKCS10CertificationRequest csr, Integer cid) {
13     //...
14       X509V3CertificateGenerator certGen = new X509V3CertificateGenerator();
15       X500Name x500Name = csr.getSubject();
16       final String ORGANIZATION = "2.5.4.10";
17       final String ORGANIZATION_UNIT = "2.5.4.11";
18       final String COMMON_NAME = "2.5.4.3";
19       final String EMAIL = "1.2.840.113549.1.9.1";
20       final String DOCID = "1.3.3.6";
21       ASN1ObjectIdentifier docId = new ASN1ObjectIdentifier(DOCID);
22       X500Principal subjectName = new X500Principal(
23           "Email=" + getX500Field(EMAIL, x500Name) +
24           ", CN=" + getX500Field(COMMON_NAME, x500Name) +
25           ", OU=" + getX500Field(ORGANIZATION_UNIT, x500Name) +
26           ", O=" + getX500Field(ORGANIZATION, x500Name) + "");
27
28       X509Extensions x509Ext = getX509ExtensionsFromCsr(csr);
29       X509Extension x509Doc = x509Ext.getExtension(docId);
30       ASN1OctetString aos = x509Doc.getValue();
31       byte[] docid_enc = removeInvalidCharacters_decodeHex(aos.getEncoded());
32       byte[] doc_id = CryptImpl.decryptRsa(docid_enc);
```

Abbildung 68: Ausschnitt CertGenerator.java Teil 1

Die Ausschnitte in Abbildung 68 und Abbildung 69 zeigen die Methode *createISAC_Load* der Klasse *CertGenerator*. Hier erfolgt die Erzeugung des ISAC-Zertifikates aus dem erhaltenen CSR. Die theoretischen Hintergründe zum Thema digitale Zertifikate und ihre Erstellung befinden sich im Kapitel 7.4.1.

Im ersten Schritt zeigt Abbildung 68 das Auslesen der Werte aus dem CSR. In den Zeilen 15–31 werden die CSR-Standardfelder (Tabelle 5), die Extensions sowie die darin enthaltene ID des zu ladenden Dokuments ausgelesen. Die im CSR übermittelte DokumentID ist aber mit dem Public Key des Servers verschlüsselt. Daher muss diese noch mit dem eigenen Private Key entschlüsselt werden (Zeile 32).

```
33 if (DBqueries.hasAccess(new String(doc_id), cid)) {
34  String aeskeys[] = DBqueries.getAesKeys(new String(doc_id));
35  byte[] aes_doc = CryptImpl.decryptRsa(Hex.decode(aeskeys[0]));
36  byte[] aes_open = CryptImpl.decryptRsa(Hex.decode(aeskeys[1]));
37  byte[] aes_comm = CryptImpl.decryptRsa(Hex.decode(aeskeys[2]));
38  //...
39  Boolean[] accrights = DBqueries.accessRights(new String(doc_id), cid);
40
41  certGen.setSerialNumber(serialNumber);
42  certGen.setIssuerDN(caCert.getSubjectX500Principal());
43  certGen.setNotBefore(startDate);
44  certGen.setNotAfter(expiryDate);
45  certGen.setSubjectDN(subjectName);
46
47  RSAKeyParameters pubkey =
48          (RSAKeyParameters) PublicKeyFactory.createKey(csr.getSubjectPublicKeyInfo());
49  RSAPublicKeySpec rsaSpec = new RSAPublicKeySpec(pubkey.getModulus(), pubkey.getExponent());
50  KeyFactory kf = KeyFactory.getInstance("RSA");
51  PublicKey rsaPub = kf.generatePublic(rsaSpec);
52  certGen.setPublicKey(rsaPub);
53  certGen.setSignatureAlgorithm("SHA512withRSA");
54  ASN1ObjectIdentifier asndocid = new ASN1ObjectIdentifier("1.3.3.6");
55  certGen.addExtension(asndocid, false, Hex.encode(docid_enc));
56  ASN1ObjectIdentifier asndoc = new ASN1ObjectIdentifier("1.3.3.7");
57  if (accrights[0])
58      certGen.addExtension(asndoc, false, Hex.encode(aes_doc));
59  ASN1ObjectIdentifier asnopen = new ASN1ObjectIdentifier("1.3.3.7.1");
60  if (accrights[1])
61      certGen.addExtension(asnopen, false, Hex.encode(aes_open));
62  ASN1ObjectIdentifier asncomm = new ASN1ObjectIdentifier("1.3.3.7.2");
63  if (accrights[2])
64      certGen.addExtension(asncomm, false, Hex.encode(aes_comm));
65  //...
66  X509Certificate cert = certGen.generate(caKey);
67  byte[] head = "-----BEGIN CERTIFICATE-----\n".getBytes("US-ASCII");
68  byte[] tail = "\n-----END CERTIFICATE-----\n".getBytes("US-ASCII");
69  byte[] certificate = new byte[head.length +
70      tail.length +
71      b64.encode(cert.getEncoded()).getBytes().length];
72   //...
73  return certificate;
74  }
75  //...
```

Abbildung 69: Ausschnitt CertGenerator.java Teil 2

Nachdem das CSR ausgelesen ist, wird das neue zu versendende ISAC-Zertifikat befüllt und generiert. In Zeile 33 wird noch überprüft, ob der anfragende Client Zugriff

auf das gewünschte Dokument hat. Besitzt er dies, werden die AES-Keys für die einzelnen Zugriffsarten (Open, Community, Organisation und Restricted), sofern er auch die Berechtigung für diese besitzt, ausgelesen und in die ISAC-Extensions geschrieben (Zeile 34–39 und 56–64). Weiterhin werden die Seriennummer, die Aussteller-ID, der Gültigkeitszeitraum, der ausgemachte Algorithmus sowie der aus dem CSR gelesene Public Key gesetzt (Zeile 41–53). In Zeile 66 ist die Generierung des Zertifikats mithilfe des Private Keys des CA zu sehen. Abschließend wird das Zertifikat in ein sendbares Byte-Array umgewandelt, Base64 kodiert und entsprechend dem gewünschtem Format mit „-----BEGIN CERTIFICATE-----“ und „-----END CERTIFICATE-----“ umschlossen (Zeile 67–73).

12 Interfacebeschreibung

Wie schon im vorangegangenen Kapitel 11.3 wird sich die nachfolgende Interfacebeschreibung auf einen einzelnen Prozess beziehen. Hierfür wird erneut der bereits aus Kapitel 9.2 bekannte Prozess „Datei laden“ herangezogen und mittels einer bildlichen Darstellung der eSprint-Client-Anwendung genauer beschrieben. Als Kontext dient hier der Anwendungsfall aus Kapitel 2. Dieser befasst sich mit einer einstweiligen Verfügung nach einem Belästigungsvorfall.

Das Ziel dieses Kapitels ist es, zunächst einen Überblick über die Anwendung im Allgemeinen und die Bedienelemente der Anwendung sowie deren Zweck im Speziellen zu geben und dann den exemplarischen Durchlauf des Prozesses abzubilden.

12.1 Überblick

In der unten dargestellten Abbildung 70 wird ein Überblick über die eSprint-Anwendung gegeben. In dieser wird das bereits in Kapitel 2 erläuterte Anwendungsszenario zu Veranschaulichungszwecken abgebildet.

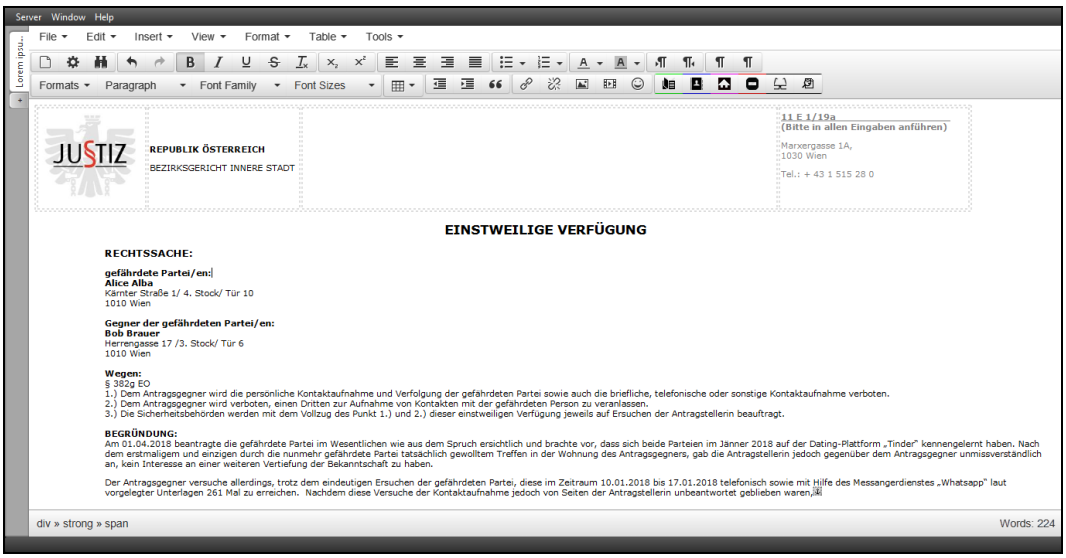

Abbildung 70: Überblick – eSprint-Client

12.2 Bedienelemente

Die in Abbildung 71 gezeigte Darstellung des eSprint-Clients gibt einen kurzen Überblick über die vorhandenen (Text-) Bedienelemente. Diese dienen vorwiegend dem Zweck der Formatierung des jeweiligen Dokuments und entsprechen in etwa den bekannten Optionen aus anderen Texteditoren. Weiters gibt es Steuerelemente zum Hinzufügen oder Entfernen weiterer Editor- und/oder Policy-Fenster.

Abbildung 71: Überblick Bedienelemente – eSprint-Client

Element	Beschreibung
1	*Seitenleiste* In der Tab- oder Seitenleiste befinden sich die Steuerelemente zum Hinzufügen oder Entfernen neue Editor- und Policy-Fenster.
2	*Bedienleiste* Über die obere Bedienleiste könne diverse Editoroperationen durchgeführt werden. Hierzu zählen etwa: • Speichern einer Datei • Aufrufen der Dokumenteneigenschaften • Finden und Ersetzen von Text innerhalb des Editors • Einfügen von Sonderzeichen oder Formatierungshilfen • Anzeigen von versteckten Formatierungen • Einfügen von Tabellen • Anzeigen des HTML-Quellcodes der erstellten Datei
3	*Schnellformatierung* Mittels der Schnellformatierungsleiste lassen sich die wichtigsten Textformatierungen durch einfachen Knopfdruck durchführen.
4	*eSprint-Steuerung* Die Elemente bei Punkt 4 stellen das Kernstück des eSprint-Systems dar. Hierüber lassen sich im Text die individuellen Textstellen für eine spätere De-Perimeterisation kennzeichnen. Darüber hinaus kann hier auch der zugehörige Policy-Editor zur Anpassung der Zugriffsrechte geöffnet werden.
5	*Editor-Fenster* Im eigentlichen Editor-Fenster kann der Benutzer des eSprint-Clients seine Texte erstellen, formatieren und mittels der eSprint-Steuerelemente kennzeichnen.

Tabelle 28: Bedienelemente eSprint-Client

12.2.1 Shortcuts

Neben der Nutzung der Schaltflächen zur Steuerung des Editors gibt es auch noch die Möglichkeit der Bedienung mittels Tastatur-Kürzel. Für die wichtigsten Operationen wurden Shortcuts in den Editor implementiert, um die Nutzung von eben diesem für den Benutzer zu erleichtern. Diese Kürzel umfassen:

Tastatur-Kürzel	Beschreibung
Strg + S	Speichern der aktuellen Datei
Strg + L	Laden einer neuen Datei im aktuellen Tab
Strg + N	Öffnen eines neuen Editor-Tabs
Alt + P	Öffnen des Policy-Editors
Alt + 1	Kennzeichnung „Open Access“
Alt + 2	Kennzeichnung „Community Access“
Alt + 3	Kennzeichnung „Organisation Access“
Alt + 4	Kennzeichnung „Restricted Access“

Tabelle 29: Tastaturkürzel – eSprint-Client

12.3 Prozess „Laden einer Datei“

In den nachfolgenden Abbildungen wird von der Situation ausgegangen, dass bereits eine entsprechende Datei durch den Dateiinhaber (Owner) erstellt wurde und er diese erneut öffnen möchte, um zusätzliche Zugriffsrechte für weitere beteiligte Parteien freizugeben.

12.3.1 Ladedialog

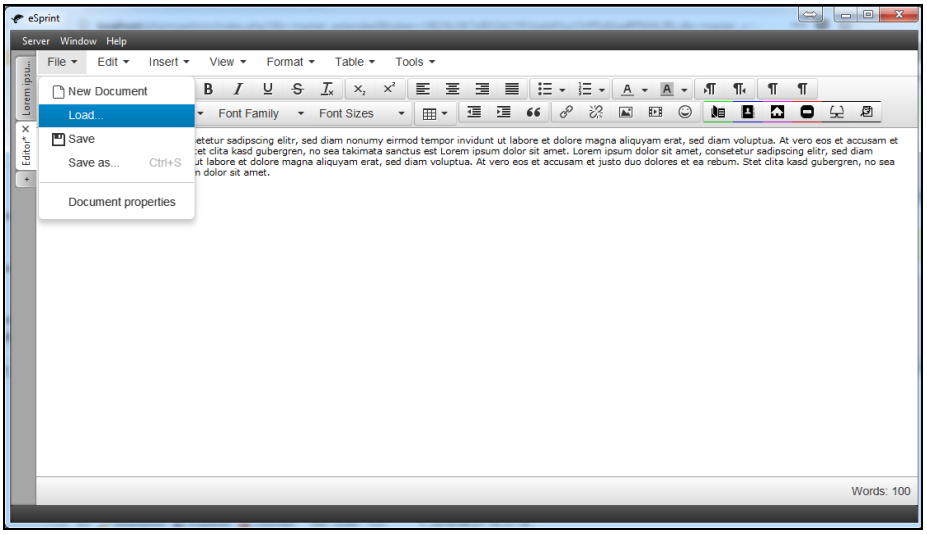

Abbildung 72: Ladeprozess starten

Der Prozess beginnt damit, dass der eSprint-Client durch den Benutzer gestartet wird. Sobald die Anwendung läuft, kann der Benutzer über das Menü „File“ den Menüpunkt „Load…“ auswählen. Hiermit startet und öffnet er den Subprozess „Ladedialog“. Dieser fordert zunächst alle vorhandenen Dateien an, zu denen der Client Zugriffsrechte hat und gibt diese dem Benutzer über den „Filebrowser“ aus.

12.3.2 Filebrowser

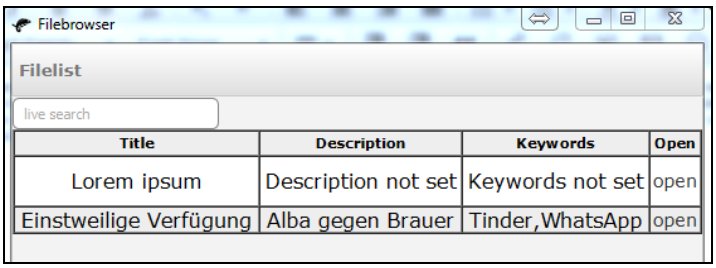

Abbildung 73: Dateiübersicht im Filebrowser

Sobald sich das Dialogfenster „Filebrowser“ öffnet, kann der Benutzer alle Dateien einsehen, die für ihn freigegeben sind. Diese werden in einer Liste dargestellt, die über

eine „Live“-Suche durchsucht werden kann. Durchsuchbar sind in diesem Fall alle vorhandenen Felder, wodurch es dem Benutzer leichter gemacht wird, die gewünschte Datei zu finden.

Nachdem er das gewünschte File gefunden hat, kann er dieses über die rechts befindliche Schaltfläche „open“ öffnen.

12.3.3 Policy-Editor

Nachdem die Datei geladen wurde, lässt sie sich mit den in Kapitel 12.2 vorgestellten Bedien- und Steuerelementen modifizieren.

Möchte der Eigentümer der Datei anderen Benutzern das Einsehen und Bearbeiten eben dieser ermöglichen, so muss diese Freigabe über den Policy-Editor geschehen. Um diesen zu öffnen, kann entweder das in Kapitel 0 vorgestellte Tastaturkürzel „Alt + P“ verwendet oder aber die entsprechende Schaltfläche aus der *eSprint-Steuerung* genutzt werden. Danach öffnet sich ein neuer Tab in der *Seitenleiste,* über die der Policy-Editor erreicht wird.

12.3.4 Gruppen- und Zugriffsrechteübersicht

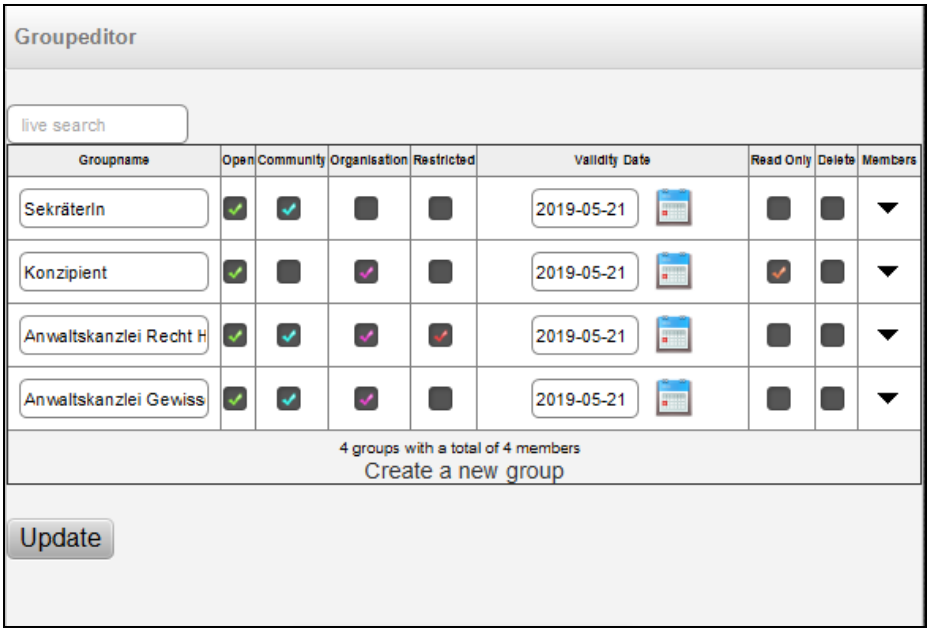

Abbildung 74: Gruppenübersicht im Policy-Editor

Im Policy-Editor können zunächst alle Gruppen und deren zugehörige Zugriffsrechte und Gruppenmitglieder eingesehen werden.

Für den Fall, dass für eine Ressource eine hohe Anzahl verschiedener Gruppen existiert, kann auch hier wiederum die „Live"-Suche zur Auffindung der gewünschten Benutzer verwendet werden. Falls noch keine Gruppe mit entsprechenden Zugriffsrechten existiert, kann diese über den seitlichen Menüpunkt „Create groups" oder über den Punkt „create a new group" angelegt werden.

12.3.5 Gruppenerstellung

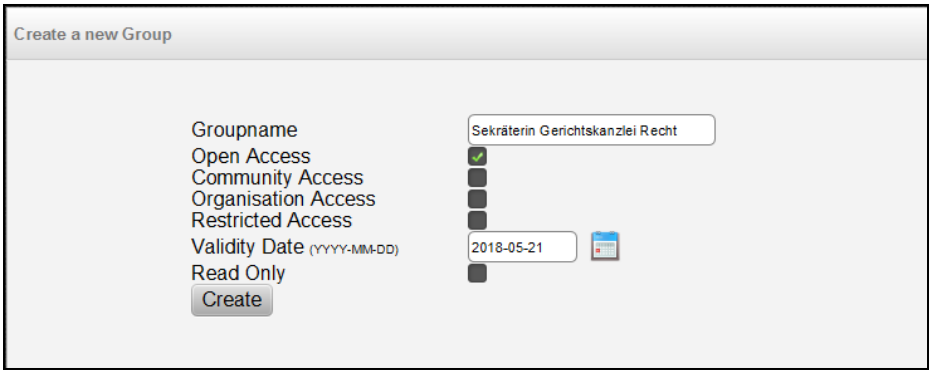

Abbildung 75: Erstellung einer Gruppe – Eigenschaften

Wenn nun eine neue Gruppe für die aktuelle Datei erstellt wird, muss zunächst ein Gruppenname festgelegt werden. Dieser dient einer späteren leichteren Auffindbarkeit. Außerdem müssen die entsprechenden Gruppenrechte für die jeweilige Gruppe vergeben werden. Darüber hinaus muss ein Gültigkeitszeitraum für die Gruppe festgelegt werden. Abschließend kann noch bestimmt werden, ob diese Gruppe eine Datei lediglich lesen oder auch bearbeiten darf.

Nachdem die Erstellung der Gruppe abgeschlossen ist, erfolgt eine Weiterleitung zur Benutzerauswahl.

12.3.6 Benutzerauswahl

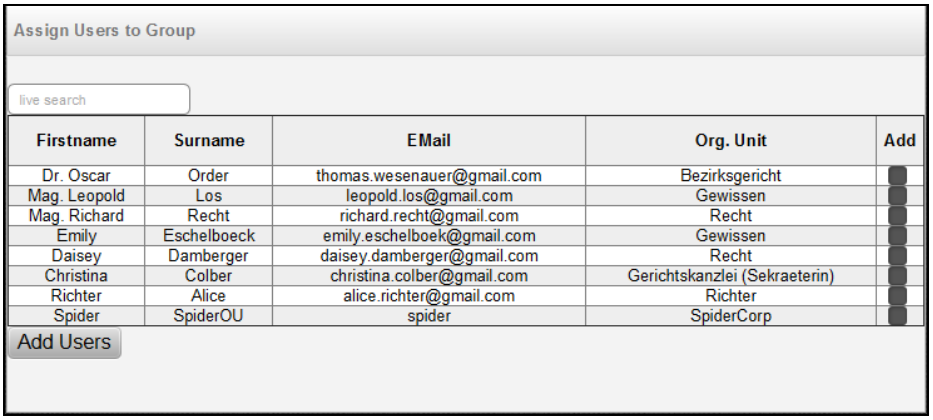

Abbildung 76: Erstellung einer Gruppe – Mitglieder

Im Zuge der Gruppenerstellung können dieser neuen Gruppe auch alle gewünschten Gruppenmitglieder zugeordnet werden. Hierfür wird nach der Festlegung der Gruppeneigenschaften zur Benutzerauswahl weitergeleitet. Diese zeigt alle im System vorhandenen Clients an, was im Laufe der Zeit zu einer langen Liste anwachsen kann. Aus diesem Grund wurde auch hier wieder eine „Live"-Suche eingefügt, um dem Benutzer eine einfachere Navigation und Auffindbarkeit zu ermöglichen.

In der Auswahlliste werden alle relevanten Informationen zu den jeweiligen Benutzern ausgegeben. Hierzu zählen etwa der Vor- und der Zuname, die E-Mail-Adresse sowie die Organisation/Abteilung des Benutzers. Die Auswahl der Benutzer erfolgt über ein gesetztes Häkchen neben den Benutzern. Abgeschlossen wird der Prozess mittels der Betätigung der Schaltfläche „Add Users".

12.3.7 Gruppen- und Zugriffsrechteübersicht

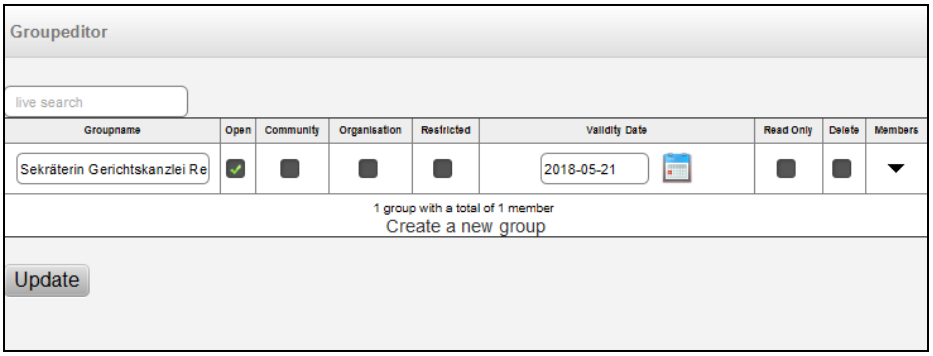

Abbildung 77: Gruppenübersicht

Nachdem die Gruppe erstellt wurde, wird der Benutzer wieder zurück zu der Gruppenübersicht geführt. In dieser Ansicht erhält er nochmals einen Überblick über die eben erstellte als auch die vorhandenen Gruppen, wie es bereits in Kapitel 12.3.4 beschrieben wurde.

13 Einordnung und Diskussion der Ergebnisse

In diesem Abschnitt soll abschließend erneut die Tabelle 2 des Kapitels 3.4 betrachtet und anhand dieser die vorliegende Arbeit bewertet werden. Die Tabelle zeigt die von Frau Nisreen Alam Aldeen im Zuge ihrer Arbeit „Enhancing Privacy Protection in Distributed Enviroments through Identification and Authentification-Based Secure Data-Level Access Control“ aufgestellte Vergleichsmatrix, die hier geringfügig erweitert wurde. Die Tabelle enthält die von eSprint angestrebten neun Systemeigenschaften:

1. *Kontrolle außerhalb von Systemgrenzen*
   Durch Implementierung des ISAC-Zertifikates kann das eSprint.System Restriktionen über Systemgrenzen hinweg kontrollieren.
2. *Keine zusätzliche Last für das Schlüsselverwaltungs-Management*
   Es treten keine besonderen Lasten für die Schlüsselverwaltung innerhalb des eSprint-Systems auf. Eine Steigerung der Last gegenüber dem Vorgänger

SPIDER ist lediglich aufgrund des Einsatzes mehrerer Schlüssel vorhanden. Die Last steigert sich demnach mit der Anzahl der Schlüssel – im vorliegenden Fall um den Faktor Fünf.

3. *Bedienbare Anwendung für die Zugriffskontrolle*

   Durch die Entwicklung des eSprint-Systems konnte unter Zuhilfenahme von Java, JavaFX, JavaScript und HTML5 eine Anwendung geschaffen werden, mit deren Hilfe ein Benutzer Zugriff auf eigene Dokumente einfach und effizient steuern kann.

4. *Ein feingranulares Klassifikations-Schema*

   Das eSprint-System übernimmt hier die in der ccREL Technologie vorgestellte Variante. Übernommen wurden die von dort bekannten Klassifikations-Schemata in Form von „Tags“. Ein Benutzer kann durch das Markieren von Textteilen und mittels der Betätigung eines der eSprint-Steuerelemente (Kapitel 12.2 Bedienelemente) Zugriffsarten in eben diese Textpassage injizieren. Durch die „Tags“ können die fixen Grenzen rund um eine Ressource entfernt und innerhalb dieser Ressource neu, nur rund um den zu schützenden Inhalt, definiert werden.

5. *Bearbeitbare und durchgängige Zugriffskontrolle nach dem Teilen von Informationen*

   Durch den Einsatz des im Zuge dieser Arbeit entwickelten ISAC-Zertifikats kann ein Dokument-Besitzer einzelnen Benutzern spezielle Lese- und Schreibrechte für sein Dokument erteilen. Diese Rechte können zeitlich begrenzt oder nachträglich durch den Besitzer wieder entzogen werden.

6. *Einfache Handhabung durch den Besitzer eines Dokuments*

   Zugriffsrechte können durch Dokument-Besitzer einfach über den Policy-Editor innerhalb eines eSprint-Clients bearbeitet werden.

7. *Einfache Handhabung durch einen aufrufenden Client*

   Ein aufrufender eSprint-Client zeigt einem Benutzer nur das, was er anhand seiner Zugriffsrechte sehen darf, also die vom Dokument-Besitzer freigegebenen Textpassagen. Je nach Rechtevergabe können diese vom aufrufenden Client mittels vielfältiger Editor-Möglichkeiten bearbeitet werden.

8. *Verschlüsselung durch Verwendung mehrerer Schlüssel*
   Innerhalb des eSprint-Systems werden für jedes Dokument immer fünf symmetrische AES256-Schlüssel erstellt. Mittels derer werden die Textpassagen der vier Zugriffsrechte innerhalb des Dokuments und das Dokument selbst verschlüsselt. Durch sie erhält das feingranulare Klassifikations-Schema des eSprint-Systems den notwendigen Schutz.
9. *Identifikation und Zugriff mittels digitaler Zertifikate*
   Die Kommunikation des eSprint-Systems ist durch eine SSL-Verbindung geschützt. Zugriff auf einzelne Dokumente erhält ein Benutzer nur, wenn er im Besitz eines passenden und gültigen ISAC-Zertifikats ist.

13.1 Umsetzungsphasen

Zum Startzeitpunkt der Entwicklung dieses Projektes waren mehrere Technologien am Markt verfügbar, die zur Darstellung des Frontends geeignet gewesen wären. Es galt also zu evaluieren, welche Anforderungen die Applikation für den Endnutzer erfüllen muss, um eine möglichst effiziente, aber auch übersichtliche Darstellung erzielen zu können.

Der Ausgangspunkt für den eSprint-Client war das Vorgängerprojekt namens SPIDER. Dieses setzte Java-Swing zur Applikationsdarstellung ein. Jedoch existiert hierfür kein Referenzprogramm, um diese Annahme zu bestätigen. Als Beispiel für diese ursprüngliche Umsetzung dient das in Abbildung 78 dargestellte Mock-up.

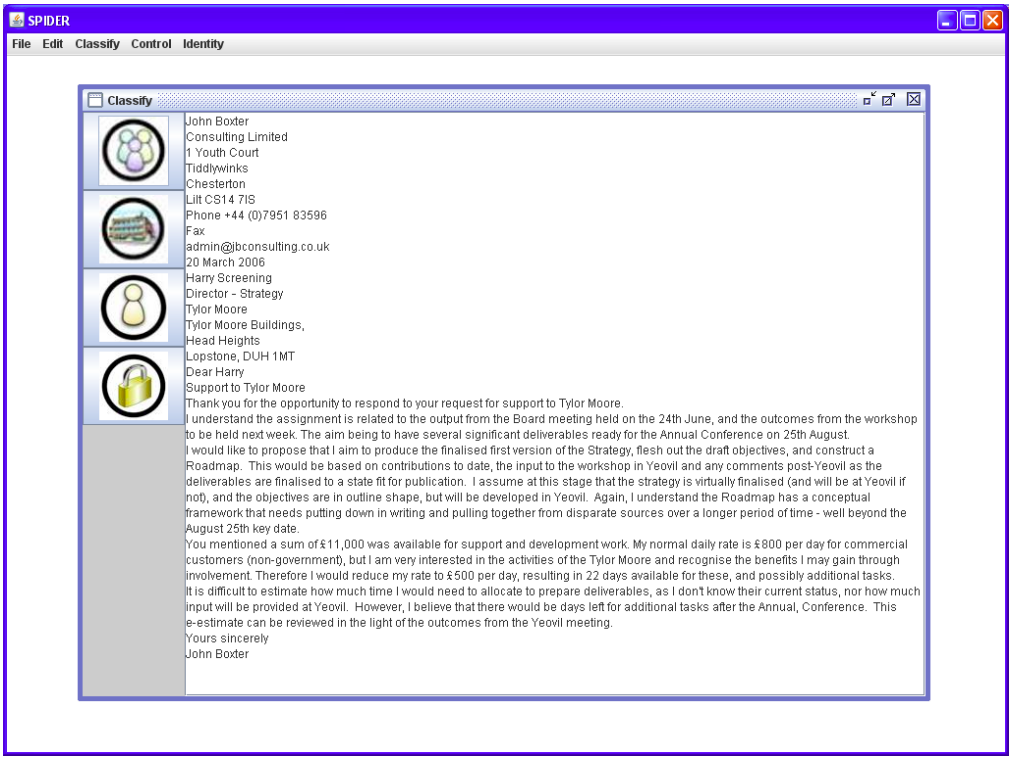

Abbildung 78: SPIDER-Projekt – Konzept

Während unserer Entscheidungsfindung erprobten wir mehrere Möglichkeiten der Umsetzung, um die für uns geeignetste zu finden. Im Zuge dessen versuchten wir, analog zu SPIDER, zunächst eine Umsetzung mit der JavaFX Vorgängertechnologie namens Swing. Diese stellt zwar eine ausgereifte und erprobte Technologie dar, jedoch konnte sie nicht alle Anforderungen zu diesem Zeitpunkt erfüllen (ausgereifte Unterstützung von *WebViews* für die Darstellung umfangreicher Texteditoren). Aus diesem Grund setzten wir in einem ersten Entwurf das eSprint-System wie in Abbildung 79 dargestellter Form um. Mit dieser Art von Editor stießen wir jedoch bald auf technische Grenzen, deren Überwindung zu sehr kompromissbehaftet gewesen wäre.

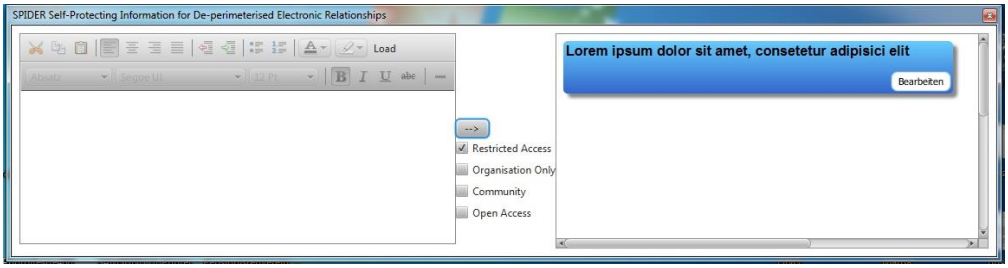

Abbildung 79: eSprint Client – erster Entwurf

Eine weitere unserer Umsetzungsideen, zu sehen in Abbildung 80, basierte erneut auf JavaFX und beinhaltete eine modifizierte Variante des zuvor bereits verwendeten Editors. Erneut ließen sich hiermit jedoch nicht alle Umsetzungswünsche von Frau Alam Aldeen und uns realisieren, weswegen wir nach einem komplett neuen Ansatz zu forschen begannen.

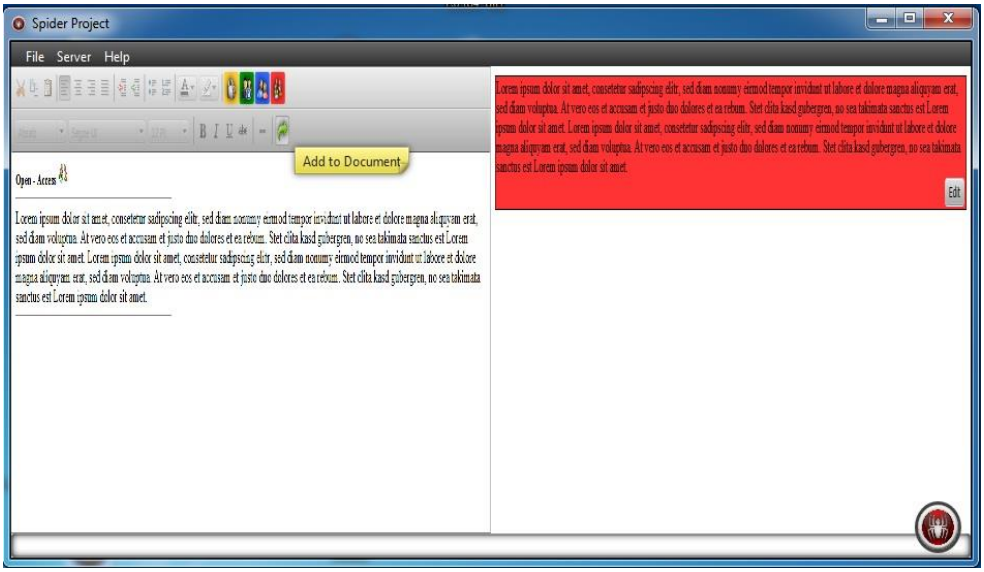

Abbildung 80: eSprint-Client – zweiter Entwurf

Unser finales Konzept basierte noch immer auf JavaFX, vereinte jedoch diverse weitere Technologien zur Darstellung aller benötigten Elemente in sich. Zum einen nahmen wir nun die (damals neu geschaffene) Möglichkeit zum Einsetzen eines WebViews wahr, um unseren Editor und den damit geschaffenen Text darzustellen.

Zum anderen kamen damit verbunden auch die Technologien JavaScript, HTML5 sowie CSS zum Einsatz. Dies ermöglichte uns ein breites Spektrum an Möglichkeiten, um dem Benutzer einen umfangreichen Editor mit vielen Gestaltungsmöglichkeiten und intuitiver Bedienbarkeit bieten zu können. Das Resultat hierzu ist in Abbildung 81 zu sehen.

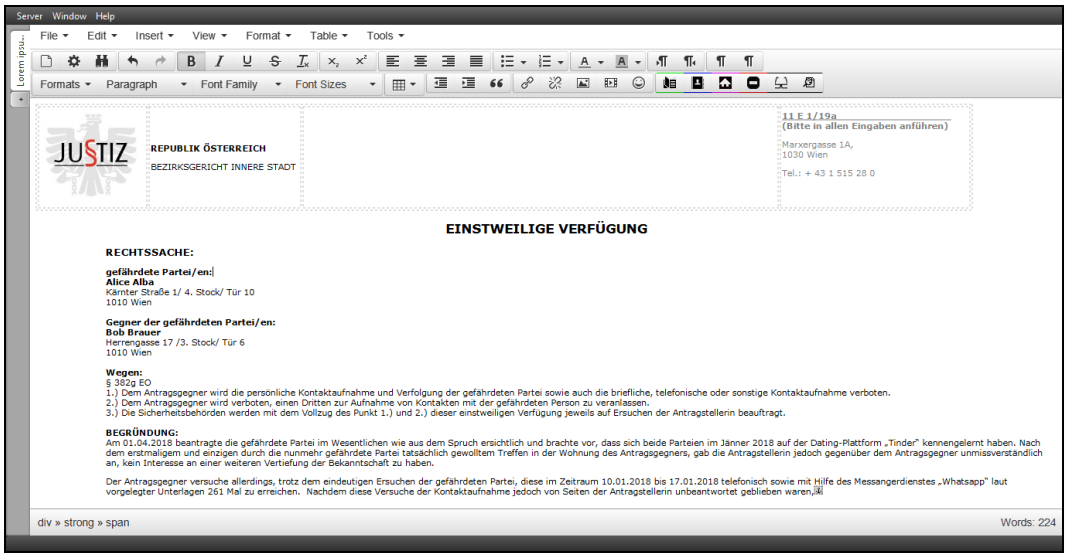

Abbildung 81: eSprint.Client – finales Erscheinungsbild

Die oben dargestellte Abbildung verdeutlicht, dass unser Client nicht mehr viel mit dem ursprünglichen SPIDER-Projekt zu tun hat. Zwar setzen wir dessen Ideen um, jedoch wurden diese um viele weitere und neue Maßnahmen verbessert und erweitert. Das Interface hat sich zu einem vollwertigen Texteditor entwickelt und durch den enthaltenen Policy-Editor, den Gruppenrichtlinien, den Multiple Keys, der Continous-Access-Control, dem verarbeiteten technologischen Fortschritt, dem ISAC-Zertifikat sowie der Live-URL sind wir unserem Vorgänger in allen Belangen überlegen.

13.2 Offene Fragen und Ausblick

Dieser Abschnitt soll abschließend unsere Gedanken zum aktuellen Stand unseres eSprint-Systems wiedergeben und eine eventuelle mögliche Zukunft davon betrachten.

Hierzu wollen wir kurz Themengebiete, die nicht im Zuge dieser Arbeit bearbeitet wurden, aufgreifen und mögliche Anpassungen des eSprint-Systems beleuchten.

Zunächst lässt sich zusammenfassen, dass wir mit der von uns erzielten Lösung zufrieden sind. Die zu Projektbeginn aufgekommenen Bedenken bezüglich der Umsetzbarkeit der eSprint-Idee konnten nach einer intensiveren Beschäftigung mit technologischen Möglichkeiten ausgeräumt werden. Zwar waren Herausforderungen zu bewältigen, aber letztlich kann die entwickelte Lösung in den zentralen funktionalen Aspekten überzeugen und bietet wenige Punkte, an denen für sinnvolle Erweiterungen unseres Systems angesetzt werden könnte.

Ein Punkt, dem wir während der Umsetzung nur beschränkt Aufmerksamkeit gewidmet haben, aber dennoch wichtig für ein sicheres System ist, war die Absicherung unserer Datenbank. Hier wurde den Standard-Absicherungen der verwendeten Technologien (JDBC, MySQL) vertraut. Eine kritische Betrachtung bzw. Ausweitung innerhalb dieser Arbeit hätte den Rahmen überschritten und den Fokus vom im Mittelpunkt stehenden ISAC-Zertifikat weggelenkt.

Ein weiterer Punkt, der in Zukunft eine Möglichkeit zur Erhöhung der Sicherheit bietet, wäre eine Nachbesserung bei der Verteilung der eSprint-Client-Anwendungen. Der derzeitige Stand der Entwicklung sieht eine einfache Versendung eines Download-links zur Anwendung mit E-Mail vor. In dieser Umsetzungsvariante könnte das Szenario einer Man-in-the-Middle-Attacke tragend werden und somit ein probates Mittel zur Erlangung von unberechtigten Zugriffen sein. Dies hätte zur Folge, dass Dritte in Besitz der RSA-Schlüssel kämen und somit die digitale Identität des tatsächlichen Nutzers einnehmen könnten.

Ein aus der Perspektive der aktuellen Forschung interessanter Aspekt zur Erweiterung des eSprint-Systems bezieht sich auf die grundlegende Architektur des gesamten Projektes. Bisher nutzt eSprint eine Client-Server-Architektur, da dies für die Verteilung der Dateien unter den diversen Clients die einzig sinnvolle Variante zum Zeitpunkt des Projektbeginns darstellte. Umsetzungsvarianten der dezentralen, clientbezogenen Dateispeicherung wurden bereits in der Anfangsphase besprochen und verworfen, da dadurch eine dauerhafte Zugriffsmöglichkeit auf die Dateien verlorengehen würde. Wird aber der aktuelle Stand der Technik in den Prozess mit einbezogen, so ergeben sich neue Möglichkeiten. Würde beispielsweise das in Kapitel

3.5 besprochene Blockchain-Konzept mit einer dezentralen Client-Steuerung und einer minimalen Server-Infrastruktur in Verbindung bringen, so würde ein komplett unabhängiges System entstehen. Ein Server würde nur noch eine untergeordnete und vor allem beliebig austauschbare Rolle einnehmen, während die auszutauschenden Dateien jederzeit und überall über die Blockchain abrufbar wären. Darüber hinaus würden die bereits bekannten Vorteile einer Blockchain, wie etwa die Unverfälschbarkeit der gespeicherten Informationen, wirksam werden.

Während der eSprint-Entwicklung beschränkten wir uns, in Hinblick auf unser erdachtes Anwendungsszenario, auf die durchgehende Zugriffskontrolle von textbasierten Dokumenten. Ebenfalls kommerziell interessant könnte sich auch eine Anwendung auf Video und Bilddateien herausstellen, die im Mittelpunkt vieler DRM-Technologien stehen. Deren Daten-Streams könnten per Editor durch den Benutzer entsprechend getaggt und somit für andere Nutzer in der Sichtbarkeit beschränkt werden. Bei einer entsprechend getaggten Videodatei beispielsweise könnten nicht einsehbare Abschnitte einzeln verschlüsselt und durch einen Platzhalter temporär ersetzt werden. Die ersten 30 Sekunden von Musikdateien könnten etwa als Hörprobe dienen. Ein Schlüssel für den Rest könnte nach Kauf bei einem Provider innerhalb eines ISAC-Zertifikates hinzugefügt werden.

Abschließend könnte noch die weitere Steigerung der internen lokalen Sicherheit erörtert werden. In der Welt des *Trusted Computing* existiert bereits seit längerem die Möglichkeit des Einsatzes eines sogenannten *Trusted Platform Module*. Diese Technologie – es handelt sich dabei um ein Hardware-Modul, das sich eindeutig genau einem PC mit einer spezifischen und unveränderbaren Hardware-Konstellation zuordnen lässt – würde eine noch höhere Sicherheit in Bezug auf Speicher-Leaks und Man-in-the-Middle-Attacken garantieren. Letzteres könnte hierüber komplett ausgeschlossen werden. Hierbei würde aber unter Umständen eine gewisse Hürde bei der Verteilung und Verwendung des eSprint-Systems geschaffen werden.

14 Zusammenfassung

Das Ziel dieser Arbeit war es, mithilfe moderner technischer Hilfsmittel den Datenaustausch auf ein höchstmögliches Niveau zu heben. Die hierfür notwendige

intensive Auseinandersetzung mit verschiedenen Security-Mechanismen war für uns einer der spannendsten Aspekte und verhalf uns zu einem tieferen Verständnis in Bezug auf IT-Security. Im Zuge dessen wurden von uns zwei der gebräuchlichsten Verschlüsselungsalgorithmen behandelt, nämlich das in Kapitel 7.4 bearbeitete x.509 Zertifikat, mit dessen Hilfe sich zertifikatsgesicherte SSL-Verbindungen etablieren lassen, aber auch die in Kapitel 5.1 besprochene AES-Verschlüsselung, mit deren Hilfe sich, bei entsprechender Schlüssellänge, sichere Dateiverschlüsselungen erstellen lassen. Gerade der Einsatz und die Adaptierung dieser Technologien zur Anpassung an unsere Anforderungen war eine der interessantesten Herausforderungen dieser Arbeit. Wie in Kapitel 1.2 näher ausgeführt, war es zu Beginn der Masterarbeit nicht klar, ob eine technische Realisierung des erdachten Systems durchführbar ist. Nicht zuletzt waren es die Gespräche mit Frau Nisreen Alam Aldeen – die Idee einer Weiterentwicklung der bereits von der Cardiff University durchgeführten Konzeption eines solchen Datenaustauschsystems namens SPIDER stammte von ihr– in denen wir gemeinsam Problemlösungen erarbeitet und später im Team umgesetzt haben. Die zur Umsetzung eingesetzten Technologien Java, JavaScript, JavaFX und HTML sind in unseren Augen, damals wie heute, die geeignetsten Programmiersprachen, um ein derart interdisziplinäres Web- und Security-Projekt zu realisieren.

Nicht immer waren die technischen Herausforderungen im Zuge dieser Arbeit leicht zu meistern. Gerade das Kernstück, die Continous-Access-Control mittels der Multiple Keys und der De-Perimeterisation, stellte einen ‚Meilenstein' in unserer Umsetzung dar. Das von Frau Alam Aldeen erdachte System zum Schlüsselmanagement mittels Zertifikatserweiterungen wurde zuvor noch nie umgesetzt und stellte deshalb technisches Neuland für uns dar. Aber auch die Umsetzung der De-Perimeterisation stellte uns vor Herausforderungen: Das hierfür notwendige Zusammenspiel aus Java, JavaFX (WebViews), JavaScript und HTML entwickelte sich als Folge verschiedener aufeinander aufbauender Lösungsansätze, wie sie im Kapitel 13.1 gezeigt und besprochen wurden.

Abbildungsverzeichnis

Abbildung 1: einstweilige Verfügung – Teil 1 14
Abbildung 2: Einstweilige Verfügung – Teil 2 15
Abbildung 3: Anwendungsfall – Übersicht 16
Abbildung 4: Anwendungsfall – einstweilige Verfügung erstellen 17
Abbildung 5: Anwendungsfall – einstweilige Verfügung schreiben 18
Abbildung 6: Anwendungsfall – Dokument öffnen 18
Abbildung 7: Gruppenrechte – eSprint-System 21
Abbildung 8: eSprint-Ansicht Richter und Rechtsanwalt Antragstellerin 22
Abbildung 9: eSprint-Ansicht (aller) Sekretärinnen und Sekretäre 23
Abbildung 10: eSprint-Ansicht Rechtsanwalt Antragsgegner 24
Abbildung 11: eSprint-Ansicht Konzipient 25
Abbildung 12: ccREL-Textpassage mit Klassifikations-Tag 30
Abbildung 13: Architektur des SPIDER-Projekts 32
Abbildung 14: eSprint-3-Tier-Architektur (Überblick) 36
Abbildung 15: Webservice-URI – allgemeines Beispiel 37
Abbildung 16: Simples Modell der symmetrischen Verschlüsselung 51
Abbildung 17: Feistel-Grundstruktur des DES-Algorithmus 52
Abbildung 18: AddRoundKey 54
Abbildung 19: SubBytes – Substitutionstabelle 55
Abbildung 20: ShiftRows 55
Abbildung 21: MixColumns 56
Abbildung 22: KeySchedule – Berechnung der ersten Spalte 57
Abbildung 23: Stromchiffre 58
Abbildung 24: Blockdiagramm eines Secure-Hash-Algorithmus h=H (M) 59
Abbildung 25: Public-Key-Nachrichten-Authentifizierung 60
Abbildung 26: Verschlüsselung mit Public Key 63
Abbildung 27: Verschlüsselung mit Private Key 63
Abbildung 28: SSL-Protocol-Stack 67
Abbildung 29: SSL-Record-Protocol-Operation 69
Abbildung 30: Handshake-Protocol-Action 71

Abbildung 31: Digital Envelope Übersicht 81
Abbildung 32: Verwendung eines Public-Key-Zertifikats 85
Abbildung 33: Struktur eines x.509-Standardzertifikates 86
Abbildung 34: Beispiel x.509-Zertifikat 88
Abbildung 35: Beispiel für ein Certificate Signing Request 89
Abbildung 36: Multiple-Keys-Verschlüsselung 91
Abbildung 37: Ansicht einer Beispieldatei durch Gruppe 1 93
Abbildung 38: Ausschnitt MVC – Layer-Security 95
Abbildung 39: Einsatzbereiche von ISAC 97
Abbildung 40: ISAC-Zertifikat – Detaildarstellung der Erweiterung 99
Abbildung 41: Ablaufdiagramm Ressourcenentschlüsselung mittels ISAC 101
Abbildung 42: Architekturübersicht 105
Abbildung 43: JDBC-Verbindung 110
Abbildung 44: Client-Server-Kommunikation 111
Abbildung 45: Komponentendiagramm – Server 113
Abbildung 46: Komponentendiagramm – Client 115
Abbildung 47: Übersicht – Prozesslandschaft 117
Abbildung 48: Ablaufdiagramm – neue Datei speichern 119
Abbildung 49: Ablaufdiagramm – Datei laden 123
Abbildung 50: Ablaufdiagramm – Speichern einer bestehenden Datei 127
Abbildung 51: Ablaufdiagramm – User erstellen 130
Abbildung 52: Ablaufdiagramm – Rechte vergeben 135
Abbildung 53: Klassendiagramm Übersicht – Server 140
Abbildung 54: Klassendiagramm Übersicht – Client 143
Abbildung 55: Datenbankmodell 146
Abbildung 56: Datentransfer Content-Struktur 150
Abbildung 57: Schnittstellendefinition 151
Abbildung 58: NetBeans IDE 164
Abbildung 59: JavaFX Scene Builder 165
Abbildung 60: Java FXML-Benutzeroberfläche (Frontend) 166
Abbildung 61: Ausschnitt – Methode: loadContent(..) 168
Abbildung 62: Ausschnitt – Methode: request_Isac(..) 170

Abbildung 63: Ausschnitt – Methode: csr_generator(..) 172
Abbildung 64: Ausschnitt – Methode: getISAC(..) 174
Abbildung 65: Ausschnitt IsacResource.java 176
Abbildung 66: Ausschnitt CryptImpl.java 177
Abbildung 67: Ausschnitt IsacImpl.java 178
Abbildung 68: Ausschnitt CertGenerator.java Teil 1 180
Abbildung 69: Ausschnitt CertGenerator.java Teil 2 181
Abbildung 70: Überblick – eSprint-Client 183
Abbildung 71: Überblick Bedienelemente – eSprint-Client 183
Abbildung 72: Ladeprozess starten 186
Abbildung 73: Dateiübersicht im Filebrowser 186
Abbildung 74: Gruppenübersicht im Policy-Editor 188
Abbildung 75: Erstellung einer Gruppe – Eigenschaften 189
Abbildung 76: Erstellung einer Gruppe – Mitglieder 190
Abbildung 77: Gruppenübersicht 191
Abbildung 78: SPIDER-Projekt – Konzept 194
Abbildung 79: eSprint Client – erster Entwurf 195
Abbildung 80: eSprint-Client – zweiter Entwurf 195
Abbildung 81: eSprint.Client – finales Erscheinungsbild 196

Tabellenverzeichnis

Tabelle 1: Exemplarische Übersicht zu Gruppenrechten 20
Tabelle 2: Vergleichsmatrix behandelter Methoden [ARE13] 34
Tabelle 3: Beispiel eines POST-Requests und der zugehörigen Response 41
Tabelle 4: Public-Key-Applikationen 65
Tabelle 5: Inhalt eines Certificate Signing Requests 89
Tabelle 6: Exemplarische Übersicht zu Gruppenrechten 93
Tabelle 7: E-Mail mit Downloadlink zum Client 133
Tabelle 8: Komponentenbeschreibung Implementierung Server 142
Tabelle 9: Komponentenbeschreibung Implementierung Client 145
Tabelle 10: Schnittstellendefinition getisac/save 152
Tabelle 11: Schnittstellendefinition getisac/load 153
Tabelle 12: Schnittstellendefinition getisac/getgroupname 154
Tabelle 13: Schnittstellendefinition getisac/getgroups 154
Tabelle 14: Schnittstellendefinition getisac/getgroupsinfo 155
Tabelle 15: Schnittstellendefinition getisac/getownrights 155
Tabelle 16: Schnittstellendefinition getisac/creategroup 156
Tabelle 17: Schnittstellendefinition getisac/getusers 156
Tabelle 18: Schnittstellendefinition getisac/assignuserstogroup 157
Tabelle 19: Schnittstellendefinition getisac/createusers 158
Tabelle 20: Schnittstellendefinition getisac/getgroupmembers 158
Tabelle 21: Schnittstellendefinition getisac/getowner 159
Tabelle 22: Schnittstellendefinition /load 160
Tabelle 23: Schnittstellendefinition load/filechooser 160
Tabelle 24: Schnittstellendefinition load/keepalive 161
Tabelle 25: Schnittstellendefinition load/sendkill 161
Tabelle 26: Schnittstellendefinition /store 162
Tabelle 27: Schnittstellendefinition store/projects 163
Tabelle 28: Bedienelemente eSprint-Client 184
Tabelle 29: Tastaturkürzel – eSprint-Client 185

Quellenverzeichnis

[AAL08] H. Abelson, B. Adida, M. Linksvayer, N. Yergler, ccREL: The Creative Commons Rights Expression Language, Creative Commons, USA, 2008

[ARE13] N. A. Aldeen, Enhancing Privacy Protection in Distributed Environments through Identification and Authentication-Based Secure Data-Level Access Control, Asia Ares Conference (AReS 2013), Universität Wien, 2013

[BC01] E. Bertino, S. Castano, On Specifying Security Policies for Web Documents with an XML-based Language, In Proceedings of the sixth ACM symposium on Access control models and technologies (SACMAT 2001), Pages 57-65, USA, 2001

[BCC14] BouncyCastle Documentation, The Cryptoworkshop Guide to Java Cryptography and the Bouncy Castle APIs, https://www.bouncycastle.org, Geprüft 2018

[BG08] W. Buhse, D. Günnewig, Digital Rights Management, In Ökonomie der Musikindustrie, Pages 225-235, Gabler, Deutschland, 2008

[BH09] P. Burnap, J. Hilton, Self Protecting Data for De-perimeterised Information Sharing, Cardiff School of Computer Science, Cardiff University, Third International Conference on Digital Society, UK, 2009

[BHT09] P. Burnap, J. Hilton, A. Tawileh, Self-Protecting Information for De-perimeterised Electronic Relationships (SPIDER), Final Report, In Conference: Digital Society (ICDS '09), UK, 2009

[BIT14] I. Eyal, E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, In Financial Cryptography and Data Security, Springer Berlin Heidelberg, Deutschland, 2014

[BIT15] E. Heilman, A. Kendler, A. Zohar, S. Goldberg, Eclipse attacks on bitcoin's peer-to-peer network. In USENIX Security, USA, 2015

[BLBC+02] T. Berners-Lee, T. Bray, D. Connolly, P. Cotton, R. Fielding, C. Lilley, D. Orchard, N. Walsh, S. Williams, Architectural principles of the world wide web (2002), World Wide Web Consortium, http://www.w3.org/2001/tag/2002/0828-archdoc, Geprüft 2018

[BLFM05] T. Berners-Lee, R. Fielding, L. Masinter, Uniform resource identifier (uri): Generic syntax, The Internet Society (2005), http://gbiv.com/protocols/uri/rfc/rfc3986.html, Geprüft 2018

[BLK16] A. Kosba, A. Miller, E. Shi, Z. Wen, Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts, In IEEE Symposium on Security and Privacy (SP2016), USA, 2016

[CD+02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI, In IEEE Internet computing (IC2002), Pages 86-93, USA, 2002

[CD11] C. Dea, JavaFX 2.0 - Introduction by Example, Apress Verlag, USA, 2011

[CK98] C. Kennedy, Management Gurus - 40 Vordenker und ihre Ideen, Springer Verlag, Deutschland, 1998

[CSR00] M. Nystrom, B. Kaliski, PKCS #10: Certification Request Syntax Specification Version 1.7, RSA Security Network Working Group, USA, 2000

[CW09] A. Van Cleeff, R. J. Wieringa, Rethinking de-perimeterisation: Problem analysis and solutions. In Proceedings of the IADIS International Conference Information Systems 2009, IADIS press, Spanien, 2009

[DBR16] DB-Engines Ranking, http://db-engines.com/de/ranking, Geprüft 2018

[DIH76] W. Diffie, M. Hellman, New Directions in Cryptograph. In Proceedings of the AFIPS National Computer Conference, USA, 1976

[DRM06] T. Grimes, K. Mai, U.S. Patent No. 7,036,011, Washington, DC: U.S. Patent and Trademark Office, USA, 2006

[DVS02] E. Damiana, S. De Cabitani Di Vimercati, S. Paraboschi, P. Samarati, A Fine-Grained Access Control System for XML Documents, In ACM Transactions on Information and System Security, Vol. 5/No.2, Pages 169–202, USA, 2002

[EPP11] N.A. Aldeen, G. Quirchmayr, Enhancing Privacy Protection in Distributed Environments through Identification and Authentication-Based Secure Data-Level Access Control, Springer Verlag, Österreich, 2011

[FI08] M. Fischer, Advanced Encryption Standard (2008), http://w6hde4t7p.homepage.t-online.de/03_Kryptologie/KryptDateien/Kapitel%2008_AES.pdf, Geprüft 2018

[FLD00] R. Fielding, Architectual styles and the design of network-based software architectures (2000). PhD Dissertation, http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm, Geprüft 2018

[GHM+00] M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau, H. Frystyk, A. Karmarkar, Y. Lafon, Soap version 1.2 part 1: Messaging framework (second edition)(2000), World Wide Web Consortium, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, Geprüft 2018

[HC17] W. Stallings, Introduction to Cryptography, http://williamstallings.com/Crypt-Tut/Crypto%20Tutorial%20-%20JERIC.html, Geprüft 2018

[JCR10] H. Schildt, Java: The complete Reference, Tenth Edition, USA, 2017

[KR01] B. Krishnamurthy, J. Rexford, Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement, Addison-Wesley Longman Publishing Co., USA, 2001

[LEU94] K. Leutwyler, Superhack, Scientific American, USA, 1994

[MA03] W. Mao, Modern cryptography: theory and practice, Prentice Hall, 1 edition, USA, 2003

[MBU14] M. Budny, Sending binary data along with a REST API request, http://blog.marcinbudny.com/2014/02/sending-binary-data-along-with-rest-api.html, Geprüft 2018

[MIY06] M. I. Yagüe, Survey on XML-Based, Policy Languages for Open Environments, Computer Science Department, In Journal of Information Assurance and Security, University of Málaga, Spanien, 2006

[ML07] N. Mitra, Y. Lafon, Soap version 1.2 part 0, Primer (second edition), World Wide Web Consortium, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, Geprüft 2018

[NAI00] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, A national-scale authentication infrastructure, In IEEE Computer Volume 33 Issue 12, Pages 60-66, USA, 2000

[NET18] Oracle Corporation, Netbeans Platform, https://netbeans.org/, Geprüft 2018

[ORA13] Oracle Corporation, Why, Where, and How JavaFX Makes Sense, http://www.oracle.com/technetwork/articles/java/casa-1919152.html, Geprüft 2018

[PG05] G. Palmer, De-perimeterisation: Benefits and limitations, In Information Security Technical Report Volume 10 Issue 4, Pages 189-203, Elsevier Advanced Technology Publications, UK, 2005

[PZ+08] C. Pautasso, O. Zimmermann, F. Leymann, Restful web services vs. big web services: making the right architectural decision, In 17th international conference on World Wide Web, Pages 805-814, ACM, China, 2008

[RFC2616] RFC2616, World Wide Web Consortium, Hypertext Transfer Protocol - HTTP/1.1, http://www.w3.org/Protocols/rfc2616/rfc2616, Geprüft 2018

[RFC5280] RFC5280,Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, https://www.ietf.org/rfc/rfc5280.txt, Geprüft 2018

[RFC822] RFC822, STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES, https://tools.ietf.org/html/rfc822, Geprüft 2018

[RH18] D. Rehbein, Hashwert-Funktionen MD5 und SHA-1, http://www.daniel-rehbein.de/hashwerte.html, Geprüft 2018

[RI14] I. Ristić, Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications, Feisty Duck, UK, 2004

[RO08] A. Rodriguez, Restful web services: The basics, IBM developerWorks, Page 33, USA, 2008

[RR07] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly, USA, 2007

[RSA78] R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, In Magazine Communications of the ACM, Volume 21 Issue 2, Pages 120-126, USA, 1978

[SA10] S. Allamaraju, RESTful Web Services Cookbook, O'Reilly, USA, 2010

[SB12] W. Stallings, L. Brown, Computer security: principles and practice, Pearson, USA, 2012

[SC10] J. Schwenk, Sicherheit und Kryptographie im Internet: von sicherer E-Mail bis zu IP-Verschlüsselung, Springer Verlag, Deutschland, 2010

[SOW3C] World Wide Web Consortium, http://www.w3.org/TR/soap/, Geprüft 2018

[ST05] D. R. Stinson, Cryptography: theory and practice, CRC press, USA, 2005

[ST13] W. Stallings, Cryptography and Network Security: Principles and Practice, Pearson, USA, 2013

[ST16] W. Stallings, Network Security Essentials: Applications and Standards, Pearson, USA, 2016

[TOL07] T. Olovsson, The Road to Jericho and the myth of Fortress Applications, AppGate Network Security, Schweden, 2007

[VON07] H. Vonhoegen, Einstieg in XML 8. Auflage, Galileo Computing, Deutschland, 2015

[WC+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson, Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more. Prentice Hall PTR, USA, 2005

[WIV12] J. Weaver, S. Chin, D. Iverson, J. Vos, Pro JavaFX 2 - A Definitive Guide to Rich Client with Java Technology, Apress Verlag, USA, 2012

[WM08] M. E. Whitman, H. J. Mattord, Management of Information Security, 2nd edition, USA, 2008

Anhang

Abstract (Deutsch)

Das steigende Bewusstsein für einen verantwortungsvollen Umgang mit der Sicherheit beim Austausch von digitalen Informationen macht die Neu- und Weiterentwicklung von Sicherheitsstandards und Werkzeugen notwendig. In der heutigen Zeit kann ein zu lockerer Umgang mit sensiblen personenbezogenen Daten bereits als Verletzung der DSGVO geahndet werden.[18] Immer wieder kommt und kam es in den vergangenen Jahren zu Mängeln in Informationssystemen, durch die teils große Mengen an privaten Nutzerdaten in die Hände Dritter gelangten.[19][20] Vor diesem Hintergrund entstand die Idee von Frau Nisreen Alam Aldeen, ein Werkzeug zum de-perimeterisierten Datenaustausch zu erdenken und entwickeln. Die De-Perimeterisierung ist der Schutz einer Ressource unter Zuhilfenahme von Verschlüsselungsmechanismen über Systemgrenzen hinweg.[21]

Die nachfolgende Arbeit beschäftigt sich zunächst mit dem Themengebiet der Sicherheit in Informationssystemen und stellt ein auf diesen theoretischen Grundlagen und aktuellen Technologien basierendes, neu entwickeltes Werkzeug zum sicheren Informationsaustausch vor. Bei dieser Anwendung handelt es sich um das sogenannte Enhanced-self-protecting-Information-Technology-System oder auch kurz eSprint-System. Konkret stellt eSprint dem Benutzer einen Texteditor mit grafischer Oberfläche und erweiterten Sicherheitsmechanismen, aufbauend auf den Verschlüsselungstechnologien RSA (Entwickelt von Rivest, Shamir und Adleman) und AES (Advanced Encryption Standard), zur Verfügung. Die Kernthemen dieser Arbeit sind die Vorstellung und der Einsatz einer eigenen Erweiterung des x.509-Standards. Als zentrales Kommunikationselement ermöglicht es dem Benutzer, ein de-perimeterisieres Dokument mit individueller und kontinuierlicher Zugangskontrolle zu entwerfen, das zu jeder Zeit durch den Ersteller der Datei modifiziert werden kann.

[18] EU Datenschutz Grundverordnung, https://www.jusline.at/gesetz/dsgvo, geprüft 2018
[19] http://www.faz.net/-ikh-98sfv, geprüft 2018
[20] http://www.faz.net/-gqi-92fgh, geprüft 2018
[21] Siehe Kapitel 3.2

Das Ziel dieser Arbeit ist es, die technische Umsetzbarkeit der theoretischen Überlegungen zu zeigen und einen funktionsfähigen Prototyp zu entwickeln.

Schlüsselwörter – De-Perimeterisation / RSA / AES / Digital Envelope / eSprint / x.509 / Continous Access Control / Multiple Keys / ISAC Zertifikat / JavaFX

Abstract (Englisch)

The increasing awareness for responsible usage of security measures while exchanging digital information, leads to new and further development of security standards and tools. In this day and age loose handling of sensitive personal data could be judged as an infringement of the GDPR (in german: DSGVO) and will be punished.[18] In the past few years, flaws inside information systems, again and again led to large amounts of private user data being accessed by unauthorized third parties.[19,20]

Due to this fact, the idea to create and develop a de-perimeterized data exchange tool was born by Mrs Nisreen Alam Aldeen. De-perimeterization is the protection of a resource within the aid of encryption mechanisms across system boundaries.[21]

The following work first deals with the subject area of security in information systems and proposes a newly developed tool for a secure exchange of information based on theoretical principles and current technologies. Our proposed and developed application is the so-called Enhanced self-protecting information technology system or short eSprint. It provides a text editor with a graphical interface and advanced security mechanisms. eSprint is based on the encryption technologies RSA (developed by Rivest, Shamir and Adleman) and AES (Advanced Encryption Standard). The core themes of this work are the definition and the usage of an extension of the x.509 standard. As a central communication element the proposed extension enables the user to create a de-perimeterized document with individual and continuous access control, which can be adapted and modified by the creator at any time. The aim of this work is to prove the technical implementability of the theoretical considerations and to develop a working prototype.

Keywords – De-Perimeterization, RSA, AES, Digital Envelope, eSprint, x.509, Continous Access Control, Multiple Keys, ISAC Certificate, JavaFX

Arbeitsteilung

1 EINFÜHRUNG ..... POLLAK/WESENAUER

2 ANWENDUNGSFALL ..... POLLAK/WESENAUER

2.1 SZENARIO ..... POLLAK

2.2 AKTEURE ..... WESENAUER

2.3 BEISPIEL FÜR EINE EINSTWEILIGE VERFÜGUNG ..... POLLAK

2.4 USE-CASE-DIAGRAMM ..... WESENAUER

2.5 ABLAUF EINES TYPISCHEN ARBEITSGANGES ..... POLLAK

2.6 EXEMPLARISCHE DARSTELLUNG ..... WESENAUER

3 STATE OF THE ART UND HERAUSFORDERUNGEN ..... POLLAK/WESENAUER

3.1 SYSTEM-LEVEL-ACCESS-CONTROL ..... POLLAK

3.2 DE-PERIMETERISATION ..... POLLAK

3.3 BESTEHENDE DE-PERIMETERISATION-PROTECTION-ANSÄTZE ..... POLLAK/WESENAUER

3.3.1 DIGITAL-RIGHTS-MANAGEMENT ..... WESENAUER

3.3.2 CREATIVE-COMMONS-RIGHTS-EXPRESSION-LANGUAGE ..... POLLAK

3.3.3 DAS SPIDER-PROJEKT ..... WESENAUER

3.4 VERGLEICHSMATRIX ..... WESENAUER

3.5 ABGRENZUNG ZU BLOCKCHAINS ..... WESENAUER

4 IMPLEMENTIERUNGSUMFELD ..... POLLAK

5 SECURITY-MANAGEMENT ..... WESENAUER

6 SYSTEMKONZEPTION ..... POLLAK/WESENAUER

6.1 GRUNDLEGENDE IDEE ..... POLLAK/WESENAUER

6.2 CLIENT ..... POLLAK

6.3 SERVICE ..... WESENAUER

6.4 DATENHALTUNG ..... WESENAUER

6.5 SICHERHEITSMAßNAHMEN ..... WESENAUER

7 IMPLEMENTIERUNGSANSATZ ..... POLLAK/WESENAUER

7.1 JAVAFX ..... POLLAK
7.2 DIGITAL ENVELOPE ..... WESENAUER
7.3 DE-PERIMETERISATION ..... POLLAK
7.4 SECURITY-MANAGEMENT UND CONTINOUS-ACCESS-CONTROL ..... POLLAK/WESENAUER
7.4.1 IDENTITÄTSMANAGEMENT – WAS IST X.509? ..... WESENAUER
7.4.2 MULTIPLE-KEY-MANAGEMENT MITTELS EXTENDED X.509 ..... POLLAK
7.4.3 GRUPPENRICHTLINIEN UND GÜLTIGKEITSRAHMEN ..... POLLAK
7.4.4 SINGLE-SESSION-SECURITY ..... POLLAK
7.4.5 LAYER-SECURITY ..... POLLAK
7.5 DAS ISAC-ZERTIFIKAT ..... WESENAUER
7.6 KONKRETER UMSETZUNGSANSATZ ..... POLLAK

8 SYSTEMARCHITEKTUR UND -KOMPONENTEN ..... POLLAK/WESENAUER

8.1 ARCHITEKTUR ..... POLLAK/WESENAUER
8.1.1 CLIENT ..... POLLAK
8.1.2 SERVER ..... WESENAUER
8.1.3 DATABASE ..... WESENAUER
8.1.4 DATENTRANSFER ..... POLLAK
8.2 SYSTEMKOMPONENTEN ..... POLLAK/WESENAUER
8.2.1 SERVER ..... WESENAUER
8.2.2 CLIENT ..... POLLAK

9 PROZESSBESCHREIBUNG ..... POLLAK/WESENAUER

10 REALISIERUNG ..... POLLAK/WESENAUER

10.1 KLASSENDIAGRAMM ÜBERBLICK ..... POLLAK/WESENAUER
10.1.1 SERVER ..... WESENAUER
10.1.2 CLIENT ..... POLLAK
10.2 DATENMODELL ..... WESENAUER
10.3 SCHNITTSTELLENDEFINITION UND –BESCHREIBUNG ..... POLLAK/WESENAUER

11 IMPLEMENTIERUNGSBESCHREIBUNG ..... POLLAK/WESENAUER

11.1 ENTWICKLUNGSUMGEBUNG ..... WESENAUER

11.2 IMPLEMENTIERUNG DES LAYOUTS.....POLLAK
11.3 IMPLEMENTIERUNG DER ZERTIFIKATSGENERIERUNG.... POLLAK/WESENAUER
11.3.1 CLIENT – ISAC REQUEST ABSETZEN (.CSR) .....POLLAK
11.3.2 SERVER – ISAC-ZERTIFIKAT FÜR USER/FILE KREIEREN..... WESENAUER

12 INTERFACEBESCHREIBUNG..... POLLAK

13 EINORDNUNG UND DISKUSSION DER ERGEBNISSE.....POLLAK/WESENAUER

14 ZUSAMMENFASSUNG .....POLLAK/WESENAUER