Hochschule
Augsburg University of
Applied Sciences

Fakultät für Informatik

Bachelorarbeit

Studienrichtung
Wirtschaftsinformatik (B.Sc.)

Michael Fürmann

Gesicherte Ende-zu-Ende-Kommunikation in Webapplikationen

Prüfer: Dipl.-Inf. (FH) Dipl.-Designer (FH)
Erich Seifert, MA
Abgabe der Arbeit am: 20.06.2017

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon +49 821 55 86-3450
Fax +49 821 55 86-3499

Verfasser der Bachelorarbeit:
Michael Fürmann
Jahnstr. 10
86316 Friedberg
Telefon +49 821 604901
michael.fuermann@hs-augsburg.de

Danksagung

Ich möchte an dieser Stelle all denjenigen danken, die mich bei der Erstellung dieser Arbeit unterstützt und damit zu deren Gelingen beigetragen haben.

Zuerst sind hier mein Betreuer, Herr Dipl.-Inf. (FH) Dipl.-Designer (FH) Erich Seifert, sowie mein Zweitprüfer Herr Prof. Dr.-Ing. Torsten Schöler zu nennen. Beiden möchte ich für die gute Betreuung und die anregenden und hilfreichen Diskussionen danken.

Des weiteren danke ich meiner Familie für ihre Geduld und Unterstützung in den letzten Monaten.

Abschließend möchte ich meiner Frau einen besonderen Dank aussprechen. Sie hat mich nicht nur im Studium und bei der Erstellung dieser Arbeit großartig unterstützt, sondern auch grundlegend zur Motivation zu diesem Studium beigetragen.

Gesicherte Ende-zu-Ende-Kommunikation in Webapplikationen

Michael Fürmann

Augsburg, 20. Juni 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Kurzzusammenfassung

In der Fachliteratur und den gängigen Medien ist in den letzten Jahren ein wachsendes Interesse am Thema der Informationssicherheit zu erkennen. Immer öfter ist von Attacken durch Behörden anderer Staaten oder nichtstaatliche Gruppierungen auf Informationssysteme zu lesen. Das Ziel der Angreifer ist dabei oft das Entwenden von sensiblen Personendaten, Geschäftsgeheimnissen oder Zahlungsinformationen und Zugangsdaten. Oft werden Forderungen nach einer wirkungsvollen Sicherung schützenswerter Informationen gegen unbefugte Zugriffe laut. Diese sind nach aktueller Gesetzeslage sogar rechtlich begründet, womit der Informationsschutz im Grunde eingefordert werden kann.

Gerade Fachartikel für die Zielgruppe der Softwareentwickler richten den Fokus in den meisten Fällen auf die Transportsicherung der Informationen. Eine vollständige Sicherung der Daten, um diese auch vor Zugriffen des Applikationsbetreibers oder unberechtigter Dritter zu schützen, bleibt dabei unbeachtet. Auf dem Markt gibt es dennoch bereits einige Anbieter, die ein vorbildliches Konzept zur kryptografischen Sicherung sensibler Kommunikationsdaten vorzuweisen haben. Die Implementierung einer wirkungsvollen Informationsverschlüsselung in einer eigenen Anwendung ist dagegen sehr komplex. Durch gründliche Planung ist dieses Ziel dennoch gut zu erreichen.

1. Einleitung

Die Sicherheit von Informationen in IT-Systemen gewinnt seit Jahren zunehmend an Bedeutung. Die Frequenz von Berichterstattungen zu Passwortdiebstählen, gekaperten Zugängen und entwendeten Informationen steigt gefühlt immer schneller an. Als Gegenmaßnahme war in den letzten Jahren die Transportverschlüsselung, also der Schutz von Daten auf dem Weg in das Rechenzentrum des Anbieters, das große Thema. Diese bietet einen guten Schutz, um das Abhören einzelner sensibler Daten auf dem Übertragungsweg zu verhindern. Wirkungslos ist sie hingegen, um eine unerwünschte oder unberechtigte Verarbeitung der Informationen durch den Anbieter zu unterbinden, sei es der Verkauf an Drittpartner oder die Bildung von Profilen. Auch gegen den Zugriff staatlicher oder nichtstaatlicher Organisationen direkt auf den Serversystemen sind die Informationen nicht geschützt.

Um diesen Schutz zu erreichen, bedarf es einer Verschlüsselung der Informationen noch vor der Übertragung, im Falle einer Webanwendung also noch im Browser des Benutzers. Durch ein gutes Angebot stabiler Cryptobibliotheken in der Programmiersprache JavaScript sowie eine Empfehlung des World Wide Web Consortium (W3C)[1] ist dies heute gut zu bewerkstelligen. Allerdings bedarf es einiger Vorkenntnisse und einer guten Planung. Im Vergleich zu Transportverschlüsselung gibt es bisher jedoch nur wenig unterstützende Literatur. Auch eine Suche in der beliebten Hilfecommunity StackOverflow[2] bescheinigt ein eher geringes Interesse an clientseitiger Verschlüsselung. So finden sich zum Zeitpunkt der Erstellung dieser Arbeit 1.560 Einträge zu den Tags „JavaScript“ in Kombination mit „Cryptograpy“ oder „Encryption“, während zu „TLS“ 30.764 Suchergebnisse gelistet werden. Alleine das noch sehr junge und anbieterbezogene Thema LetsEncrypt bringt es mit 491 Treffern bereits auf ein knappes Drittel der allgemeinen Anfragen zu browserseitiger Kryptografie.

Einen interessanten Ansatz zu Informationssicherheit nach dem Zero-Knowledge Prinzip, der Doktrin, dass der Anbieter zu keinem Zeitpunkt Wissen über die gespeicherten Informationen erlangen kann, bietet die Forschungsplattform Mylar des Massachusetts

[1] `https://www.w3.org/TR/WebCryptoAPI/`
[2] `https://www.stackoverflow.com`

Institute of Technology (Popa u. a. 2016). Das siebenköpfige Team skizziert in seiner Forschungsarbeit ein ausgefeiltes kryptografisches System, um Informationen wirkungsvoll vor unbefugten Zugriffen zu schützen. Die auf dieser Grundlage entstandene Bibliothek mag für manche Projekte geeignet sein. Es ist dennoch zu vermuten, dass sie für die meisten Anwendungen entweder zu umfangreich ist oder sich aufgrund von im Verlauf dieser Arbeit erörterten Nachteilen als ungeeignet erweist. Soll in einer zu erstellenden Applikation die Vertraulichkeit der Informationen kryptografisch gesichert werden, stellt sich die Frage, ob dies auch selbst geplant und implementiert werden kann. Mit der Klärung dieser Fragestellung beschäftigt sich die vorliegende Arbeit.

Im Rahmen dieser Ausarbeitung sollen zunächst in Kapitel 2 Gründe dargelegt werden, die den erhöhten Aufwand bei der Entwicklung einer Webapplikation mit abgesicherter Kommunikation rechtfertigen oder sogar deren Notwendigkeit aufzeigen. In Kapitel 3 wird erörtert, wie zu verschlüsselnde Daten der Applikation identifiziert werden und welche Möglichkeiten zur Verschlüsselung in einem Client-Server-System bestehen. Nach der Analyse der Sicherungskonzepte einiger populärer Webanwendungen in Kapitel 4, einschließlich des bereits genannten Forschungsprojektes Mylar, erfolgt schließlich in Kapitel 5 eine Ausführung zu der im Rahmen dieser Arbeit entstandenen Implementierung. Dabei handelt es sich um eine Beispielanwendung, welche sich primär auf die Wahrung der Vertraulichkeit der erfassten Informationen konzentriert. In Kapitel 6 werden anschließend die kryptografischen Konzepte der vorgestellten properitären Anwendungen diskutiert und Unterschiede zu der entstandenen Implementierung aufgezeigt. Abschließend werden in Kapitel 7 die Ergebnisse dieser Arbeit zusammengefasst und Anregungen zu weiteren Entwicklungen und Forschungen gegeben.

Die Ausführungen der folgenden Kapitel legen ferner die Annahme zugrunde, dass die Kommunikation der Benutzer über ein offenes Netz erfolgt. Im Gegensatz zu geschlossenen Netzen entzieht sich hierbei der Übertragungsweg zwischen den Informationsverarbeitungssystemen der Kontrolle des Anbieters. Prinzipiell ist es jedem Benutzer möglich, von außerhalb eines kontrollierten Netzwerkes auf die Applikation zuzugreifen. (Meinel und Sack 2009) Die Beispiele der nachfolgenden Kapitel beschreiben zur anschaulichen Darstellung die Kommunikation zweier fiktiver Benutzer mit den Namen Alice und Bob.

2. Gründe für Verschlüsselung

In den meisten Fällen werden Webanwendungen in bereits eingangs beschriebenen offenen Netzen betrieben. Um die Kommunikation der Benutzer wirkungsvoll zu schützen, ist ein erheblicher Mehraufwand in der Konzeption und Implementierung des geplanten Systems vonnöten. Die folgenden Abschnitte erläutern einige Gründe, die diesen Aufwand rechtfertigen oder, je nach Zielgruppe und Anwendungsfall, sogar die Notwendigkeit dieser Maßnahmen aufzeigen.

2.1. Rechtliche Grundlage

Die Datenschutzgesetze der Bundesrepublik Deutschland liefern eine solide rechtliche Grundlage für die Verschlüsselung von Kommunikationsinhalten zum Schutz der enthaltenen Informationen. So werden sämtliche „Einzelangaben über persönliche oder sachliche Verhältnisse einer bestimmten oder bestimmbaren natürlichen Person“ (BDSG Gola u. a. 2015, §3(1)) als besonders schützenswert eingestuft. Auch Informationen, die intime Lebensbereiche eines Menschen berühren, werden hierzu gezählt. So sind etwa Informationen zum gesundheitlichen Zustand, therapeutischen Ergebnissen, Religion oder Sexualität eines Menschen durch dieses Gesetz gleichwertig geschützt. (BDSG ebd., §3(9)) Nach dem Verbotsprinzip wird ein sehr enger Rahmen gesteckt, in dem ein Anbieter personenbezogene Daten erheben und nutzen darf. Auch das Bundesverfassungsgericht kam 2008 zu dem Schluss, dass ein „Grundrecht auf Gewährleistung der Vertraulichkeit und Integrität informationstechnischer Systeme“ schon heute als Teil des Persönlichkeitsrechts anzusehen ist. (BVerfG 2008)

Im Kontrast dazu geben heutzutage viele Nutzer freiwillig große Mengen an personenbezogenen Daten online preis, beispielsweise in sozialen Netzwerken. Dieses Verhalten steht im Widerspruch mit der Auflage an Anbieter, nur die nötigsten Daten zu erfassen (Prinzip der Datenvermeidung). Daher kommt Jochen Schneider zu dem Schluss, dass in der Rechtsprechung ein Paradigmenwechsel vom zuvor genannten Verbotsprinzip hin zu Daten als materiell-rechtlichem Schutzgut notwendig ist. Dies ermöglicht die Ausgestaltung einer freien Kommunikation, in der auch weiterhin eine Zweckbindung der erhobenen Daten besteht. Das Verbotsprinzip wäre seiner Forderung nach lediglich

auf Informationen anzuwenden, die unmittelbar die Privatsphäre eines Individuum betreffen. (Schneider 2013)

Unter Einsatz von Kryptografie mit dem Prinzip der selektiven Sichtbarkeit kann eine Applikation dafür konzipiert werden, große Mengen persönlicher Informationen ihrer Benutzer rechtskonform zu erfassen. (ebd.) Durch geeignete clientseitige Verschlüsselungsverfahren muss sichergestellt werden, dass sensible Daten weder von anderen Nutzern noch vom Anbieter selbst gelesen oder verarbeitet werden können, solange der betreffende Benutzer dies nicht wünscht. Davon auszunehmen sind Informationen, deren Verarbeitung im Rahmen des BDSG §3 beispielsweise zum ordnungsgemäßen Betrieb der Applikation oder zu Abrechnungszwecken gestattet ist. Nach diesem Ansatz werden Informationen, welche über den genannten Rahmen hinausgehen, verschlüsselt im Nutzerauftrag gespeichert. Eine weitergehende Auswertung durch den Anbieter ist nicht möglich. Dadurch können auch Vorschriften zur Datensparsamkeit unter Anwendung des Verbotsprinzips weiterhin erfüllt werden. Es besteht bei sorgfältiger Umsetzung keine Möglichkeit, unbefugt Auswertungen des Datenbestandes durchzuführen. Damit ist die Vertraulichkeit personenbezogenen Daten gewährleistet und somit auch die Forderung der Verfassungsrichter erfüllt.

2.2. Schutzziele der Informationssicherheit

In der Informationssicherheit wird zwischen sechs grundlegenden Schutzzielen unterschieden, an deren Anforderungen sich ein als sicher anzusehendes Informationsverarbeitungssystem orientieren sollte.

Schutzziele der Informationssicherheit

1. **Verfügbarkeit**
   Das System ist zu den benötigten Zeiten (beispielsweise Geschäftszeiten oder rund um die Uhr) zu 100% verfügbar.

2. **Datenintegrität**
   Es ist gewährleistet, dass von einem Benutzer erfasste Informationen auf dem Weg zu einem anderen Benutzer nicht verändert werden können.

3. **Vertraulichkeit und Geheimhaltung**
   Einem unberechtigten Dritten ist es nicht möglich, Einblick in die Kommunikation zwischen anderen Benutzern zu nehmen.

4. **Authentifikation**
   Der Verfasser einer Nachricht kann zweifelsfrei ermittelt werden. Die Identität von Sender und Empfänger ist zuverlässig feststellbar.

5. **Verbindlichkeit**
   Das Versenden und Empfangen einer Nachricht sind belegbar und können nicht abgestritten werden.

6. **Autorisierung**
   Nach Feststellung der Identität eines Benutzers hat dieser nur Zugriff auf Inhalte, die tatsächlich für ihn bestimmt sind.

(Meinel und Sack 2009)

Nicht alle dieser Schutzziele können in Webapplikationen durch den Einsatz von Kryptografie erfüllt werden. Eine hohe Verfügbarkeit beispielsweise kann nur durch Redundanz betriebskritischer Komponenten oder Hochverfügbarkeitscluster im Rechenzentrum erreicht werden. Wenn die Möglichkeiten der Kryptografie korrekt eingesetzt werden, kann damit zumindest dreien der genannten Kernziele Rechnung getragen werden. Vertraulichkeit durch Geheimhaltung von Inhalten, Authentifikation durch Verifizierung der Identität und Integrität der Daten durch Absicherung der Daten bei der Übertragung. (Ristic 2014) Konkrete Möglichkeiten, die Vertraulichkeit der Informationen bereits bei der Konzeptionierung einer Applikation sicherzustellen, zeigt Abschnitt 5.2.

2.3. Nutzervertrauen

Eine Vielzahl von Webapplikationen erfasst eine große Menge persönlicher Informationen ihrer Benutzer. Vielen potentiellen Anwendern fehlt jedoch das Vertrauen, dass ein Anbieter verantwortungsvoll mit ihren sensiblen, erfassten Informationen umgeht. Dies kann ein großes Hindernis für die Nutzung des Dienstes darstellen. (Chen, Paxson und Katz 2010) Im Vergleich mit Applikationen der Konkurrenz kann es somit ein Alleinstellungsmerkmal darstellen, nach dem Prinzip der selektiven Sichtbarkeit (siehe 2.1) sensible Kommunikationsdaten grundsätzlich verschlüsselt abzulegen.

Für den Anbieter kann sich so der höhere Aufwand für die Implementierung strategisch durch das gesteigerte Vertrauen seiner Anwender und Kunden lohnend auswirken.

2.4. Anforderungen der Zielgruppe

Nicht selten gehört die Zielgruppe einer Applikation zu einem besonders sicherheitsbewussten Klientel mit hohen Anforderungen an die Sicherheit der Kommunikationsdaten in einer Applikation. Anwendungsfälle hierfür wären beispielsweise Anwendungen zur Verwaltung von Forschungs- und Entwicklungsunterlagen oder sensibler Unternehmensdaten im Management.

Daneben kann auch die Natur der erfassten Informationen von Gesetzeswegen einen hohen Aufwand in der Absicherung der Informationen erforderlich machen. Daten zum gesundheitlichen Zustand oder therapeutische Informationen, wie sie etwa in E-Health-Applikationen erfasst werden, sind als besonders schützenswert eingestuft (siehe 2.1). Das Recht auf informationelle Selbstbestimmung der betroffenen Personen sollte hier im grundlegenden Konzept der Anwendnung verankert werden. Ein Anwender muss nach eigenem Ermessen entscheiden können, welche Informationen erfasst werden und wem es gestattet ist, darauf zuzugreifen.

Da die aufgeführten Informationen in der Regel auf den Systemen eines Anbieters gespeichert werden, müssen Kunden und Anwender sich auf die Wahrung der Vertraulichkeit ihrer Daten verlassen können. In einem System, welches von mehreren Benutzern verwendet wird, ist dies nur durch Umsetzung eines gut geplanten Kryptosystems erreichbar. (Karlinger, Ettmayer und Schrefl 2011) So behält der Eigentümer einer Information die Kontrolle darüber und entscheidet selbst, wem durch Weitergabe des benötigten Schlüssels ein Zugriff auf seine Informationen ermöglicht wird.

3. Kryptografie in Webanwendungen

Der Schutz von Integrität und Vertraulichkeit der Kommunikationsdaten einer Anwendung kann durch den gut geplanten Einsatz von Kryptografie in der Anwendung

abgesichert werden. Allerdings stehen verschiedene kryptografische Verfahren zur Verfügung, die jeweils Vor- und Nachteile für verschiedene Einsatzgebiete haben. Auch der Ort der Verschlüsselung, im Client- oder Serverteil einer Webapplikation, hat einen starken Einfluss auf die Erfüllung der Schutzziele (vgl. 2.2) sowie auf den Funktionsumfang der Anwendung. Ebenfalls zu entscheiden ist, welche der von den Nutzern eingegebenen Daten verschlüsselt werden sollen und wie in einer Mehrbenutzeranwendung anderen Benutzern der kontrollierte Zugriff auf verschlüsselte Informationen ermöglicht werden kann.

3.1. Kryptografische Systeme

Die Kryptografie beschäftigt sich im Wesentlichen mit dem Ver- und Entschlüsseln von Informationen (Plain Text). Bei der Verschlüsselung werden diese durch Nutzung eines bestimmten Chiffre (Verschlüsselungsverfahren) in Schlüsseltext umgewandelt (Chiffretext). Dazu benötigt das Chiffre eine Schlüsselinformation, die aus einer astronomisch großen Menge möglicher Werte frei gewählt werden kann und damit fast unmöglich zu erraten ist. Auch zum Entschlüsseln wird neben der Kenntnis des verwendeten Chiffre ein passender Schlüssel benötigt, um den Chiffretext wieder in die ursprüngliche, lesbare Information zurückzuwandeln. (Meinel und Sack 2009) Der Einsatz geeigneter Verschlüsselungsverfahren kann dazu beitragen, die Informationen in einer Applikation wirksam vor unbefugtem Zugriff zu schützen.

3.1.1. Symmetrische Kryptografie

Symmetrische Verschlüsselungsverfahren verwenden sowohl für die Ver- als auch für die Entschlüsselung mit dem selben Schlüssel. Nach den Regeln der gewählten Chiffre werden in mehreren Runden die Zeichen der Klartextinformation miteinander vertauscht oder anhand einer Ersetzungstabelle ausgetauscht. Hierbei spielt der gewählte Geheimschlüssel eine essentielle Rolle. Diese vergleichsweise simplen Operationen, welche sowohl beim Ver- als auch beim Entschlüsseln zum Einsatz kommen, machen symmetrische Verfahren sehr performant, auch für größere Datenmengen. (ebd.)

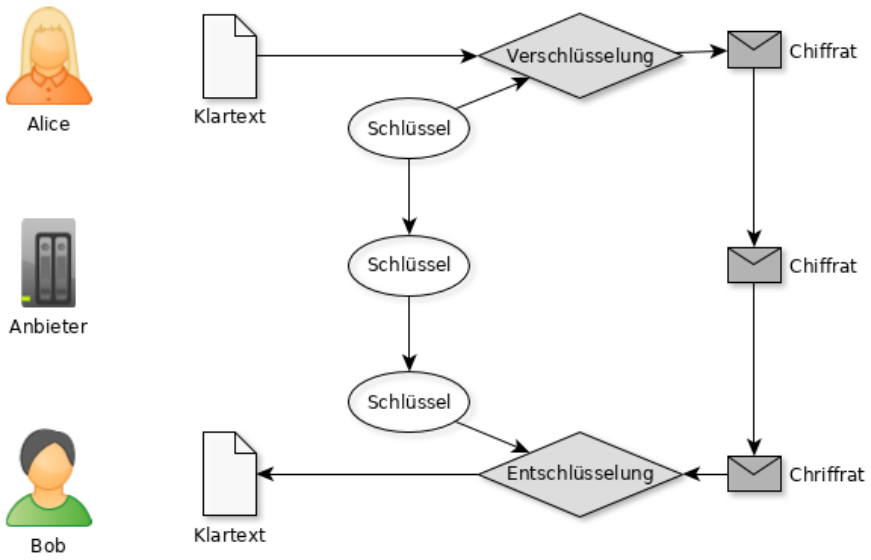

Abbildung 1: Schlüsseltausch bei symmetrischen Verfahren

Soll in einer Webanwendung nun Bob in der Lage sein, eine verschlüsselte Information von Alice zu lesen, so benötigt er den geheimen Schlüssel, welchen Alice bei der Verschlüsselung gewählt hat. Dieser muss entweder offline an Bob übermittelt werden oder, wie in Abbildung 1 dargestellt, von Alice an den Server übertragen und dort für Bob aufbewahrt werden. Der geheime Schlüssel wird so zu einem geteilten Geheimnis, welches je nach gewähltem Übertragungsweg auch dem Anbieter der Anwendung zur Verfügung steht. Soll die verschlüsselte Information darüber hinaus von weiteren Benutzern gelesen werden können, muss Alice auch diesen ihren geheimen Schlüssel anvertrauen.

Da nun Bob und der Anbieter den geheimen Schlüssel dieser Information kennen und diesen neben dem Ent- auch zum Verschlüsseln der Information nutzen können, ist die Integrität und Authentizität der Information nicht mehr gewährleistet. Auch kann die Schlüsselinformation auf dem Übertragungsweg zu Bob von einem unberechtigten Dritten abgefangen werden. Daher kann nicht mehr mit Gewissheit festgestellt werden, ob die Information noch mit der eingangs von Alice verschlüsselten Version übereinstimmt und tatsächlich nur von Bob entschlüsselt werden kann. Die Vertraulichkeit einer symmetrisch verschlüsselten Information steht und fällt also mit der Sicherheit der Schlüsselübertragung.

3.1.2. Asymmetrische Kryptografie

Abgefangene Schlüsselinformationen stellen bei asymmetrischen Verfahren keinen Bruch der Vertraulichkeit dar. Diese erstmals 1976 von Whitfield Diffie und Martin Hellman vorgestellten Verfahren arbeiten mit einem sogenannten Schlüsselpaar. Jeder Benutzer erstellt seinen eigenen privaten Schlüssel. Dieser sollte sicher abgelegt und nur für den Benutzer selbst zugänglich sein. Ein komplexes, nicht umkehrbares mathematisches Verfahren errechnet daraus einen öffentlichen Schlüssel, welcher mit einem Anbieter und jedem anderen Nutzer geteilt werden kann. Aus einem Chiffrat, welches unter Verwendung des öffentlichen Schlüssels erzeugt wurde, kann ausschließlich mit Kenntnis des zugehörigen privaten Schlüssel die ursprüngliche Information errechnet werden. (Ristic 2014)

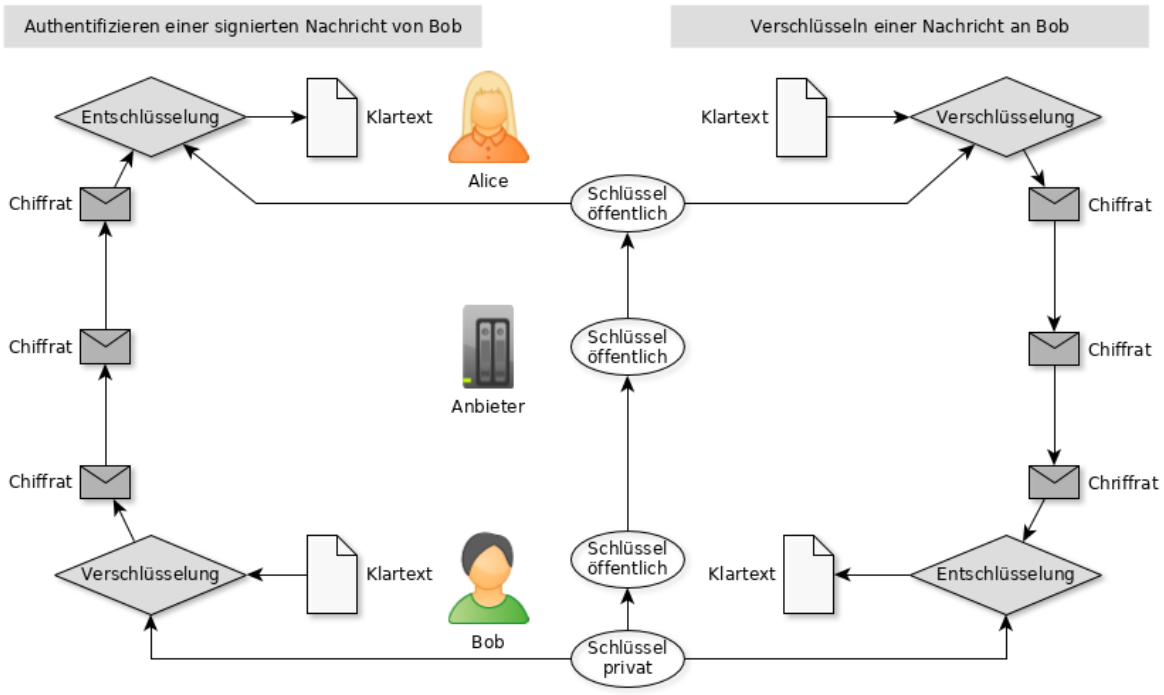

Abbildung 2: Verschlüsseln und signieren mit asymmetrischen Verfahren

Um eine geheime Information mit Bob zu teilen, benötigt Alice lediglich dessen allgemein bekannten öffentlichen Schlüssel. Der so entstandene Kryptotext kann, wie in Abbildung 2 dargestellt, nur von Bob selbst mit seinem zugehörigen privaten Schlüssel dechiffriert werden. Asymmetrische Verfahren können auch in der Gegenrichtung verwendet werden, um eine Nachricht digital zu signieren. Verwendet Bob seinen privaten

Schlüssel, um ein Chiffrat zu erzeugen, kann dieses, wie ebenfalls in Abbildung 2 gezeigt, nur mit dem zugehörigen öffentlichen Schlüssel entschlüsselt werden. Alice kann nun mit Sicherheit feststellen, ob eine so signierte Nachricht tatsächlich von Bob stammt. (Ristic 2014)

In beiden Fällen kann der Anbieter ohne Gefahr für Vertraulichkeit und Authentizität sowohl das Chiffrat als auch den öffentlichen Schlüssel speichern. Er ist nicht in der Lage, ein für Bob verschlüsseltes Dokument zu lesen oder eine von Bob signierte Information unbemerkt zu verändern. Damit sind je nach Anwendung die Vertraulichkeit oder die Authentizität und Integrität der Information gesichert, da die kompromittierbare Übertragung einer geteilten Geheiminformation gänzlich entfällt. (Meinel und Sack 2009)

Die zugrundeliegenden mathematischen und logischen Operationen, welche das Ver- und Entschlüsseln mit unterschiedlichen Schlüsseln ermöglichen, tragen auch zu dem größten Nachteil asymmetrischer Verfahren bei. Durch die hohe Komplexität mathematischer Berechnungen eignen sie sich nicht zum Verschlüsseln großer Datenmengen. Die benötigte Rechenzeit asymmetrischer Chiffren ist um den Faktor 1000 höher als bei gängigen symmetrischen Verfahren. Gerade bei Nutzeranforderungen an gut und flüssig bedienbare Webanwendungen ist dies nur schwer hinzunehmen. Daher werden diese Verfahren meist nicht direkt zur Verschlüsselung der Daten genutzt, sondern zum Authentifizieren und zur Sicherung von geteilten Geheimnissen vor der Übertragung, wie in Abschnitt 3.4 näher erläutert. (Ristic 2014)

3.2. Aufgabenverteilung in Client-Server Systemen

In Webanwendungen gibt es zwei Systeme, von denen die kryptografischen Aufgaben bearbeitet werden können. Auf der einen Seite kann dies von der serverseitigen Anwendung übernommen werden, es kann aber auch auf die Clientanwendung im Browser des Benutzers ausgelagert werden. Die folgenden Abschnitte stellen beide Möglichkeiten vor und diskutieren Vorteile und Nachteile, die dabei jeweils abzuwägen sind.

3.2.1. Verschlüsselung auf dem Server

Werden sämtliche kryptografischen Aufgaben auf der Serverseite der Applikation durchgeführt, so sendet der Browser weiterhin Anfragen und Informationen im Klartext an den Webdienst. Es ist die Aufgabe des Servers, aus den vom Nutzer übermittelten Informationen unter Verwendung des richtigen Schlüssels ein Chiffrat zu erzeugen, welches in der Datenbank gespeichert wird. Bei der Abfrage von Informationen wird dieses zunächst ebenfalls vom Server entschlüsselt und anschließend im Klartext an die Clientanwendung im Browser übermittelt. Die serverseitige Abfragelogik ist dadurch kaum begrenzt. Der Server ist weiterhin in der Lage, die Informationen der Nutzer zu durchsuchen, zu sortieren und zu gruppieren. Auch ein Suchindex der verschlüsselt gespeicherten Daten kann geführt werden. Die Leistungsfähigkeit der Anwendung ändert sich bei diesem Ansatz nur in dem Maße der Rechenzeit, welche für die Durchführung der Kryptooperationen auf dem Server benötigt wird. (Karlinger, Ettmayer und Schrefl 2011)

Diese Vorteile gehen allerdings stark zu Lasten der Informationssicherheit. Um Daten im Auftrag der Nutzer auf dem Server ver- und entschlüsseln zu können, muss dieser Kenntnis von den geheimen Schlüsseln der Anwender haben. Damit ist es mit Zugriff auf den Datenbestand möglich, die Nutzerinformationen auch ohne Erlaubnis der Benutzer einzusehen und zu verändern. Zudem werden Informationen zwischen Client und Server im Klartext übertragen, wobei sie grundsätzlich abgehört werden können. Von Anwenderseite ist daher im Ausgleich für Performance und Abfragemöglichkeiten ein erheblicher Vertrauensvorschuss gegenüber dem Anbieter und der Absicherung seiner Systeme notwendig. (ebd.)

3.2.2. Verschlüsselung in der Clientapplikation

Bei Verschlüsselung der Informationen im Browser wird nur das dabei entstandene Chiffrat über das Internet an den Server gesendet. Der verwendete Schlüssel verbleibt in der Clientapplikation. Dadurch können die Daten zwar auf dem Server zentral abgelegt werden, doch weder der Anbieter noch ein unbefugter Dritter ist mangels Schlüsselinformation in der Lage, diese zu entschlüsseln. Auch etwaige auf dem Übertragungsweg abgefangene Daten sind nicht verwertbar. (ebd.) Damit sind allerdings auch

die Abfragemöglichkeiten der Serveranwendung stark eingeschränkt. Bei Einsatz von sogenannten ordnungserhaltenden Verschlüsselungsalgorithmen ist es noch möglich, einfache algebraische Vergleiche mit verschlüsselten Informationen durchzuführen (Agrawal u. a. 2004). Dagegen kann beispielsweise ein Filtern nach Textinhalten erst nach der Entschlüsselung im Client erfolgen. Dieses Modell stellt durch den Verbleib der kryptografischen Geheimnisse im Client die Vertraulichkeit der gespeicherten Informationen sicher. Durch zusätzliche Speicherung eines Hashed Message Authentication Codes, einer unter Verwendung der Daten nach einem kryptografischen Verfahren erzeugten Prüfsumme (Ristic 2014), ist darüber hinaus auch die Integrität der Daten nachweisbar.

Die Bereitstellung der Schlüsselinformationen durch die Clientanwendung bringt allerdings auch komplexe Probleme mit sich. Diese sind an den Browser gebunden, den der Nutzer zur Verschlüsselung seiner Informationen genutzt hat. Für den Fall, dass die Sitzungsdaten gelöscht, der Browser oder auch der Computer gewechselt wird, muss die Applikation eine Möglichkeit vorsehen, die Schlüsselinformationen zu sichern und zu importieren. Andernfalls ist bei Verlust der Informationen in der Browsersitzung kein Zugriff auf die verschlüsselten Daten mehr möglich. Diese sind auch für den Urheber unwiederbringlich verloren. Auch das Teilen von Informationen mit anderen Nutzern der Anwendung wird erschwert, da diese ebenfalls Zugriff auf die genutzte Schlüsselinformation benötigen. Eine Möglichkeit zur Lösung dieses Problems ist eine manuelle Exportfunktion für die genutzten Schlüssel, welche durch den Nutzer anschließend auf andere Geräte transferiert oder per E-Mail oder Telefon mit anderen Anwendern geteilt werden können. Alternativ kann der Server des Anbieters als Treuhänder zur Archivierung der Schlüsselinformationen in ebenfalls verschlüsselter Form dienen. Die Clientanwendung kann von diesem benötigte Schlüssel des Anwenders abfragen und erst lokal entschlüsseln, sowie geteilte Schlüsselgeheimnisse als Chiffrat an den Server übermitteln. Die Anforderungen an die Planung dieses Kryptosystems sind hoch, damit sichergestellt ist, dass der Anbieter keinen Zugriff auf Schlüsselinformationen im Klartext hat und im Browser dennoch mit den empfangenen Daten gearbeitet werden kann. Die Vertraulichkeit der Informationen bleibt damit gewährleistet. Eine mögliche Umsetzung beschreibt Abschnitt 5.2.

3.3. Identifizieren sensibler Daten

Für die Entscheidung, welche Daten als schützenswert anzusehen sind und in verschlüsselter Form verarbeitet werden sollten, kann die in Abschnitt 2.1 beschriebene Gesetzesgrundlage aus Bundesdatenschutzgesetz und Telemediengesetz herangezogen werden. Die von diesen Gesetzen geregelten Informationen können in der Anwendung dennoch in unterschiedlicher Form auftreten.

3.3.1. Nutzerinformationen

Bei Informationen, die von Nutzern der Anwendung selbst erfasst werden, ist der Grad des Personenbezugs zu berücksichtigen. Mit äußerster Vorsicht zu behandeln sind Daten, welche etwa die Intimsphäre oder den Gesundheitszustand der Nutzer betreffen. Im Klartext sollten diese Daten dem Anbieter daher nur in pseudonymisierter, nicht zurückverfolgbarer Form vorliegen. Weniger sensible Informationen dürfen, sofern der einzelne Anwender einwilligt, für eine Profilbildung etwa zu Marketingzwecken oder der bedarfsgerechten Gestaltung des Dienstes herangezogen werden. Auch hier ist auf eine strikte Zweckgebundenheit der erfassten Informationen zu achten. Explizit gestattet ist allerdings die Verarbeitung von Informationen zur Rechnungsstellung, auch über die eigentliche Nutzungsdauer des Dienstes hinaus. Einer Erhebung und Verarbeitung von Adress- und Zahlungsinformationen kann ein Kunde in der Regel nicht widersprechen. (Voigtländer und Schmischke 2011; Schneider 2013)

Daraus ergibt sich, dass nur Informationen, welche zu diesem Zweck benötigt werden, in unverschlüsselter Form vorliegen sollten. Sofern eine Individualisierung in der Applikation erfolgt und der Nutzer dem zugestimmt hat, kann die Klartextspeicherung auf weitere, dazu benötigte Informationen ausgeweitet werden. Alle weiteren von Anwendern erfassten Informationen, insbesondere mit Berührung der intimsten Lebensbereiche, sollten ausschließlich in verschlüsselter Form übertragen und gespeichert werden, um deren Nutzung durch Administratoren oder unberechtigte Dritte von vornherein auszuschließen. Sollten dennoch Informationen daraus etwa für statistische Auswertungen benötigt werden, ist es besser, pseudonymisierte Ableitungen aus den Informationen bereits in der Clientanwendung zu erzeugen.

3.3.2. Dateien

Ähnlich wie bei den vorhergehend beschriebenen Nutzerinformationen kann auch die Sensibilität von Dateien bewertet werden, welche von Anwendern auf die Server des Anbieters hochgeladen werden. Handelt es sich um ein Dokument, welches zur Kundenverwaltung oder Abrechnung benötigt wird, beispielsweise einen Gewerbeschein zur Legitimation eines Businesskunden, oder um Informationen, die zur genehmigten Individualisierung der Anwendung genutzt werden, so kann die Datei in unverschlüsselter Form verarbeitet werden. Allerdings sollte ein entsprechend höherer Aufwand zur Autorisierung betrieben werden, um unberechtigte Zugriffe auf im Klartext vorliegende Dokumente zu verhindern. Auch für hochgeladene Dateien gilt der Grundsatz, Dokumente mit Bezug zur Intimsphäre des Nutzers und solche, ohne gerechtfertigte Zweckbindung, ausschließlich verschlüsselt zu übertragen. Eine Verarbeitung des Inhaltes, etwa die Erstellung eines Thumbnails zu einer Bilddatei, sollte ebenfalls vor dem Upload im Client erfolgen.

3.3.3. Metadaten

Bei erfassten Metadaten einer Anwendung handelt es sich meist um Informationen, die zur korrekten Funktionsweise der Applikation benötigt werden. Dabei handelt es sich beispielsweise um

- Benutzername zur Identifizierung
- Benutzer-ID des Erstellers eines Eintrags
- Datum der Erstellung
- Dateiname einer hogeladenen Datei
- Dateigröße und Dateityp einer hochgeladenen Datei

Einige dieser Informationen sind für die ordnungsgemäße Nutzung der Applikation unabdingbar, etwa Benutzer-IDs um Besitzer eines Datensatzes zu autorisieren. Die Größe einer hochgeladenen Datei könnte für Quota-Begrenzungen der Nutzer und damit für Verwaltungs- und Abrechnungszwecke benötigt werden. Datumsfelder wiederum ermöglichen es, Auflistungen von Datensätzen bereits serverseitig zu sortieren und zu

filtern und so etwa bei Nutzung auf mobilen Endgeräten Bandbreite und Datenvolumen einzusparen.

Das macht die Unterscheidung von sensiblen und benötigten Metadaten sehr komplex ohne deren Bedeutung zu schmälern. Alleine durch meist im Klartext benötigte IDs zur Referenzierung zugehöriger Datensätze sowie Zeitstempel ist die Bildung detaillierter Nutzungsprofile möglich. Verbindungsdaten, welche bei der Kommunikation von Anwendern einer Mehrbenutzeranwendung entstehen, geben tiefe Einblicke in soziale Kontakte und das Privatleben der betreffenden Personen. (Mayer, Mutchler und Mitchell 2016) Wo möglich sollten daher auch Metadaten, die zum Betrieb der Anwendung nicht zwingend benötigt werden, in verschlüsselter Form übertragen werden.

3.4. Teilen verschlüsselter Daten

Der folgende Abschnitt legt die Annahme zugrunde, dass Informationen in der Anwendung wie in Abschnitt 3.2.2 beschrieben in der Clientanwendung verschlüsselt und dem Anbieter die Schlüsselinformationen nicht im Klartext übertragen werden sollen.

Damit Anwender einer Mehrbenutzerapplikation miteinander kommunizieren und Daten austauschen können, müssen sie in der Lage sein, die Informationen ihrer Kommunikationspartner zu lesen. Dazu ist es nötig, passende kryptografische Verfahren auszuwählen (siehe Abschnitt 3.1) und ein Schlüsselmanagement zu implementieren, welches allen Teilnehmern einer Kommunikation sicheren Zugriff auf die benötigten Schlüssel ermöglicht.

3.4.1. Symmetrische Verschlüsselung

Auf dem Feld der symmetrischen Verfahren stehen diverse Chiffre zur Verfügung, die eine sichere und performante Verschlüsselung größerer Datenmengen auch im Browser ermöglichen, beispielsweise die Stromverschlüsselungen AES-GCM[3] oder Salsa20[4].

[3] `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf`
[4] `https://cr.yp.to/snuffle/salsafamily-20071225.pdf`

Um einen symmetrisch verschlüsselten Datensatz mit dem Benutzer Bob zu teilen, muss Alice ihm, wie in Abschnitt 3.1.1 beschrieben, das von ihr verwendete Geheimnis anvertrauen. Dies kann beim Einsatz symmetrischer Verschlüsselung als kritischer Vorgang betrachtet werden. Die Vertraulichkeit der Informationen ist nur gewährleistet, solange der benötigte Schlüssel geheim bleibt. Soll der Austausch des Geheimnisses mit Bob etwa automatisch über die Systeme des Anbieters erfolgen, so liegt es dort im Klartext vor. Schlüssel und Passwörter, die im Klartext auf einem Server gespeichert werden, können nicht als sicher angesehen werden. Sie können von Administratoren der Applikation genutzt werden, um unbefugt Zugriff auf die verschlüsselt gespeicherten Informationen zu erlangen. (Meinel und Sack 2009)

Obwohl sich symmetrische Verfahren aufgrund der hohen Performance sehr gut zur Verschlüsselung der erfassten Informationen eignen, ergeben sich schwerwiegende Probleme beim Austausch von Schlüsselinformationen mit anderen Nutzern. Bei der Weitergabe der benötigten Geheimnisse kann nicht gewährleistet werden, dass diese nicht abgehört oder von unberechtigten, nicht an der Kommunikation beteiligten Personen eingesehen werden können. (ebd.)

3.4.2. Asymmetrische Verschlüsselung

Bei Anwendung asymmetrischer Verfahren entfällt die kritische Übertragung der Schlüsselinformation. Wie bereits in Abschnitt 3.1.2 beschrieben, benötigt Alice nur den aus Bobs Geheiminformation berechneten öffentlichen Schlüssel. Da hiermit chiffrierte Daten nur mit dem zugehörigen privaten Schlüssel von Bob dechiffriert werden können, dieser jedoch nie zu Alice übertragen wird, bleibt die Vertraulichkeit der Informationen gewährleistet. Weder der Anbieter, noch unberechtigte Dritte sind in der Lage, die von Alice an Bob gesandten Informationen in die ursprüngliche, lesbare Form zu bringen. (ebd.)

Ein Nachteil erwächst direkt aus der Verschlüsselung der geteilten Information mit dem öffentlichen Schlüssel des Kommunikationspartners. Alice ist anschließend selbst nicht mehr in der Lage, die verschlüsselten Daten wiederherzustellen. Um ihr weiterhin Zugriff auf die geteilte Information zu ermöglichen, muss diese zusätzlich mit ihrem eigenen öffentlichen Schlüssel chiffriert abgelegt werden. Findet die Kommunikation mit mehr als

einem Partner statt, ist dieser Vorgang auch für jeden weiteren Teilnehmer erforderlich. Die Menge der Daten, die von der Clientanwendung zum Server des Anbieters übertragen werden steigt so linear mit der Anzahl der Kommunikationspartner. (Ristic 2014) Wie in Abschnitt 3.1.2 erwähnt, benötigen die Operationen asymmetrischer Chiffre ein Vielfaches der Rechenleistung symmetrischer Verfahren. Für die Durchführung kryptografischer Aufgaben in der Clientsoftware eignen sie sich daher nur bei sehr geringen Datenmengen. (Meinel und Sack 2009)

Obwohl asymmetrische Verfahren durch die entfallene Übertragung einer Geheiminformation die Vertraulichkeit der Informationen bestens gewährleisten können, muss ihr Einsatz in einer Applikation gut durchdacht werden. Zu eifrige Verwendung dieser Methoden kann zu einem sprunghaften Anstieg von übertragenen und gespeicherten Datenmengen sowie der von der Anwendung benötigten Rechenzeit führen.

3.4.3. Hybride Verschlüsselung

Beide hier beschriebenen Verschlüsselungsverfahren können in Kombination eingesetzt werden, um einige Nachteile des jeweils anderen Verfahrens zu kompensieren. Das so entstehende Kryptosystem wird als hybride Verschlüsselung bezeichnet. (ebd.)

Alice wählt einen ausreichend langen Schlüssel, mit welchem sie die Information symmetrisch verschlüsselt. Um dieses Geheimnis nun sicher zu Bob zu transportieren, wird daraus unter Verwendung von Bobs öffentlichem Schlüssel ein Chiffrat erzeugt. Dieses kann nun ausschließlich von Bob mit seinem privaten Schlüssel dechiffriert werden. Für Alice selbst und alle weiteren Kommunikationsteilnehmer wird ebenfalls lediglich eine Kopie der symmetrischen Schlüsselinformation asymmetrisch verschlüsselt abgelegt. Auf diese Weise sind alle Kommunikationspartner in der Lage, die verschlüsselten Informationen mit ihrer eigenen Kopie der Geheiminformation in eine lesbare Form zu überführen. Im Gegensatz zu dem alleinigen Einsatz symmetrischer Verfahren wird zu keinem Zeitpunkt eine lesbare Schlüsselinformation an den Anbieter übertragen. Auch das, bei rein asymmetrischer Kryptografie nötige, mehrfache Verschlüsseln und Abspeichern der Information entfällt, da nur der nötige symmetrische Schlüssel individuell für jeden Kommunikationspartner abgelegt wird. (ebd.)

4. Kryptografie in bestehenden Applikationen

Dieses Kapitel untersucht Mehrbenutzeranwendungen aus den Bereichen gemeinsamer Dokumentenablage und Messaging-Dienste auf den Einsatz kryptografischer Methoden zum Schutz der gespeicherten Informationen. Auch eine Bibliothek zum „Nachrüsten“ browserseitiger Informationsverschlüsselung in bestehenden Anwendungen wird analysiert.

4.1. Dateiablage und Dateiaustausch

Dienste zur kollaborativen Verwaltung und Bearbeitung von Datenbeständen in der Cloud werden in vielen privaten, geschäftlichen und wissenschaftlichen Bereichen heute wie selbstverständlich genutzt. So verzeichnete etwa das Unternehmen Dropbox im Juli 2016 über 500 Millionen Nutzer seiner Dokumentenablage[5]. Die nachfolgenden Abschnitte zeigen, wie verschiedene Anbieter auf diesem Gebiet mit den Informationen ihrer Kunden verfahren.

4.1.1. Dropbox und Google Drive

Die beiden Marktführer auf diesem Gebiet, Google Drive [6] und Dropbox [7], bieten ihre Speicherdienste ausschließlich über TLS gesicherte Verbindungen an. Viele Endbenutzer sehen dies bereits als Verschlüsselung ihrer Daten. Hierbei handelt es sich jedoch lediglich um die Transportverschlüsselung für den Übertragungsweg zu den Servern der Anbieter. Im Rechenzentrum liegen sämtliche Informationen erneut im Klartext vor.

In seinen Allgemeinen Geschäftsbedingungen sichert das Unternehmen Dropbox seinen Nutzern ein hohes Maß an Schutz ihrer Daten zu. Es verweist auf ein „Team von Mitarbeitern, deren Aufgabe es ist, Ihre Informationen zu schützen und auf Schwachstellen im System zu prüfen“. (Dropbox Inc. 2016) Mit welchen Prozessen dies gewährleistet wird,

[5] https://de.statista.com/statistik/daten/studie/326447/umfrage/anzahl-der-weltweiten-dropbox-nutzer/
[6] https://www.google.com/intl/de_ALL/drive/
[7] https://www.dropbox.com/

geht aus dem Dokument nicht hervor. Ein Einsatz clientseitiger Verschlüsselung ist unwahrscheinlich, da Dropbox sich im selben Zug das Recht einräumt, sowohl gesammelte Profilinformationen als auch die vom Nutzer hochgeladenen Daten auszuwerten und zu verarbeiten, soweit dies für die angebotenen Funktionen der Datenablage vonnöten ist. Dieses Recht wird ferner auf Tochtergesellschaften und angebundene Drittanbieter ausgeweitet. (Dropbox Inc. 2016)

Die Nutzungsbedingungen von Google gehen noch einen Schritt weiter. Anwender räumen mit der Nutzung des Dienstes „Google (und denen, mit denen wir zusammenarbeiten) das Recht ein, diese Inhalte weltweit zu verwenden, zu hosten, zu speichern, zu vervielfältigen, zu verändern, abgeleitete Werke daraus zu erstellen (einschließlich solcher, die aus Übersetzungen, Anpassungen oder anderen Änderungen resultieren, die wir vornehmen, damit Ihre Inhalte besser in unseren Diensten funktionieren), zu kommunizieren, zu veröffentlichen, öffentlich aufzuführen, öffentlich anzuzeigen und zu verteilen“. (Google Inc. 2013) Diese Nutzungserlaubnis erstreckt sich auch auf Maßnahmen, die der Verbesserung der Google-eigenen Dienste dienen und erlischt nicht mit der Löschung der Daten oder einer Trennung von dem Dienst. (ebd.) Dies macht deutlich, dass eine Vertraulichkeitssicherung der Informationen nicht nur unwahrscheinlich ist, sondern diese auch mit den geschäftlichen Interessen des Anbieters kollidieren würde.

4.1.2. Tresorit

Der schweizer Dienst Tresorit bietet Dateisynchronisation und Dateiaustausch über die eigenen Systeme nach dem Zero-Knowledge-Prinzip[8] an. In den Nutzungsbedingungen lässt der Dienst sämtliche Eigentumsrechte und daraus erwachsenden Ansprüche an die hochgeladenen Informationen bei den Anwendern und schließt eine Nutzung durch Tresorit selbst aus. (Tresorit AG 2015)

Informationen zu hochgeladenen Dateien sowie der Dateiinhalt werden nach eigenen Angaben bereits in der Clientanwendung symmetrisch AES verschlüsselt und anschließend zu den Servern des Anbieters übertragen. Auch ein mit HMAC-SHA-512 gehashter Message Authentication Code wird zur Verifizierbarkeit des verschlüsselten Inhaltes mit abgelegt. Die geheime Schlüsselinformation, welche für jede Datei separat erstellt

[8] `https://tresorit.com/blog/zero-knowledge-verschlusselung/`

wird, wird, asymmetrisch mit dem öffentlichen Schlüssel des Benutzers gesichert und ebenfalls auf den Servern von Tresorit gespeichert. (Tresorit AG 2013)

Möchte der Anwender seine Dokumente für andere Tresorit-Nutzer freigeben, so tauscht er mit diesen zunächst die öffentlichen Schlüssel aus. Den Vorgang aus zertifikatsbasierter Authentifizierung und Handshake mit Austausch einer zufällig generierten, verschlüsselten Nachricht hat Tresorit bereits als „ICE Protocol“ zum Patent angemeldet. Mit Kenntnis des öffentlichen Schlüssels des anderen Benutzers wird in der Clientanwendung ein weiteres asymmetrisches Chiffrat des symmetrischen Dokumentenschlüssels erzeugt und ebenfalls auf den Servern des Anbieters abgelegt. Damit nach Entzug einer Zugriffsberechtigung der betreffende Anwender nicht mehr in der Lage ist, aktuelle oder neue Versionen eines Datensatzes zu entziffern, wird bei jeder Änderung an einem Datenobjekt, sei es der Inhalt, Metainformationen oder die Liste zugriffsberechtigter Benutzer, ein neuer symmetrischer Schlüssel erzeugt und das Objekt damit neu verschlüsselt. Mit der Schlüsselinformation aus einer vorhergehenden Zugriffsberechtigung ist damit kein Lesen der Informationen im Klartext mehr möglich. (ebd.)

4.2. Instant Messaging

Kommunikationsdienste zum Austausch von Nachrichten mit anderen Nutzern erfreuen sich seit Jahrzehnten wachsender Beliebtheit. In der Vergangenheit war es nicht sonderlich schwer, die Kommunikation von Benutzern der Dienste zu belauschen. Für führende Programme zur Datenstromanalyse im Netzwerk existieren teilweise eigene Plugins, um speziell die Kommunikation von Messengern wie ICQ im Netzwerk herauszufiltern[9]. Abhilfe schafft die im Jahr 2005 vorgestellte Off The Record Verschlüsselung für Instant Messaging[10], welche durch Plugins nachgerüstet oder als lokales Protokollgateway betrieben werden kann. Da sich die OTR Implementierungen oft nicht durch Nutzerfreundlichkeit auszeichnen, hat diese Methode des verschlüsselten Nachrichtenaustauschs keine hohe Verbreitung. Auch der Messagingentwickler und Berater Daniel Gultsch kommt zu dem Schluss, dass es sich bei OTR eher um einen „weniger gut funktionierenden Hack“ (Gultsch 2016) handelt.

[9] `https://www.wireshark.org/docs/dfref/i/icq.html`
[10] `https://otr.cypherpunks.ca/`

Einen neuen Ansatz bringt das von Open Whisper Systems entwickelte Protokoll „Signal“. Während die ursprüngliche Version aus dem Jahr 2013 noch auf OTR basiert, implementiert Version 2 bereits den neu vorgestellten „Double Ratchet Algorithmus“ (ehem. Axolotl)[11]. Dieser ersetzt das einzelne, bisher zur Verschlüsselung genutzte Schlüsselpaar durch eine Kombination mehrerer Langzeit-, Kurzzeit- und Nachrichtenschlüssel, wie im Folgenden weiter beschrieben. Die ebenfalls von Open Whisper Systems stammende und auch unter Kryptologen sehr beliebte Applikation „Signal“[12] erhält 2014 die erste Implementierung dieses Verfahrens. Dank vorhandener Implementierungen des beschriebenen Protokolls für Programmiersprachen wie Java[13], JavaScript[14] oder C[15] verbreitet sich seine Nutzung zunehmend. Die Entwickler des populären Jabber Client Conversations[16] etwa forcieren eine Aufnahme der auf Signal basierenden Kommunkationsverschlüsselung OMEMO in den XMPP Standard[17]. Auch der inzwischen zu Facebook gehörende Anbieter WhatsApp veröffentlicht Anfang 2016 neue Versionen seiner Chatclients für mobile Endgeräte, die eine Ende-zu-Ende Verschlüsselung der übertragenen Informationen nach dem Signal Protokoll durchführen. Aufgrund der hohen Verbreitung widmet sich die folgende Beschreibung des Protokolls der Implementierung von WhatsApp. Diese stimmt aber in großen Teilen mit anderen Implementierungen wie Signal selbst oder OMEMO überein.

Das Protokoll sieht für symmetrische Verschlüsselung die Verwendung von AES-CBC vor, bei asymmetrischen Schlüsseln die Nutzung von Curve25519 und die Erstellung von Hashes nach HMAC-SHA256 zur Integritätssicherung. Bereits bei der Installation legt die Software einige asymmetrische Schlüsselpaare auf auf dem Smartphone des Nutzers an. Zu den Schlüsselpaaren gehören ein Identity Keypair als Langzeitschlüssel, ein Signed Pre Key, welcher mit dem privaten Identity Key signiert wird, sowie ein Vorrat an One-Time Pre Keys, welche nur einmalig durch den Client genutzt werden. Bei Bedarf wird dieser Vorrat durch die Anwendung automatisch wieder aufgefüllt. Die öffentlichen Schlüssel zu allen generierten Schlüsselpaaren sendet die Software an den Server des Anbieters. (WhatsApp Inc. 2016)

[11] `https://github.com/trevp/double_ratchet/wiki`
[12] Android: `https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms`
iOS: `https://itunes.apple.com/us/app/signal-private-messenger/id874139669`
[13] `https://github.com/WhisperSystems/libsignal-protocol-java`
[14] `https://github.com/WhisperSystems/libsignal-protocol-javascript`
[15] `https://github.com/WhisperSystems/libsignal-protocol-c`
[16] `https://conversations.im/`
[17] `https://xmpp.org/extensions/inbox/omemo.html`

Beim erstmaligen Versand einer Nachricht an einen anderen Benutzer fordert die Software dessen öffentlichen Identity und Signed Pre Key sowie ein Exemplar aus dem Vorrat der Einwegschlüssel an. Diese Informationen fließen neben dem eigenen privaten Schlüssel in die Erstellung eines „Master Secret“ mittels ECDH[18]. Die Vorratshaltung der öffentlichen Einwegschlüssel auf den Servern des Anbieters ermöglichen damit den Aufbau einer Verbindung auch zu Partnern, die gerade nicht verbunden sind. (WhatsApp Inc. 2016)

Um sicherzugehen, dass die Sitzung ordnungsgemäß aufgebaut werden und der Initiator der Kommunikation sofort Nachrichten schreiben kann, sendet die Anwendung so lange das „Master Secret“ an den Kommunikationspartner, bis von diesem die erste Antwort eintrifft. Aus diesem Geheimnis können beide Seiten mithilfe ihrer privaten Schlüssel durch HKDF[19] einen gemeinsamen symmetrischen Root Key und Chain Key errechnen. Letzterer wird genutzt, um für jede Nachricht einen einmaligen Message Key zu errechnen, wonach der Chain Key inkrementiert und erneut gehasht wird. So ist auch der Kommunikationspartner in der Lage, den verwendeten Schlüssel einer Nachricht zu errechnen. Zusätzlich wird der Chain Key bei jeder Antwort des Gesprächspartners durch eine neue Berechnung aus dem Root Key ersetzt. Spätestens nach Anwendung dieses Double Ratchet genannten Verfahrens gibt es keine Möglichkeit mehr, Rückschlüsse auf die Schlüssel der zuletzt versendeten Nachrichten zu ziehen. (ebd.)

Beim Versand von Dateianhängen und Medien generiert die App einen zufälligen symmetrischen Schlüssel. Dieser wird über das Chiffre AES256-CBC mit einem ebenfalls zufällig gewählten Initialisierungsvektor zur Blockverschlüsselung genutzt, um aus der Datei ein Chiffrat zu erzeugen, welches auf den Servern des Anbieters abgelegt wird. Der Kommunikationspartner erhält eine, wie zuvor beschrieben, verschlüsselte Nachricht mit dem Speicherort der Datei und den genutzten symmetrischen Geheimnissen. (ebd.)

Auch zum sicheren Teilen von Informationen mit mehreren Benutzern über Gruppenchats kommt eine Implementierung des Signal-Protokolls zum Einsatz. Jeder Teilnehmer einer Gruppe generiert bei seiner ersten Nachricht an diese einen eigenen symmetrischen Sender Key. Dieser wird zunächst zur Authentifizierung mit dem eigenen privaten

[18] Elliptic Curve Diffie Hellman
`https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf`
[19] HMAC key derivation function `https://eprint.iacr.org/2010/264.pdf`

Schlüssel signiert und anschließend mit den öffentlichen Schlüsseln der anderen Gruppenteilnehmer jeweils individuell verschlüsselt. Diese Schlüssel werden mit der ersten Nachricht an die Gruppe mitgesendet und von allen anderen Gruppenteilnehmern für die Entschlüsselung weiterer Nachrichten gespeichert. Wird ein Mitglied aus der Gruppe entfernt, so löschen alle Gruppenteilnehmer ihre gespeicherten Sender Keys. Diese werden nun erneut mit den ersten Nachrichten, wie bereits beschrieben, erzeugt und verteilt. Damit ist sichergestellt, dass ein ehemaliges Mitglied einer Gruppe nicht in der Lage ist, nach seinem Austritt geteilte Informationen zu dechiffrieren. (WhatsApp Inc. 2016)

4.3. Integration in bestehende Webanwendungen

Um ohne große Vorkenntnisse kryptografische Methoden zur Informationssicherung im eigenen Projekt einzusetzen, gibt es diverse Projekte, die zu diesem Zweck eingebunden werden können. Mit Mylar stellt dieser Abschnitt ein Projekt des Massachusetts Institute of Technology vor, welches noch weitere Ziele neben der kryptografischen Sicherung der Inhalte in der Anwendung verfolgt. Auch der Code der Applikation selbst wird signiert und im Browser des Benutzers auf Echtheit geprüft. Mit minimalen Änderungen an der Applikation selbst soll es möglich sein, vordefinierte Informationen browserseitig verschlüsselt zu übertragen und Schlüssel zum Teilen der Informationen mit anderen Benutzern zu teilen. Auch begegnet Mylar so dem Vertrauensproblem der Kryptografie im Browser (Ptacek 2011) durch Überprüfung sämtlicher empfangener Daten einschließlich des empfangenen Applikationscodes selbst. Dieser wird aus zwei unabhängigen Quellen geladen. Eine Quelle liefert im Vorfeld durch den Anbieter signierte, statische Daten wie die Haupt-HTML-Datei der Anwendung. Die andere liefert weitere Ressourcen und Assets, welche durch Prüfsummen verifiziert werden. Angreifer mit Zugriff auf den Code der Anwendung oder nicht vertrauenswürdige Administratoren haben dadurch keine Möglichkeit, Schadcode zum Abgreifen von Daten oder Schlüsselinformationen einzufügen. (Popa u. a. 2016)

Wie viele populäre Ansätze verwendet auch Mylar zur Verschlüsselung der Informationen ein symmetrisches Verfahren, dessen Geheiminformation anschließend für den Zugriffsberechtigten asymmetrisch verschlüsselt abgelegt wird. Zugriffsberechtigt kann

ein Benutzer, aber auch ein anderes Datenobjekt sein. Darüber ergibt sich ein „Access Graph“ mit einer Schlüsselkette. Diese ermöglicht es beispielsweise, die Schlüssel zu den Nachrichten in einem Chatraum mit dessen öffentlichem Schlüssel abzulegen. Benutzer mit Zugriff auf diesen Chatraum haben die Möglichkeit, auch den privaten Schlüssel des Raumes zu beziehen und damit auch die Schlüssel der enthaltenen Nachrichten zu verwenden. (Popa u. a. 2016)

Beim Teilen von Informationen mit weiteren Nutzern erfragt die Clientapplikation bei der Serveranwendung deren öffentliche Schlüssel. Um zu vermeiden, dass ein solcher von einem Angreifer unbemerkt ausgetauscht wird, um so Zugriff auf vertrauliche Informationen zu erlangen, werden öffentliche Schlüssel der Benutzer gemeinsam mit dem Benutzernamen signiert. Dazu kommt entweder ein externer Identity Provider zum Einsatz, welcher, auch unabhängig von der Applikation, die Benutzernamen und zugehörigen Public Keys speichert, oder ein fest in den Programmcode der Anwendung integriertes Zertifikat mit privatem Schlüssel, dessen Sicherungskennwort nur dem Betreiber der Anwendung bekannt ist. Die Schlüssel und Namen jedes weiteren Principal Objektes werden bereits im Browser durch den privaten Schlüssel ihres Erstellers signiert. Erstellt Alice einen Chatraum, so signiert sie dessen öffentlichen Schlüssel mit ihrem eigenen privaten Schlüssel. Der Public Key einer einzelnen Chatnachricht wiederum wird mit dem Schlüssel des Chatraumes signiert. Da entweder das festkodierte Principal oder der unabhängige Identity Provider die Echtheit des Schlüssels von Alice bestätigt, ergibt sich ein „Certification Graph“, über den Bob vor der Verwendung die Vertrauenswürdigkeit von Chatraum- und Nachrichtenschlüsseln überprüfen kann. (ebd.)

Des weiteren beinhaltet das Konzept von Mylar eine Möglichkeit, über ein Set von verschlüsselten Daten zu suchen. Dafür müssen allerdings alle Worte in durchsuchbaren Feldern einzeln verschlüsselt werden. Der Server bekommt für untergeordnete Datenobjekte (wie Chatnachrichten) jeweils ein Delta zu dem Dokumentschlüssel des übergeordneten Eintrags mitgeteilt. Aus einem Token, bestehend aus Suchbegriff und dem Schlüssel des Chatraumes, welcher clientseitig erstellt wird, und dem Schlüsseldelta kann der Server nun die chiffrierte Repräsentation des Begriffs für jeden Nachrichtenschlüssel berechnen, ohne diesen selbst zu kennen. (ebd.)

5. Beispielimplementierung in einer Webanwendung

Das folgende Kapitel erläutert die im Rahmen dieser Arbeit entstandene Konzeptionierung und Implementierung clientseitiger Informationsverschlüsselung in einem frisch gestarteten Webprojekt. Dieses konzentriert sich primär auf das Sicherheitsziel der Vertraulichkeit von Informationen (siehe Abschnitt 2.2). Die zugrundeliegenden Anforderungen sind in Anhang A.2 aufgeführt.
Es werden verschiedene Bibliotheken vorgestellt, die kryptografische Aufgaben übernehmen oder das Verwalten von Sitzungsdaten und den Informationsaustausch mit der Serveranwendung erleichtern. Abschnitt 5.2 beschreibt schließlich das aus den bisherigen Kenntnissen entstandene Konzept und erläutert die zugehörige Implementierung. Eine Bewertung der hier gezeigten Ergebnisse folgt in Kapitel 6.

5.1. Wahl geeigneter Bibliotheken

Um eine moderne Webapplikation zu entwickeln, werden viele unterschiedliche Funktionalitäten benötigt. Ein Großteil davon ist bereits zu genüge implementiert und steht in Gestalt aktiv gepflegter Bibliotheken zur Verfügung. Auch für kryptografische Aufgaben finden sich ausgereifte Pakete, von denen dieser Abschnitt zwei vorstellt. Im Folgenden werden kurz die wichtigsten Bibliotheken erläutert, die bei der Implementierung des im Rahmen dieser Arbeit durchgeführten Projekts verwendet wurden. Eine Auflistung weiterer eingesetzter Module ist im Anhang A.1 aufgeführt.

Applikation

Die Wahl des Application Framework fällt mit Meteor[20] auf ein Full-Stack JavaScript Framework. Dieses beinhaltet bereits Funktionen, um Informationen aus der Webseite an den Server zu senden und die angezeigten Inhalte, ohne erneutes Laden der Seite, mit der Datenbank des Servers zu synchronisieren. Auch das integrierte Accountsystem mit Funktionen für Registrierung, Authentifizierung und Verifizierung von Nutzern ermöglicht eine rasche Konzentration auf die wesentlichen Funktionen der geplanten

[20] `https://www.meteor.com`

Anwendung. Sehr hilfreich bei der Entwicklung ist auch die automatische Aktualisierung des clientseitigen Codes im Entwicklungsmodus bei Änderungen an JavaScript- oder CSS-Dateien. Abschließend ermöglicht der eingebaute Transpiler eine volle Nutzung des EcmaScript 6 Standards und damit die Programmierung komplexer asynchroner Objekte unter Nutzung von Promises und Lambdas.

User Interface

Zum Aufbau der bedienbaren Oberfläche kommt React[21] zum Einsatz. Die von Facebook veröffentlichte Bibliothek fördert eine komponentenbasierte Entwicklung der Webapplikation mit unidirektionalem Datenfluss von übergeordneten zu untergeordneten Komponenten. Um diese, als sogenannte Container, mit asynchron geladenen Daten des Servers zu versorgen, fällt die Wahl auf das Paket react-komposer[22]. Bei einer Änderung von Informationen, die einer Komponente entweder von react-komposer oder von der übergeordneten Komponente übergeben werden, wird die Darstellung im Browser automatisch aktualisiert. Damit ermöglicht es React im Zusammenspiel mit den Push-Funktionen von Meteor ein Interface zu erstellen, welches ohne erneutes Laden der Seite stets den aktuellen Datenstand des Servers widerspiegelt.

Kryptografie

Gerade im Bereich der Kryptografie muss das Rad bei einer einzelnen Anwendung nicht neu erfunden werden. Es existieren bereits zahllose Implementierungen gängiger kryptografischer Methoden in JavaScript. Dadurch fällt allerdings die Entscheidung für oder wider einer Implementierung schwer.

Für das Projekt evaluiert werden SJCL des Stanford Computer Security Lab[23], eine Javascript-Implementierung der NaCl Bibliothek[24] sowie die in einer Empfehlung des W3C vorgestellte Web Crypto Api für nativ unterstützte Kryptografie in Browsern[25].

[21] `https://facebook.github.io/react/`
[22] `https://www.npmjs.com/package/react-komposer`
[23] `http://bitwiseshiftleft.github.io/sjcl/`
[24] `https://www.npmjs.com/package/tweetnacl`
[25] `https://www.w3.org/TR/WebCryptoAPI/`

Die Web Crypto Api des W3C ist zum Zeitpunkt der Konzeption des Projektes noch nicht ausreichend stabil in den wichtigen Browsern implementiert. Damit muss entweder die Zielgruppe der Anwendung eingeschränkt, oder eine Fallbackstrategie mit einer zweiten Bibliothek implementiert werden. Beides ist nicht im Sinne der geplanten Anwendung. Im Unterschied zu der Bibliothek aus Stanford ermöglicht tweetnacl-js die Nutzung kryptografischer Methoden für einen bestimmten Zweck (Schlüsselerstellung, Signieren, Verschlüsseln, Authentifizieren) ohne tiefe Kenntnis der darunterliegenden Operationen. Mit XSalsa20 zur symmetrischen Verschlüsselung und Poly1305 zur Authentifizierung bei der Entschlüsselung sowie Curve25519 für Schlüsselpaare und asymmetrische Verschlüsselung implementiert die Bibliothek als robust und performant angesehene Chiffren. Zusätzlich hat tweetnacl-js eine kürzlich durchgeführte Auditierung des Quellcodes durch Cure53[26] vorzuweisen, was sowohl für die Reife der Implementierung spricht als auch als vertrauensbildendes Argument (siehe Abschnitt 2.3) genutzt werden kann.

5.2. Konzept des implementierten Systems

Der folgende Abschnitt beschäftigt sich mit dem Konzept für die Implementierung gesicherter Ende-zu-Ende Kommunikation in der entstandenen Webapplikation. Die Liste der Anforderungen, welche von den Gründen für inhaltliche Verschlüsselung (Abschnitt 2) abgeleitet wurden, findet sich in Anhang A.2. Auch Erkenntnisse über die Verschlüsselung populärer Anwendungen aus Abschnitt 4 sind darin enthalten. Auszüge aus dem Sourcecode der Implementierung befinden sich aus Platzgründen ebenfalls im Anhang.

Der grundlegende Aufbau der Anwendung ist simpel. Registrierte Benutzer sind in der Lage, Einträge mit einem Titel und einem Beschreibungstext zu erfassen und zu diesem optional einen Dateianhang hinzuzufügen. Anschließend können Einträge aufgelistet, bearbeitet, gelöscht und mit anderen Nutzern der Anwendung geteilt werden. Auch ein Download der angehängten Dateien ist möglich.

Die Implementierung des Kryptosystems nutzt zur Verschlüsselung von Informationen und Dateien symmetrische Schlüssel mit dem Chiffre XSalsa20 in Kombination mit

[26] `https://dchest.github.io/tweetnacl-js/audits/cure53.pdf`

dem Authentifizierer Poly1305. Die dazu genutzten Schlüsselinformationen werden wiederum asymmetrisch mit den Schlüsseln der Benutzer gesichert. Damit ergibt sich ein hybrides Verschlüsselungsverfahren wie in Abschnitt 3.4.3 erläutert. Eine Anforderung der eingesetzten Schlüsselbibliothek ist zudem die Nutzung einer zufälligen Nonce für jede symmetrisch oder asymmetrisch zu sichernde Information. Auch müssen symmetrische Schlüssel stets eine feste Länge von 32 Byte aufweisen, weshalb zu diesem Zweck genutzte Passwörter entweder verkürzt oder mit zufälligen Bytefolgen vergrößert werden. Sowohl die generierte Nonce, eine nur einmalig verwendete zufällige Bytefolge, als auch gegebenenfalls hinzugefügte Zufallsbytes werden im Klartext an den Server gesendet und dort gespeichert. In den folgenden Unterabschnitten wird dies nicht explizit erwähnt, ist aber immer Teil einer genannten Schlüsselübertragung.

5.2.1. Strukturierung der Anwendung

Die geplante Struktur der Applikation weist eine strikte Trennung in sicherheitsrelevante Aufgabenbereiche auf. Die beiden im Rahmen des Projektes entstandenen Pakete sec-crypter und sec-keystore decken sämtliche sicherheitsrelevanten und kryptografischen Aufgaben ab. Funktionen, welche dazu aus der Hauptapplikation aufgerufen werden müssen, werden über die Package API exportiert. Die generische Entwicklung der Kryptofunktionen in Paketen erleichtert eine spätere Weiterentwicklung der Hauptanwendung und ermöglicht auch die Verwendung der implementierten Logik in weiteren Projekten.

Das Paket sec-crypter ist für die Durchführung der kryptografischen Funktionen im Browser zuständig. Es überwacht den Entschlüsselungsstand der vom Server gesendeten Informationen, fordert benötigte Schlüssel an und dechiffriert die Informationen bei Bedarf. Zur Verschlüsselung von Benutzern erfasster Informationen stellt es simple Funktionsaufrufe bereit und übernimmt im Hintergrund auch die Erzeugung neuer Schlüsselinformationen. Auch kümmert es sich um die Modifikation serverseitiger Datenschemata, um binäre Chiffrate in der Datenbank speichern zu können. Es fügt zusätzlich bei Speicherung eines Datensatzes diesem eine Auflistung der verschlüsselten Felder hinzu. Die Aufgabe des Paketes sec-keystore dagegen beschränkt sich auf die Verwaltung gespeicherter Schlüsselinformationen, wie im folgenden Abschnitt 5.2.2 beschrieben.

Aus einer Kombination von verschlüsselten Daten und den, ebenfalls als Chiffrat gespeicherten, Schlüsseln lassen sich dennoch Kommunikationsprofile erstellen. Daher sieht das Konzept der Applikation vor, dass sich das separate Paket sec-keystore mit geringem Zusatzaufwand in einer getrennten Applikation und damit auch auf einem anderen Server betreiben lässt. Das ermöglicht eine strikte physikalische Trennung von Daten und Schlüsselinformationen.

5.2.2. Verwaltung und Übertragung von Schlüsselinformationen

Eine der wichtigsten Anforderungen ist, dass Schlüsselinformationen nur dem jeweiligen Besitzer bekannt sein und niemals im Klartext übertragen werden dürfen. Deshalb wird bei der Registrierung eines Nutzers dessen privater Schlüssel direkt im Browser erzeugt. Der Server bekommt lediglich den zugehörigen öffentlichen Schlüssel im Klartext übertragen. Der eben erzeugte private Schlüssel wird in der authentifizierten Browsersession abgelegt, wodurch er beim Ausloggen des Benutzers zuverlässig gelöscht wird. Zudem wird ein mit dem Passwort des Nutzers erstelltes Chiffrat des Schlüssels im persistenten Storage des Browsers abgelegt und an den Server gesendet. Meldet sich der Benutzer mit einem anderen Browser oder von einem anderen Gerät an, ist der Server nun in der Lage, ihm seinen privaten Schlüssel zur Verfügung zu stellen, ohne diesen selbst im Klartext zu kennen. Zur sicheren Authentifizierung des Schlüsselbesitzers verschlüsselt der Server diesen noch einmal zusätzlich mit einem zufällig generierten Transmission Key, welcher dem Benutzer per E-Mail und damit über einen zweiten Kanal mitgeteilt wird. Der Browser des Benutzers ist nun in der Lage, mit Transmission Key und dem bereits zum Login eingegebenen Passwort, den privaten Schlüssel wiederherzustellen und erneut sowohl im Storage als auch in der authentifizierten Session abzulegen. Das im Vorfeld der Implementierung geplante Schema dieses Vorgangs zeigt Abbildung 4 im Anhang. Der private Schlüssel ist durch Nutzung des zweiten Kanals vor passiven und aktiven Attacken auf das Kryptosystem, etwa vor Passwortdiebstahl oder vor Man-in-the-Middle Angriffen, geschützt. Nach Erfüllung der Forderungen von Ivan Ristić an den Austausch geheimer Schlüssel ist ausreichend gewährleistet, dass nur der tatsächliche Besitzer des Schlüssels in der Lage ist, diesen an einem anderen Gerät wiederherzustellen. (Ristic 2014)

Festgehalten werden die Schlüsselinformationen der Benutzer von der Serveranwendung in UserPrincipal Objekten. Diese enthalten, wie in Abbildung 3 dargestellt, neben der Benutzer-ID den öffentlichen Schlüssel des Benutzers sowie den passwortgesicherten privaten Schlüssel mit der genutzten Nonce und gegebenenfalls zum Passwort hinzugefügten zufälligen Bytefolgen. Listing 1 im Anhang zeigt die serverseitige Schnittstelle, über welche die öffentlichen Schlüssel aller Anwender der Applikation angefragt werden können. Die Methoden zum Abruf privater Schlüsselinformationen geben jedoch, wie in Listing 2 zu sehen, nur Auskunft über die Schlüsselinformationen des aktuell angemeldeten Benutzers.

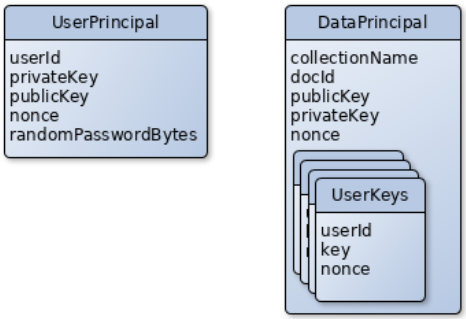

Abbildung 3: Principalobjekte zur Verwaltung von Schlüssel- und Zugriffsinformationen

Auch symmetrische Schlüssel zur kryptografischen Sicherung erfasster Informationen werden durch die Clientanwendung erzeugt. Diese werden in DataPrincipals verwaltet (siehe Abbildung 3). Ein solches enthält den Objekttyp (entspricht in der implementierten Anwendung dem Namen der MongoDB Collection), die Objekt-ID und die symmetrische Nonce. Hinzu kommt ein Schlüsselpaar, über welches die verschlüsselte Information mit Poly1305 signiert und bei der Entschlüsselung authentifiziert wird. Den eigentlichen Dokumentenschlüssel sichert der Ersteller asymmetrisch mit seinem öffentlichem Schlüssel und legt ihn mit seiner Nutzer-ID und der für die asymmetrische Verschlüsselung generierten Nonce in der Sammlung „userKeys“ des Principalobjektes ab. Dieses Principalobjekt wird zur Verwahrung an den Server gesendet und bei Bedarf von diesem wieder abgefragt. Wie in Listing 3 in der Methode getPrincipals gezeigt, stellt die Serverapplikation bei Anfragen zu DataPrincipals sicher, dass ein Benutzer nur Datensätze erhält, bei welchen er selbst einen Eintrag in der Liste der Benutzerschlüssel besitzt.

Wie in der Anforderungsliste aufgeführt, werden zu keinem Zeitpunkt geheime Schlüsselinformationen im Klartext übertragen. Auch ist es nicht möglich, die Dokumentenschlüssel der gesicherten Informationen ohne Kenntnis des zugehörigen privaten Benutzerschlüssels zu erlangen. Dieser wiederum wird durch die clientseitige Verschlüsselung mit dem Passwort des Nutzers vor unbefugtem Zugriff gesichert. Vor einer Übertragung zurück an den Benutzer wird der Schlüssel erneut durch einen zufälligen Transmission Key gesichert, welcher per E-Mail übertragen wird.

5.2.3. Behandlung von Informationen

Da in der Anwendung eingegebene Informationen weder im Klartext übertragen, noch für den Betreiber der Serverapplikation oder einen unberechtigten Dritten einsehbar sein sollen, müssen sämtlicher Ver- und Entschlüsselungsvorgänge automatisch in der Clientanwendung vorgenommen werden. Das Vorliegen verschlüsselter Daten im Binärformat Uint8Array anstatt als String oder Number erfordert aber auch serverseitige Anpassungen genutzter Schemadefinitionen. Um beidem ohne einen Monkey Patch der MongoDB Bibiliothek gerecht zu werden, verwenden Client und Server statt der üblichen Meteor.MongoCollection die im Paket sec-crypter definierte und von der Orginalklasse abgeleitete Meteor.CryptedCollection. Auf Serverseite sorgt diese dafür, dass Typinformationen für angewendete Datenschemata für verschlüsselte Felder auf das binäre Format gesetzt werden.

Bei der Erstellung eines Eintrags erzeugt die Anwendung im Browser, wie im vorhergehenden Abschnitt beschrieben, einen neuen DataPrincipal, aus welchem anschließend die symmetrischen Schlüsselinformationen entnommen werden. Alle Felder des Eintrags, welche für die Collection „Entries“ zur Verschlüsselung konfiguriert sind, werden mit dieser nun symmetrisch verschlüsselt. Nachdem der neue Eintrag an die Serveranwendung gesendet und dort gespeichert wurde, wird dem DataPrincipal die Dokument-ID des neuen Eintrags hinzugefügt. Das Principalobjekt wird im Anschluss ebenfalls an den Server gesendet. Wird ein vorhandener Eintrag modifiziert, so fordert der Client das existierende Principalobjekt für den Eintrag an und verwendet die Schlüsselinformationen aus diesem erneut.

Eine Eigenheit des verwendeten Applikationsframeworks Meteor ist, dass abonnierte MongoDB Collections bei Änderungen in der Datenbank automatisch im Client synchronisiert werden. Um damit bereits entschlüsselte Daten nicht wieder zu verlieren, macht sich die bereits genannte CryptedCollection ein Observersystem zunutze, welches ermöglicht, für jede Aktion (insert, delete, update) der Daten im Browser definierte Callbacks auszuführen. Innerhalb der CryptedCollection wird eine zweite, lokale Collection im Browser gepflegt in welche jede Aktualisierung der serverseitigen Daten migriert wird. Über eine Queue werden Einträge bei Lesezugriff in dieser lokalen Sammlung nacheinander entschlüsselt. Die Abarbeitung der Entschlüsselungsmethode über d3-queue zeigt Listing 4 im Anhang. Die lokale Collection dient damit auch, wie von Michel Floyd demonstriert, als Cache für bereits entschlüsselte Daten. (Floyd 2016) Benötigte Principalobjekte werden für jeden Entschlüsselungsvorgang von der Serveranwendung abgefragt. Für jeden entschlüsselten Eintrag wird auch eine Liste der entschlüsselten Felder geführt. Nach Update eines Datensatzes durch die Serveranwendung kann die Applikation auf diese Weise nachsehen, welche Felder bereits entschlüsselt vorliegen, um diese nicht erneut zu dechiffrieren.

5.2.4. Behandlung von Dateianhängen

Ebenso wie direkt erfasste Informationen beinhalten oft auch Dateianhänge sensible und schützenswerte Daten. Da es sich hier nicht um Informationen aus der Datenbank handelt, kann die Anwendung nicht von dem bereits genannten Observe-System profitieren. Das für den Upload genutzte Plugin meteor-files[27] bietet die Möglichkeit, nach dem Laden einer Datei von der Festplatte noch auf deren Inhalt zuzugreifen, bevor sie an den Server übertragen wird. Der Dateiinhalt liegt als Base64 kodierte DataURL vor. Listing 5 zeigt den Zugriff darauf. Die ID des Dateianhangs, welche der Server nach erfolgreichem Upload als Antwort zurückgibt, wird anschließend in die Daten des Eintrags übernommen, bevor auch dieser, wie vorhergehend beschrieben, verschlüsselt gespeichert wird. Da das Feld, welches die Anhangs-ID beinhaltet ebenfalls als verschlüsseltes Feld konfiguriert werden kann, ist es möglich, jede Verbindung von hochgeladenen Dateien zu deren Besitzern in der Datenbank zu verschleiern. Nur Benutzer, die den Eintrag entschlüsseln können, sind nun in der Lage, die zugehörigen Dateien herauszufinden.

[27] `https://atmospherejs.com/ostrio/files`

Komplexer umzusetzen ist der Download verschlüsselten Dateien. Leider gibt es keine Möglichkeit, die Standardbehandlung eines Dateidownloads im Browser zu beeinflussen. Daher muss die Datei zunächst manuell über einen XHR Request vom Server geladen werden. Es ist wichtig, für diese Anfrage den Antworttyp „blob“ zu nutzen, da in diesem Binärformat sämtliche wichtigen Metainformationen der Datei erhalten bleiben. Dies ermöglicht anschließend über einen FileReader das Auslesen des Dateityps und des verschlüsselten Inhalts als DataURL. Wie auch bei Informationen aus der Datenbank, muss auch für die Datei nun das zugehörige DataPrincipal angefordert werden um den Dokumentschlüssel daraus zu errechnen und die DataURL zu entschlüsseln. Mithilfe des Pakets file-saver[28] können die entschlüsselten Daten nun mit dem zuvor ausgelesenen Dateityp im Browser zum Download angeboten werden. Den gesamten Vorgang des Ladens und Entschlüsselns einer Datei vom Server zeigt die zu diesem Zweck implementierte React Komponente in Listing 6.

5.2.5. Teilen von Informationen mit anderen Benutzern

Das Teilen verschlüsselter Datensätze mit anderen Benutzern in der implementierten Applikation orientiert sich am Vorgehen des in Abschnitt 4.1.2 vorgestellten Anbieters Tresorit. Möchte Alice Informationen mit Bob teilen, so fordert sie dessen öffentlichen Schlüssel vom Server an. Mit diesem verschlüsselt sie den symmetrischen Dokumentenschlüssel des Datensatzes und fügt das entstandene Chiffrat gemeinsam mit Bobs Benutzer-ID und der verwendeten asymmetrischen Nonce dem DataPrincipal ihrer Information hinzu. Diesen neuen Schlüsseldatensatz sendet sie dem Server zur Aufbewahrung. Bob ist nun in der Lage, den von Alice erstellten Dokumentenschlüssel zu entziffern und für den Zugriff auf die geschützte Information zu nutzen.

Um nur Einträge mit Zugriffsberechtigung in einer Webapplikation anzuzeigen wird meist eine Datenbankabfrage selektiv eingeschränkt, etwa auf den Ersteller des Datensatzes oder über eine Mappingtabelle zur Verwaltung von Zugriffen für mehrere Benutzer. Beides ist für das entstandene Konzept mit zusätzlichen Schlüsselinformationen in einem Principalobjekt nicht mehr sinnvoll. Es sind nicht länger Metainformationen der Hauptanwendung dafür zuständig, den Zugriff auf Informationen zu autorisieren. Für

[28] `https://www.npmjs.com/file-saver`

die Lesbarkeit eines Objektes ist entscheidend, ob ein Nutzer einen Schlüsseleintrag im zugehörigen Sicherheitsprinzipal besitzt oder nicht. Daher greift die Applikation zur Auflistung der Einträge nicht auf die MongoDB Collection der Einträge selbst zurück. Statt dessen wird, wie in der Methode userDocuments in Listing 3 demonstriert, aus gespeicherten DataPrincipals eine Liste von Einträgen erstellt, für die der Benutzer einen Schlüssel besitzt. Auch die Information, wer Zugriff auf einen bestimmten Eintrag hat, wird nicht aus Informationen der Anwendung selbst gewonnen. Die Methode getPermittedUsers des selben Listings zeigt die Erstellung einer Liste von Benutzer-IDs mit einem Schlüsseleintrag für ein bestimmtes Datenobjekt. In beiden Fällen wird stets darauf geachtet, keine Schlüsselinformationen anderer Benutzer aus der Datenbank des Keystore zu laden. Diese können damit auch nicht unbefugt versendet werden. Auch erstellen beide Abfragen einen reaktiven Cursor, der durch die Echtzeitsynchronisierung von Meteor neue Einträge sofort in den Auflistungen der Clientsoftware hinzufügt.

6. Diskussion der unterschiedlichen Lösungen

Nach der Beschreibung implementierter Verschlüsselungsverfahren unterschiedlicher Anbieter und Ausführungen zu dem selbst entwickelten Ansatz werden diese im Folgenden abschließend mit Blick auf Erreichung der Schutzziele aus Abschnitt 2.2 bewertet. Anschließend folgt ein Fazit zur Gewährleistung der Vertraulichkeit von Informationen in der entstandenen Implementierung im Vergleich zu den beschriebenen Lösungen am Markt.

6.1. Bewertung der bestehenden Lösungen

Bei Betrachtung der Dateihoster in Abschnitt 4.1 wird deutlich, dass kommerzielle Anbieter eine sichere Informationsverschlüsselung nur anbieten, wenn dies im Sinne ihres Geschäftsmodelles ist. Bei Google Drive etwa ist zu vermuten, dass der Benutzer mit dem Einstellen seiner Daten für die Nutzung des kostenfreien Dienstes bezahlt. An diesen räumt sich der Anbieter wie dargestellt weitgehene Rechte ein. Um Informationen hier dennoch vor dem Zugriff der Anbieter und Dritter zu schützen, ist der Einsatz

weiterer Software nötig. Boxcryptor [29] beispielsweise wurde speziell zu dem Zweck entwickelt, in Cloudspeichern abgelegte Daten noch auf der eigenen Festplatte zu ver- und entschlüsseln und damit vor fremdem Zugriff zu schützen. Damit ist auch bei der Verwendung der unverschlüsselten Angebote Google Drive und Dropbox die Vertraulichkeit und Integrität der übermittelten Daten gewährleistet.

Tresorit dagegen ist ganz klar als Positivbeispiel zu sehen. Der Schutz der Vertraulichkeit eingestellter Informationen ist hier nicht nur ein Werbeversprechen, sondern wird als Kernfunktion der eigenen Plattform gesehen. Daraus resultiert ein solide geplantes Kryptosystem. Zwar ist die gleichzeitige Speicherung der kryptografisch gesicherten Dokumente mit den zugehörigen Schlüsselinformationen auf dem System des Anbieters problematisch, da die Schlüssel jedoch ebenfalls vor dem Versand verschlüsselt werden, hat Tresorit keine Möglichkeit, die hochgeladenen Daten in lesbarer Form zu verarbeiten. Durch das Verbleiben der privaten Schlüssel in der Clientanwendung ist die Vertraulichkeit der Informationen sichergestellt. Der Schlüsseltausch über die eigene Public-Key-Infrastruktur und das „ICE-Protokoll“ sichert zudem die Identität und Authentizität weiterer zugriffsberechtigter Nutzer. Mit der zusätzlichen Erstellung eines Message Authentication Codes kann ferner die Integrität der gespeicherten Daten verifiziert werden. (Tresorit AG 2013) Daraus erwächst dennoch ein Nachteil für diesen Anbieter. Im Gegensatz zu seiner Konkurrenz kann Tresorit auf Serverseite etwa keine Vorschaubilder für hochgeladene Dokumente produzieren. Wo benötigt müssen Derivate der Informationen vor dem Upload ebenfalls in der Clientanwendung erzeugt werden.

Auch die Messaging-Software WhatsApp erreicht durch die Implementierung des Signal Protokolls ein Maximum an durch Kryptografie erreichbarer Sicherheit. Mit komplexen Verfahren zum Schlüsseltausch, der Vorratshaltung authentifizierter Schlüssel zum Kommunikationsaufbau sowie dem häufigen Auswecheln der Schlüsselinformationen bietet sie nur sehr wenige Möglichkeiten für Angriffe auf das Kryptosystem. Die Kenntnis eines einzelnen Nachrichtenschlüssels ermöglicht, wie in Abschnitt 4.2 beschrieben, keine Rückschlüsse auf die Schlüssel weiterer Nachrichten. Gerade aus Verfahren wie dem Double Ratchet Algorithmus entsteht jedoch ein hoher Aufwand an Erzeugung, Validierung und Verwaltung kryptografischer Schlüssel. Nach Vorgabe von Libsignal erzeugt und verwirft WhatsApp weit mehr Schlüsselinformationen als die anderen

[29] `https://www.boxcryptor.com`

vorgestellten Implementierungen. Moderne Smartphones verfügen zwar über ausreichend Rechenleistung, um diese Operationen ohne spürbare Leistungseinbußen vorzunehmen, vor einer Adaption dieses Protokolls in die eigene Anwendung sollte man sich über den entstehenden Aufwand jedoch im Klaren sein.

Ein vielversprechendes Konzept beschreibt auch die am MIT entstandene Forchungsplattform Mylar. Die gegenseitige Authentifizierung und der Austausch von Schlüsselinformationen sind darin ähnlich geregelt, wie es auch für die properitäre Anwendung Tresorit beschrieben wird. Vertraulichkeit, Integrität und Authentizität der Informationen ist mit Blick auf die Informationen bereits gewährleistet. Der Ansatz geht aber noch einen Schritt weiter und signiert auch die Schlüssel der Benutzer durch ein fest eingebautes Zertifikat oder einen unabhängigen Identitätsprovider. Das Verifizieren des signiert an den Browser gelieferten Programmcodes schließlich sichert auch eine vollständige Integrität der Anwendung selbst. Dazu ist allerdings der Betrieb weiterer Applikationsserver vonnöten, da diese unterschiedlichen Aufgaben von getrennten Systemen vorgenommen werden müssen. Der erhöhte administrative Aufwand ist durch den Zugewinn an Sicherheit sicherlich gerechtfertigt. Dagegen stellt das zur Prüfung der Signaturen bei Benutzerschlüsseln und Programmcode nötige Browserplugin eine deutliche Einschränkung des potentiellen Nutzerkreises dar. Es muss gut überlegt sein, ob die angestrebte Zielgruppe bereit ist, dieses zur Nutzung der Anwendung zu installieren und ob alle benötigten Browser unterstützt werden. Auch, wenn der entstandene Programmcode der Forschungsplattform sich mit wenigen Änderungen am Orginalprogramm in bestehende Anwendungen einbinden lässt, ist von einer direkten Übernahme für produktive Umgebungen abzuraten. Die entstandene Software wird auf der Mylar Webseite[30] ausdrücklich als experiementell gekennzeichnet. Die letzte Änderung am GIT Coderepository des Projekts[31] liegt bereits über drei Jahre in der Vergangenheit. Es ist nicht zu erwarten, dass das Projekt fortgeführt wird. Daher sollte es eher als Inspirationsquelle und Machbarkeitsstudie für die Implementierung eines verschlüsselten Informationsaustauschs direkt in der eigenen Anwendung gesehen werden.

Abschließend kann festgestellt werden, dass sowohl der Ansatz von Mylar, wie auch die Verschlüsselung von Tresorit und WhatsApps Implementierung des Signal-Protokolls

[30] `http://g.csail.mit.edu/mylar`
[31] Mylar Repository: `git://g.csail.mit.edu/mylar`

einen vollständigen Schutz im Sinne der vorgestellten Schutzziele (siehe Abschnitt 2.2) bieten. Bei allen drei Lösungen sind Integrität und Vertraulichkeit der Informationen zu jeder Zeit gewährleistet. Auch die Authentizität von Kommunikationspartnern und den eingestellten Informationen ist gesichert. Lediglich die Marktführer auf dem Bereich der Onlinespeicher, Google Drive und Dropbox, lassen jede Form kryptografischer Informationssicherung vermissen.

6.2. Bewertung der entstandenen Implementierung

Der entstandene Programmcode widmet sich, wie eingangs erwähnt, primär der Vertraulichkeitssicherung übermittelter Daten. Zu diesem Zweck ist der Ansatz von Mylar deutlich überdimensioniert. Weder die Authentizität von Benutzern und Informationen noch die Integritätssicherung von Informationen oder gar der ganzen Applikation ist in den Anforderungen gelistet. Die Libsignal Implementierung von WhatsApp bietet zwar gute Konzepte für den Schlüsseltausch, diese funktionieren aber nur mit genau einer Gegenseite oder, im Falle von Gruppen, mit einem vor Erfassen der Nachricht bekanntem Nutzerkreis. Für eine klassische Chatanwendung ist dieser Ansatz perfekt geeignet.

Am besten vergleichbar ist der enstandene Ansatz mit der Lösung von Tresorit. Für jede Information wird ein Objekt mit individuell verschlüsselten Sicherheitsinformationen für jeden zugriffsberechtigten Nutzer verwaltet. Das serverseitige Register verschlüsselter Felder pro Datensatz ermöglicht spätere Änderungen, wie die Aufnahme weiterer Felder in die Verschlüsselung, ohne mit alten Datensätzen auf Fehler bei der Entschlüsselung zu stoßen. Eine Anlehnung an Mylar stellt die Arbeitsteilung der entstandenen Pakete dar. Während alle clientseitigen kryptografischen Aufgaben und die Verwaltung der verschlüsselten Informationen von dem Paket „sec-crypter“ vorgenommen werden, werden sämtliche Prinzipale mit den Schlüsselinformationen von dem zweiten Paket „sec-keystore“ verwaltet. Das Konzept sieht die Möglichkeit vor, den Keystore mit geringem Programmieraufwand vollständig von der Hauptapplikation abzutrennen und in einer eigenständigen Applikation laufen zu lassen. Die Kommunikation der beiden Pakete untereinander ist darauf bereits ausgelegt. Jedoch findet der Abruf der

Schlüssel aus dem Browser immer bei der Hauptapplikation statt, die quasi als Proxy die benötigten Informationen aus dem Keystore durchleitet.

Eine wichtige Anforderung für eine Applikation mit großem Benutzerkreis ist auch die bequeme Nutzung an unterschiedlichen Geräten. Dazu benötigen alle Geräte Zugriff auf den privaten Schlüssel des Benutzers. Die vorliegenden Ansätze von Tresorit und Mylar verwahren diesen durch das Passwort des Benutzers symmetrisch gesichert auf den Servern der Applikation. Bei der Anmeldung an einem anderen Gerät wird er in den neuen Browser übertragen und dort entschlüsselt. Auch wenn das vorrangige Ziel der Implementierung keine Authentifizierung der Benutzer enthält, scheint diese bei Aushändigung des privaten Schlüssels von höchster Bedeutung zu sein. Aus diesem Grund verschlüsselt das Paket „sec-keystore“ den Schlüssel vor Herausgabe noch einmal mit einem zufallsgeneriertem Passwort. Neben seinem eigenen Nutzerpasswort benötigt der Anwender auch dieses, per E-Mail übermittelte Geheimnis, um seinen privaten Schlüssel wiederherzustellen. Damit wird in diesem, für die Vertraulichkeit der Informationen als hoch kritisch anzusehenden Schritt, die Authentizität des Nutzers über einen zweiten Kanal verifiziert.

7. Zusammenfassung

Die verschlüsselte Speicherung eingegebener Informationen in einer Webanwendung nach dem Zero-Knowledge-Prinzip bringt für die Planung und Implementierung der Anwendung einen nicht unerheblichen Mehraufwand mit sich. Wie die Ausführungen zur Begründung dieses Aufwandes gezeigt haben, kann daraus je nach Zielgruppe ein Wettbewerbsvorteil erwachsen. In einigen Bereichen wäre es dagegen sogar grob fahrlässig, auf eine Vertraulichkeitssicherung der Informationen zu verzichten. Ein Anbieter im E-Health-Bereich etwa kann das Risiko nicht eingehen, dass hochsensible persönliche Informationen seiner Nutzer ohne deren Einwilligung weitergegeben werden.

Es hat sich gezeigt, dass bereits heute einige Anbieter, sofern es zu den eigenen Geschäftszielen passt, gute Konzepte zur Abdeckung aller kryptografisch erreichbaren Schutzziele implementiert haben. Um die Informationssicherung im Internet voranzubringen, werden die dahinter liegenden Protokolle und Quellcodes zum Teil bewusst offengelegt. Dies

zeigt das Beispiel der Signal Bibliothek und des Double Ratchet Algorithmus von Open Whisper Systems. Daneben geben Forschungsplattformen wie Mylar wichtige Anregung zur Implementierung eigener Lösungen.

Wie die im Rahmen dieser Arbeit entstandene Implementierung gezeigt hat, kann unter Einsatz bestehender Kryptobibliotheken auch ein eigenes Kryptosystem für eine Applikation entworfen und implementiert werden. Bei der Wahl des Verschlüsselungsverfahrens hat sich ein hybrider Ansatz als vorteilhaft herausgestellt. Durch symmetrische Verschlüsselung der Informationen ist die Vertraulichkeit der Informationen gesichert. Die anschließende asymmetrische Sicherung des genutzten symmetrischen Schlüssels ermöglicht dessen Hinterlegung in der Serveranwendung, ohne dem Server sowie dem Anbieter einen Zugriff auf den Schlüssel selbst zu geben. Für die Durchführung kryptografischer Aufgaben wurde bewusst eine in JavaScript geschriebene Bibliothek genutzt, anstatt auf die Web Crypto Api nach der Empfehlung des W3C zurückzugreifen. Diese wird zwar bereits von vielen Browsern unterstützt, doch schwankt die Qualität der einzelnen Implementierungen noch zu stark für einen produktiven Einsatz.

Neben der Bedeutung der Informationssicherheit für heutige Webanwendungen zeigt diese Arbeit damit auch, dass die Vertraulichkeit gespeicherter Informationen mit ausreichend Planung in einem selbst implementierten System gewährleistet werden kann.

Spätestens die Übertragung der sicherheitskritischen Kryptobibliothek als JavaScript Datei an den Browser macht auch eine Sicherung der Integrität dieser Bibliothek zu einem wichtigen Thema. Der Überprüfung der Integrität nachgeladener Assets widmet sich etwa die Empfehlung „Subresource Integrity“ (SRI) des W3C aus dem Juni 2016[32]. Diese wird von den meisten wichtigen Browsern unterstützt und kann bereits in der Praxis eingesetzt werden[33]. Dennoch ist seitens der Browserhersteller, Frameworkmaintainer, CDN-Anbieter und nicht zuletzt Webseitenbetreiber noch viel Arbeit nötig, um das Konzept SRI mit Mehrwert für sichere Anwendungen umzusetzen. Weitergehende Forschungen zur optimalen Umsetzung fehlen bisher.
Auch die Übertragung der privaten Schlüsselinformation eines Nutzers auf andere Geräte ist nicht unproblematisch. Gesichert durch das Nutzerpasswort befindet sich der

[32] `https://www.w3.org/TR/SRI/`
[33] https://r-n-d.informatik.hs-augsburg.de:8080/quest/talk-sri/

Schlüssel bei den meisten Lösungen auf den Systemen des Anbieters und kann mit ausreichend Rechenkapazität wiederhergestellt werden. Hier wäre eine anbieterunabhängige Übertragung von Vorteil. Für weitere Forschungen bietet sich daher auch die Evaluierung anbieterunabhängiger Authentifikationsdienste wie OpenID oder die Nutzung des elektronsichen Personalausweises zur benutzerfreundlichen Schlüsselerstellung und Schlüsselübertragung an.

Literaturverzeichnis

Agrawal, Rakesh u. a. (2004). „Order preserving encryption for numeric data“. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*. ACM Press. DOI: 10.1145/1007568.1007632.

BVerfG (2008). *Urteil des Ersten Senats vom 27. Februar 2008*. URL: http://www.bverfg.de/e/rs20080227_1bvr037007.html (zuletzt geprüft am 16. 05. 2017).

Chen, Yanpei, Vern Paxson und Randy H. Katz (2010). *What's New About Cloud Computing Security?* Techn. Ber. UCB/EECS-2010-5. EECS Department, University of California, Berkeley. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html.

Dropbox Inc. (2016). *Dropbox - Geschäftsbedingungen*. URL: https://www.dropbox.com/de/terms (zuletzt geprüft am 19. 05. 2017).

Floyd, Michel (2016). *Using Meteor Local Collections to Support Browser-Based Cryptography*. https://themeteorchef.com/tutorials/using-meteor-local-collections-to-support-browser-based-cryptography. URL: https://themeteorchef.com/tutorials/using-meteor-local-collections-to-support-browser-based-cryptography (zuletzt geprüft am 19. 06. 2017).

Fröschle, Hans-Peter (2011). *IT-Sicherheit & Datenschutz*. Dpunkt.Verlag GmbH. 128 S.

Gola, Peter u. a. (2015). *BDSG*. Beck C. H.

Google Inc. (2013). *Google Nutzungsbedingungen*. URL: https://www.google.de/policies/terms/regional.html (zuletzt geprüft am 19. 05. 2017).

Gultsch, Daniel (2016). *The State of Mobile XMPP in 2016*. URL: https://gultsch.de/xmpp_2016.html (zuletzt geprüft am 03. 06. 2017).

Karlinger, Michael, Klaus Ettmayer und Michael Schrefl (2011). „Verschlüsselung bei ausgelagerter Datenhaltung“. In: *IT-Sicherheit & Datenschutz*. Dpunkt.Verlag GmbH.

Mayer, Jonathan, Patrick Mutchler und John C. Mitchell (2016). „Evaluating the privacy properties of telephone metadata“. In: *Proceedings of the National Academy of Sciences* 113.20, S. 5536–5541. DOI: 10.1073/pnas.1508081113.

Meinel, Christoph und Harald Sack (2009). *Digitale Kommunikation*. Springer-Verlag GmbH.

Popa, Raluca Ada u. a. (2016). *Building web applications on top of encrypted data using Mylar*. Cryptology ePrint Archive, Report 2016/893. http://eprint.iacr.org/2016/893.

Ptacek, Thomas (2011). *Javascript Cryptography Considered Harmful*. URL: https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/ (zuletzt geprüft am 04. 06. 2017).

Ristic, Ivan (2014). *Bulletproof SSL and TLS*. FEISTY DUCK. 550 S.

Schneider, Jochen (2013). „Datenschutzrechtliche Anforderungen an die Sicherheit der Kommunikation im Internet“. In: *Daten- und Identitätsschutz in Cloud Computing, E-Government und E-Commerce*. Springer-Verlag GmbH.

Tresorit AG (2013). *Tresorit White Paper*. Techn. Ber. Tresorit AG. URL: `https://tresorit.com/files/tresoritwhitepaper.pdf` (zuletzt geprüft am 26. 05. 2017).

– (2015). *Tresorit Terms and Conditions of Use*. URL: `https://tresorit.com/terms-of-use` (zuletzt geprüft am 26. 05. 2017).

Voigtländer, Daniel und Thomas Schmischke (2011). „Der Datenschutzbeauftragte“. In: *IT-Sicherheit & Datenschutz*. Dpunkt.Verlag GmbH.

WhatsApp Inc. (2016). *WhatsApp Encryption Overview. Technical white paper*. URL: `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf` (zuletzt geprüft am 31. 05. 2017).

A. Anhang

A.1. Übersicht im Projekt eingesetzter Bibliotheken

- **meteor:** Full JS Web Application Framework mit integrierter Benutzerverwaltung und Echtzeitaktualisierung übertragener Informationen
- **react:** Komponentenbasierte UI-Entwicklung für Webapplikationen
- **tweetnacl:** Auditierte JavaScript Implementierung der NaCl (Kochsalz) Crypto Bibliothek
- **simpl-schema:** Definition von Schema-Vorgaben für Datenstrukturen
- **aldeed:collection2-core:** Anwenden von Schema-Definitionen auf MongoDB-Collections mit automatischer Validierung bei Insert und Update
- **matb33:collection-hooks:** Hook-Erweiterung, um lesende und schreibende Datenbankzugriffe zu beeinflussen und darauf zu reagieren
- **u2622:persistent-session:** Erweiterung der Browsersession in Meteor-Applikationen, um Informationen kontrolliert bis zum Logout oder unbegrenzt vorzuhalten
- **ostrio:files:** Meteor-Plugin zur Verwaltung von Fileuploads
- **file-saver:** Browserseitig generierte Informationen als Dateidownload anbieten
- **mobx:** State- und Datestore nach dem Observable Pattern mit Unterstützung berechneter Werte
- **mobx-react:** Component Wrapper, der React Komponenten aktualisiert, wenn sich benutzte Observables ändern
- **d3-queue:** Abarbeitung rechenintensiver Aufgaben über eine Warteschlange
- **semantic-ui:** UI Framework zur Erstellung responsiver Webseiten mit semantisch sinnvoller HTML-Struktur
- **semantic-ui-react:** React Komponenten für Semantic UI
- **randomstring:** Erstellung zufällig generierter Passwörter zur Absicherung der Schlüsselübertragung über einen zweiten Kanal
- **kadira:flow-router:** Router zum Mapping von URLs auf bestimmte Seiten / Komponenten
- **accounts-password:** Meteoreigenes Paket für Benutzeraccounts mit Passwortauthentifizierung
- **check:** Serverseitige Typevalidierung für aus dem Client übergebene Parameter
- **underscore:** Utility Library
- **email:** Versand von E-Mails mit Meteor
- **jcbernack:reactive-aggregate:** Zugriff auf die MongoDB Aggregation Pipeline mit reaktiv aktualisierndem Cursor

A.2. Anforderungen an die Implementierung einer Ende-zu-Ende gesicherten Kommunikation in Webanwendungen

1. Die Applikation muss für den Massenmarkt geeignet sein.
2. Es dürfen keine Schlüssel im Klartext übertragen werden.
3. Der private Schlüssel darf nur dem jeweiligen Eigentümer bekannt sein.
4. Neue Elemente werden in Echtzeit auf der Seite angezeigt.
5. Benutzer können verschlüsselte Einträge anlegen.
6. Zu jedem Eintrag kann es einen verschlüsselten Dateianhang geben.
7. Einträge können mitsamt Anhang mit anderen Benutzern geteilt werden.
8. Einträge können nachträglich bearbeitet werden.
9. Dateianhänge können nachträglich angefügt oder ersetzt werden.
10. Schlüssel und Sicherheitsinformationen werden in Principals gespeichert.
11. Sicherheitsinformationen der Benutzer werden als User Principal gespeichert.
12. Sicherheitsinformationen für Datenobjekte werden als Data Principal gespeichert.
13. Die Verarbeitung und Speicherung der Principals kann physisch von der Applikation abgekoppelt werden.
14. Registrierte Benutzer können sich von jedem Computer anmelden, ohne ihren privaten Schlüssel selbst einspielen zu müssen.
15. Der Versand eines privaten Benutzerschlüssels erfolgt nur an den tatsächlich registrierten Nutzer selbst.

A.3. Abgeleitete Konzepte für die Implementierung der Beispielanwendung

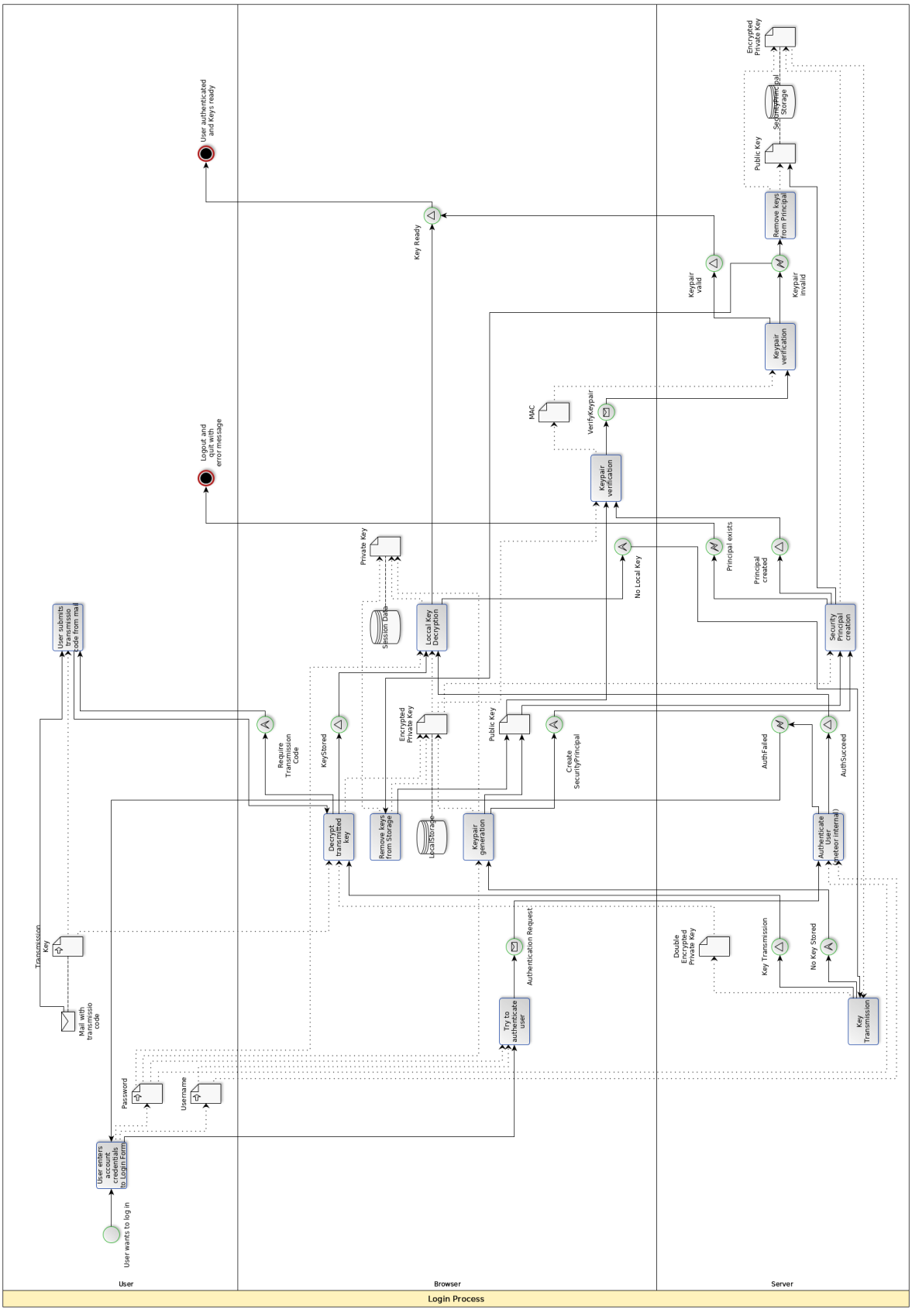

Abbildung 4: Schlüsselverwaltung bei Registrierung und Login

A.4. Codeauszüge aus der implementierten Anwendung

Die folgenden Listings zeigen Codeauszüge aus der im Rahmen der Ausarbeitung entstandenen Applikation. Dies stellt nur einen kleinen Teil der gesamten Implementierung dar. Die vollständige Implementierung befindet sich auf dem GitLab Server der Hochschule Augsburg[34].

```
1  /**
2   * Get the public key of a specific user, encrypted for the user
    that is currently signed on
3   * @param username Username of the specific user
4   * @returns {{encryptedPubkey, nonce, publicKey}|{encryptedPubkey:
    *, nonce: *, publicKey: *}}
5   */
6  'sec-crypter.user.publickey': function (username) {
7    check(username, String);
8    const requestedUser = Meteor.users.findOne({ username }, { fields:
    { _id: 1 } });
9    // get encrypted public key of user from keystore
10   const pubkey = UserPrincipalTools.publicKey(this.userId,
    requestedUser);
11   return { _id: requestedUser._id, key: pubkey };
12 },
```

Listing 1: Meteor Methode zur Abfrage von öffentlichen Schlüsseln nach Benutzername

```
1  'sec-crypter.data.principal.getPrincipal': function (collectionName,
     docId) {
2    check(collectionName, String);
3    check(docId, Match.OneOf(String, [String]));
4    return DataPrincipalTools.getPrincipals(this.userId,
    collectionName, docId);
5  },
6  'sec-crypter.data.principal.getSecret': function (collectionName,
    docId) {
7    check(collectionName, String);
8    check(docId, Match.OneOf(String, [String]));
9    return DataPrincipalTools.getPrincipalSecret(this.userId,
    collectionName, docId);
10 },
```

Listing 2: Serverseitige Einschränkung auf ID des Nutzers bei sensitiven Informationen

[34] https://r-n-d.informatik.hs-augsburg.de:8080/quest/webapp-gesicherte-kommunikation/

```
1  /**
2   * Save DataPrincipal record created on client side for a new
    document
3   * @param principal pre-generated principal, usually sent by client
4   * @returns {Promise}
5   */
6  static create(principal) {
7    // ensure there is no other principal for this document
8    if (DataPrincipals.findOne({ docId: principal.docId })) {
9      throw new Meteor.Error(
10       `Cant create DataPrincipal for Document ${principal.docId}.
11       Document already has a principal.`
12     );
13   }
14   return DataPrincipals.insert(principal);
15 }
16
17 /**
18  * Add a new encrypted document key for a user to a document
    principal
19  * The used query ensures, that the user that attempts to add the
    key (callerId) has himself
20  * an encrypted key in the principal
21  * @param callerId User that adds the user key
22  * @param collectionName
23  * @param docId
24  * @param userKey New User Key to be added
25  */
26 static addUserKey(callerId, collectionName, docId, userKey) {
27   return DataPrincipals.update({
28     collectionName, docId,
29     userKeys: { $elemMatch: { userId: callerId } },
30   },
31     {
32       $push: {
33         userKeys: userKey,
34       },
35     });
36 }
37
38 /**
39  * Remove a user encrypted document key if the caller hisself owns a
    user key for this doc
40  * @param callerId ID of the calling user
41  * @param collectionName
42  * @param docId
43  * @param userId ID of user to remove from principal
```

```
44      * @returns {any}
45      */
46     static removeUserKey(callerId, collectionName, docId, userId) {
47       // count keys, at least one must remain in keystore
48       const count = DataPrincipals.aggregate([
49         { $match: { collectionName, docId }},
50         { $project: { count: { $size: '$userKeys' } } },
51       ])[0].count;
52       if (count < 2) {
53         throw new Meteor.Error('Cannot remove the last user from the key
       list');
54       }
55       return DataPrincipals.update({
56         collectionName, docId,
57         userKeys: { $elemMatch: { userId: callerId } },
58       }, {
59         $pull: { userKeys: { userId } },
60       }, { multi: true });
61     }
62
63     /**
64      * Return cursor to a collection of document IDs the user has access
        to
65      * (owns a encrypted document key)
66      * @param userId
67      * @param collectionName (optional)
68      * @returns {Cursor}
69      */
70     static userDocuments(userId, collectionName) {
71       const query = { userKeys: { $elemMatch: { userId } } };
72       if (!!collectionName) {
73         query.collectionName = collectionName;
74       }
75       return DataPrincipals.find(query, { fields: { docId: 1,
      collectionName: 1 } }).fetch();
76     }
77
78     /**
79      * Return encrypted document keys for the requested document /
       documents
80      * @param userId
81      * @param collectionName
82      * @param docId single ID or array of IDs
83      * @returns {Cursor}
84      */
85     static getPrincipals(userId, collectionName, docId) {
86       const query = {
87         collectionName,
```

```
88      };
89      const projection = {
90        fields: {
91          docId: 1, collectionName: 1, publicKey: 1, privateKey: 1,
      nonce: 1,
92          userKeys: { $elemMatch: { userId } },
93        },
94      };
95      if (!!docId) {
96        if (_.isArray(docId)) {
97          query.docId = { $in: docId };
98        } else {
99          query.docId = docId;
100       }
101     }
102     return DataPrincipals.find(query, projection).fetch();
103   }
104
105   /**
106    * Returns a cursor to all user ids of users with encrypted document
        keys for the given document
107    * @param collectionName
108    * @param docId
109    * @returns {Cursor}
110    */
111   static getPermittedUsers(collectionName, docId) {
112     return DataPrincipals.aggregate([
113       { $match: { collectionName, docId } },
114       { $unwind: '$userKeys' },
115       { $project: { 'userKeys.userId': 1, _id: 0 } },
116       { $lookup: { from: 'users', localField:'userKeys.userId',
      foreignField: '_id', as: 'user' } },
117       { $unwind: '$user' },
118       { $project: { _id: '$user._id', username: '$user.username' } },
119     ]);
120   }
```

Listing 3: Kontrollierter Zugriff auf DataPrincipals im Keystore

```
1  static async decryptObject(object, principal) {
2    const decryptedDocumentKey = CollectionTools.
       usersDocumentKeyFromPrincipal(principal);
3    // Contains crytpted fields?
4    if (!_.isUndefined('_crypt', object)) {
5      // decryption method used for this object
6      const decrypter = (value) =>
```

```
7         CryptHelper.sym.decrypt(value, principal.nonce,
      decryptedDocumentKey);
8       return new Promise(resolve => {
9         this.decryptionQueue.defer(this._objectDecryptionWorker, object,
       decrypter, resolve);
10      });
11    }
12    return Promise.resolve(object);
13  }
14
15  // Method, that decrypts cipher fields in a dataObject using a given
      decryption method
16  // Resolves a Promise afterwards and calls the q3-queue callback to
      continue the queue
17  static _objectDecryptionWorker(dataObject, decrypter, resolve,
      callback) {
18    const encryptionConfig = dataObject._crypt;
19    const decryptedFieldsRegister = dataObject._decryptedFields || [];
20    // Object crawler, that loops all fields configured as encrypted and
       calls the decryption method on them
21    CollectionTools._crawlObjectForDecryption(
22    dataObject, encryptionConfig, decryptedFieldsRegister, '', decrypter
23    );
24    _.extend(dataObject, { _decryptedFields: decryptedFieldsRegister });
25    resolve(dataObject);
26    callback(null);
27  }
```

Listing 4: Entschlüsselung einzelner Felder über eine d3-queue

```
1   function uploadAttachment(entry) {
2     if (!!entry.attachment) {
3       // delete old attachment first
4       Meteor.call('entries.attachments.delete', entry.attachment);
5     }
6     // Read file content from disk
7     const uploader = EntryAttachments.insert({
8       file: attachment,
9       streams: 'dynamic',
10      chunkSize: 'dynamic',
11    }, false);
12    uploader.pipe(plainContent => {
13      // create principal and encrypt content (converted back from
      dataURI by atob)
14      const { principal, encryptedData } =
```

```
15        Crypt.Collection.encryptDataStreamWithNewPrincipal('entries.
      attachment', atob(plainContent));
16      uploader.on('end', (err, attachmentEntry) => {
17        if (err) {
18          console.error(err);
19          return;
20        }
21        _.extend(entry, {attachment: attachmentEntry._id});
22        Crypt.Document.savePrincipal(principal, attachmentEntry).then(()
      => {
23            saveEntry(entry);
24        });
25      });
26      // convert encrypted data back to dataURI using btoa
27      return btoa(encryptedData);
28    }).start();
29  }
```

Listing 5: Verschlüsseln einer Datei vor dem Upload

```
1  import React from 'react';
2  import { Button, Card, Icon } from 'semantic-ui-react';
3  import { saveAs } from 'file-saver';
4  import Crypt from 'meteor/spicyweb:sec-crypter';
5
6  class DownloadFile extends React.Component {
7    constructor(props) {
8      super(props);
9      this.state = {
10       fileContentPlain: null,
11       fileAsDataURL: null,
12       fileType: null,
13       decrypted: false,
14     };
15   }
16
17   async decryptFile(fileId) {
18     const { collection } = this.props;
19     return new Promise((resolve, reject) => {
20       // convert content from datauri to ciphertext and pass to
      decrypter
21       Crypt.Collection.decryptDataStreamFromCollection(
22         collection, fileId,
23         atob(this.state.fileAsDataURL))
24         .then(decryptedContent => {
```

```
25            this.setState({ decrypted: true, fileContentPlain: btoa(
    decryptedContent) });
26            resolve(decryptedContent);
27          });
28      });
29    }
30
31    async downloadAndSave(url, id, name, type) {
32      await this.downloadFile(url)
33        .then(() => this.decryptFile(id))
34        .then(() => this.saveFile(name, type));
35      return Promise.resolve();
36    }
37
38    async downloadFile(url) {
39      const loader = new XMLHttpRequest();
40      const fr = new FileReader();
41      return new Promise((resolve) => {
42        loader.responseType = 'blob';
43        loader.onload = () => {
44          const fileBlob = loader.response;
45          fr.onload = (e) => {
46            const arrContent = e.target.result.split(',');
47            const fileType = arrContent[0].match(/:(.*?);/)[1];
48            const fileAsDataURL = arrContent[1];
49            this.setState({ fileType, fileAsDataURL });
50            resolve(fileAsDataURL);
51          };
52          fr.readAsDataURL(fileBlob);
53        };
54        loader.open('GET', url, true);
55        loader.send();
56      });
57    }
58
59    async saveFile(name, type) {
60      const u8content = this._dataUri2Arraybuffer(this.state.
    fileContentPlain);
61      const backToBlob = new Blob([u8content], { type });
62      saveAs(backToBlob, name, false);
63      return Promise.resolve();
64    }
65
66    _dataUri2Arraybuffer(dataurl) {
67      const bstr = atob(dataurl);
68      let n = bstr.length;
69      const u8arr = new Uint8Array(n);
70      while (n--) {
```

```
71        u8arr[n] = bstr.charCodeAt(n);
72      }
73      return u8arr;
74    }
```

Listing 6: Ausschnitt aus Komponente zum Laden und Entschlüsseln einer Datei vom Server

Hochschule
Augsburg University of
Applied Sciences

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

_____ _____

Ort, Datum Unterschrift des/der Studierenden