Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

Konzeption und Entwicklung eines erweiterbaren Passwortmanagers als Cross-Platform Open-Source-Software mit gemeinsamer Code-Basis

Masterarbeit

im Studiengang Medieninformatik
zur Erlangung des akademischen Grades
Master of Science

Fachbereich Medien
Hochschule Düsseldorf

David Littig
Matrikel-Nr.: 649784
Datum: 07.12.2018

Erst- und Zweitprüfer

Prof. Dr.-Ing. Holger Schmidt
Prof. Dr.-Ing. M.Sc. Markus Dahm

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Quellen sind vollständig zitiert. Diese Arbeit wurde weder in gleicher noch ähnlicher Form einem anderen Prüfungsamt vorgelegt oder veröffentlicht. Ich erkläre mich ausdrücklich damit einverstanden, dass diese Arbeit mittels eines Dienstes zur Erkennung von Plagiaten überprüft wird.

Ort, Datum *David Littig*

Kontaktinformationen
David Littig
Schlebuscher Heide 37
51375 Leverkusen
—
david.littig@study.hs-duesseldorf.de

Zusammenfassung

Konzeption und Entwicklung eines erweiterbaren Passwortmanagers als Cross-Platform Open-Source-Software mit gemeinsamer Code-Basis

David Littig

Die im Laufe des Jahres veröffentlichten Passwörter auf der Plattform „Have I been pwnd“ zeigen deutlich, dass Nutzer aufgrund von besserer Merkbarkeit schwache Passwörter wählen. Dies hat zur Folge, dass die sensiblen Daten der Nutzer nur unzureichend geschützt sind. Aus diesem Grund gewinnt das Themengebiet der IT-Sicherheit zunehmend an Bedeutung. Passwortmanager stellen für dieses Szenario eine mögliche Lösung dar.

Der im Rahmen dieser Masterthesis entwickelte Passwortmanager „Safeword“ ermöglicht, sensitive Daten, die mittels adäquaten kryptografischen Verfahren geschützt werden, zentral an einer Stelle abzulegen und sukzessive unsichere Passwörter durch starke zufällige Passwörter zu ersetzen. Der Nutzer muss sich nur ein starkes Hauptkennwort merken, mit dem er sich bei dem Passwortmanager authentisiert und anschließend die gespeicherten Passwörter abrufen kann. Der Passwortmanager bietet zahlreiche Funktionen, die die Nutzung der Anwendung erleichtern, wie die simple und moderne Oberfläche, einen Passwort-Generator, der kryptografisch sichere Zufallspasswörter erstellt, oder Funktionen zur besseren Handhabung von Passwörtern, beispielsweise Label oder Notizen.

Safeword basiert auf einer modularen und erweiterbaren Architektur, die auf Code-Sharing setzt. Das Code-Sharing vereinfacht die Wartung der Anwendung und ermöglicht bei Schwachstellen eine schnellere Behebung. Die Ergebnisse aus dem Vergleich existierender Passwortmanager flossen direkt in die Entwicklung von Safeword ein, sodass aus Fehlern anderer Mitbewerber gelernt werden konnte. Um den Beitrag zur Open-Source-Welt sicherzustellen, wird Safeword unter der GPLv3-Lizenz bereitgestellt, sodass auch in Zukunft viele Nutzer von dem Passwortmanager profitieren können.

Voraussetzung für den Produktiveinsatz von Safeword ist die Entwicklung noch fehlender Funktionen und die Prüfung des Quelltexts nach aktuellen Standards der IT-Sicherheit.

Abstract

Konzeption und Entwicklung eines erweiterbaren Passwortmanagers als Cross-Platform Open-Source-Software mit gemeinsamer Code-Basis

David Littig

The publication of several passwords on the online plattform „Have I been pwnd“ are showing clearly that users prefer weak passwords in favor of better memorability. This results in the fact that sensitive user data insufficently secured. Therefore gains the topic attention increasingly. Passwords could be possible solution for this scenario.
The in this thesis developed password manager „Safeword“ enables to store sensitive data, which is protected by appropriate cryptographic method, centrally at one place and to replace unsecure passwords by strong random passwords progressively. The user just needs to memorize a strong master password, with which he authenticates at the password manager and access to the passwords is being granted afterwards. The password manager provides several functions that simplify the use, for example the simple and modern userinterface, a password generater that generates cryptographic secure random passwords, or functions that improve the handling of passwords like labels or notes.
Safeword is based on an modular and extensible architecture, which relies on code sharing. Code sharing simplifies the maintenance of the application and reduces the time to close a weakness effectively. The results of the comparison of existing password managers were included directly in the development of Safeword, so that lessons from mistakes of the competitors could be learned. To ensure the contribution to the open source world Safeword is licensed under GPLv3, so that users can profit of this password manager in the future.
Requirement for the production use of Safeword is the development of missing functions and the validation of the source code regarding the current metrics of IT security.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

API	Application Programming Interface
BSD	Berkeley Software Distribution
BSI	Bundesamt für Sicherheit in der Informationstechnik
DNS	Denial of Service
FTP	File Transfer Protocol
FTPS	FTP over SSL
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
OSI	Open Source Initiative
OSS	Open Source Software
REST	REpresentational State Transfer
SFTP	SSH File Transfer Protocol
TCATO	Two-Channel Auto-Type Obfuscation

Kapitel 1

Einleitung

1.1 Kontext und Motivation

Passwörter sind zentrales Element vieler *Authentifizierungssysteme*. Gemäß ISO/IEC 27000 („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 2; Swoboda, Pramateftakis, & Spitz, 2008, S. 15-16) ist die Authentifizierung der Prozess der Überprüfung von charakteristischen Merkmalen, ob diese mit vorgegebenen Merkmalen übereinstimmen. Authentifikationsmerkmale lassen sich dabei in drei Kategorien separieren: *wissensbasierte*, *besitzbasierte* und *eigenschaftsbasierte* Authentifikationsmerkmale. Das klassische Beispiel *Passwort* gehört zur Kategorie der wissensbasierten Authentifikationsmerkmale (Eckert, 2014, S. 468; Swoboda u. a., 2008, S. 150). Diese unterliegen strenger Geheimhaltung und sind somit nur dem eigentlichen Besitzer bekannt. Beispielsweise Web-Plattformen verwenden dieses Authentifikationsmerkmal, um sicherzustellen, dass die Person diejenige ist, für die sie sich ausgibt. Das besitzbasierte Authentifikationsmerkmal Smart-Card ist zum Beispiel bei der Auszahlung einer bestimmten Summe Geld von einem Bankautomaten relevant. Der Kunde besitzt eine Bank-Karte, die ihm eindeutig zugeordnet ist und somit authentifiziert. Zusätzlich kommt aber auch die PIN als Teil der sogenannten *Zwei-Faktor-Authentifizierung* hinzu. Denn mit der Karte lässt sich schwer überprüfen, ob derjenige, der gerade die Karte besitzt, wirklich der Eigentümer des Kontos ist. Ein Angreifer muss also neben der Inbesitznahme der Karte zusätzlich auch noch den PIN für den Zugriff in Erfahrung bringen. Das Merkmal *Fingerabdruck* der eigenschaftsbasierten Merkmale ist zurzeit im Smartphone-Bereich sehr verbreitet. Der Besitzer erhält erst vollen Zugriff auf die Funktionen des Smartphones, wenn dieser sich mit einem Fingerabdruck auf dem Sensor authentifiziert.

Die Thesis beschränkt sich auf das Passwort als Authentifikationsmerkmal, da vor allem die *Passwortsicherheit* bei vielen Passwörtern zu wünschen übrig lässt[1]. Die

[1] Troy Hunt: https://haveibeenpwned.com/About, abgerufen am 04.04.2018

Passwortsicherheit lässt sich anhand der Parameter *Entropie* (Newton u. a., 2013, S. 9) und *Zufälligkeit* bewerten. Die Entropie eines Passwortes beschreibt die Anzahl möglicher unterschiedlicher Passwörter. Diese ist zum einen abhängig von der Länge des Passwortes, aber auch von den gewählten Zeichen, die sich in verschiedene Symbolgruppen separieren lassen. Eine Symbolgruppe wäre beispielsweise „Großbuchstaben“ (26 Zeichen) oder „Ziffern“ (10 Zeichen). Je mehr Zeichen aus unterschiedlichsten Symbolgruppen genutzt werden, desto höher ist die Entropie. Die Länge des Passwortes erhöht drastisch die Anzahl möglicher Kombinationen. Der letztgenannte Parameter „Zufälligkeit“ ist ebenfalls sehr entscheidend. Denn würde der Nutzer aus jeder Symbolgruppe nur ein einziges Symbol auswählen und wiederholt nutzen (bspw. $gA2\_A\_2g$) sinkt die Passwortsicherheit, da ein Muster verwendet wurde. Muster werden häufig genutzt, um die Merkbarkeit und Wortlänge von Passwörtern zu erhöhen. Muster, wie etwa „Geburtsjahr am Ende“, sind sehr beliebt und verringern die Passwortsicherheit (Picolet & LLC, 2016, S. 39).

Bei der Passwortsicherheit spielt aber auch die kryptografische Nachbehandlung eines Passwortes eine elementare Rolle. Wird ein unsicheres Verfahren gewählt, so kann auch ein sehr starkes Passwort sehr schnell erfolgreich kompromittiert werden. Passwörter haben im Laufe ihrer Geschichte eine interessante Entwicklung erlebt. Zu Beginn wurden Passwörter noch im Klartext abgelegt. Zur Überprüfung wurde dann das eingegebene Passwort auf Übereinstimmung mit dem hinterlegten Passwort überprüft. Ein Angreifer hatte aber leichtes Spiel, da das Passwort nicht geheim war und einfach ausgelesen werden konnte. Daher kam die Idee auf, den Klartext in einen anderen Text zu überführen, der auf dem Passwort basiert. Wichtig ist dabei, dass die Überführung in den anderen Text für das selbe Wort immer den gleichen resultierenden Text ergibt und dieser Vorgang irreversibel ist, denn sonst könnte ein Angreifer das Verfahren einfach umkehren. Der resultierende Text wird *Hash* und der Vorgang des Umwandelns folglich *Hashing* genannt. Bekannte Hashing-Verfahren sind beispielsweise MD5 (RFC 1321[2]), SHA1 (RFC 3174[3]) oder SHA512 (FIPS 180-4[4]).

Im April 1992 wurde MD5 als Passwort-Hashing-Verfahren offiziell vorgestellt und hat sich seitdem weit verbreitet. Der Angreifer erzeugt durch Ausprobieren MD5-Hashes von möglichen Passwörtern und überprüft diesen auf Gleichheit zu dem entwendeten Passwort-Hash. Da diese Methode sehr zeitaufwändig ist und die Berechnungen für jeden Angriff erneut durchgeführt werden müssen, kann der Angreifer einen *Lookup-Angriff* ausführen, bei dem der Angreifer eine Tabelle erzeugt, die alle möglichen Passwort-Hashes mit dem MD5-Verfahren enthält. Der Angreifer muss vor

[2]IETF: https://www.ietf.org/rfc/rfc1321.txt, abgerufen am 12.04.2018
[3]IETF: https://www.ietf.org/rfc/rfc3174.txt, abgerufen am 12.04.2018
[4]FIPS: https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf, abgerufen am 12.04.2018

dem Angriff eine Leistung erbringen und die Hashes berechnen, weshalb diese Art von Angriffen auch *Precomputation attacks* genannt werden (Stajano, Mjølsnes, Jenkinson, & Thorsheim, 2016). Der Angreifer vergleicht dann den Passwort-Hash des Zielnutzers mit der Tabelle der im voraus berechneten Hashes. Eine Art dieser Angriffe sind die *dictionary attacks*. Bei einem Wörterbuchangriff erzeugt der Angreifer eine Tabelle mit Klartext-Hash-Paaren, die nach dem Hash-Wert sortiert sind, um den Zugriff auf die Paare zu vereinfachen (Adams, 2011, S. 332). Der Angreifer vergleicht daraufhin einen entwendeten Hash-Wert mit denen in der Tabelle, um so das Passwort des Nutzers in Erfahrung zu bringen. Die Speicherung ist aber sehr ressourcenintensiv und ineffizient. Sogenannte *Rainbow-Tables* sind jedoch eine effiziente Möglichkeit Hashes innerhalb von Ketten anzuordnen und in einer Tabelle zu speichern (Oechslin, 2003, S. 6). Klartexte werden von einer Hash-Funktion verarbeitet und daraufhin mittels einer Reduktionsfunktion auf einen anderen Klartext reduziert. Wiederholtes hashen und reduzieren führt dann zu einer Kette. Das Verfahren ist durch die alleinige Speicherung von Start- und Endpunkt der Ketten äußerst effizient. Diese Berechnung im Voraus benötigt nicht nur viel Zeit, sondern auch viel Speicherplatz. Der Knackpunkt ist da die beste Balance zu finden, da nicht alle existierenden Passwörter errechnet werden können, was im Allgemeinen als *Space-Time-Tradeoff* oder *Time-Memory-Tradeoff* bekannt ist (Oechslin, 2003, S. 2). Der verfügbare Speicherplatz ist im Vergleich zu damals aber stark gestiegen, sodass der Speicher seltener den limitierenden Faktor darstellt. Um sich gegen diese Art von Angriffen zu schützen, wurden *Salt* und *Pepper* eingeführt. Ein Salt ist eine zufällig generierte Zeichenkette mit einer bestimmten Länge, die dem Passwort vor dem Hashing-Prozess beigefügt wird (Turan, Barker, Burr, & Chen, 2010, S. 6; Eckert, 2014, S. 486). Dadurch wird das Passwort nicht nur länger sondern auch komplexer. Ein weiterer Vorteil ist auch, vorausgesetzt jeder Nutzer bekommt einen eigenen Salt für sein Passwort, dass zwei gleiche Passwörter zweier Benutzer, einen anderen Hash für jeden ergeben. Dadurch ist es für den Angreifer aufgrund des Zeitaufwandes nicht lukrativ eine Rainbow-Table zu erstellen (Burnett, 2006, S. 86), da der Salt zufällig ist und jede mögliche Kombination von Salt und Passwort errechnet werden müsste. Der Salt wird dabei auch in der Datenbank abgelegt. Der Pepper ist für die ganze Anwendung für alle Passwörter identisch und kann ebenfalls mit einem Passwort konkateniert werden, wird aber geheim gehalten und nicht in der Datenbank abgelegt[5]. Falls bei einem Angriff gespeicherte Daten entwendet werden können und der Angreifer dabei keinen Zugriff auf den Dienst erlangt hat, kann der Angreifer die Passwörter nicht oder nur unter sehr großem Aufwand errechnen, da ihm der Zugriff auf den möglichst stark gewählten Pepper fehlt, der zur Rekonstruktion des Passworts genutzt wird. Wird der Salt

[5]Guardian: https://www.theguardian.com/technology/2016/dec/15/passwords-hacking-hashing-salting-sha-2, abgerufen am 29.11.2018

ausreichend lang gewählt, wird der Pepper obsolet, da der Aufwand zur Berechnung der Passworthashes mit zugehörigem Salt ohnehin zu groß ist. Salt und Pepper können auch gemeinsam auf ein Passwort angewendet werden. Um den Rechenaufwand für alle Angriffe auf Hashes zu erhöhen, wurde die Hash-Länge, also die Länge des resultierenden Klartextes, angehoben. SHA512 erzeugt, wie der Name bereits andeutet, eine Zeichenkette mit 512 Bit Länge (Swoboda u. a., 2008, S. 106) im Vergleich zu 128 Bit Wortlänge bei MD5. Somit wächst auch die Größe der Rainbow-Tables drastisch an. Ein weit verbreitetes Problem, das auch keine der zuvor genannten technischen Maßnahmen lösen kann, ist, dass die selben Passwörter häufig für verschiedene Dienste genutzt werden. Dies stellt einen sehr elementaren Angriffsvektor dar, denn gestohlene Anmeldedaten können somit auf anderen Plattformen durch Angreifer ausprobiert werden.

Salt und Pepper erschweren zwar das Errechnen der Passwörter, berücksichtigen aber nicht die Weiterentwicklungen in der Rechengeschwindigkeit. Der Hashing-Algorithmus *bcrypt* führt einen Kostenfaktor ein, sodass die Berechnungszeit des Hashes reguliert werden kann (Wiemer & Zimmermann, 2014, S. 3). Einen Schritt weiter geht *scrypt* (IETF RFC 7914[6]), bei dem auch die Parallelisierbarkeit sowie der Speicherbedarf angepasst werden kann. Damit werden Precomputation-Attacks effektiv erschwert.

Die vorherigen Ausführungen zeigen ein Szenario der Nutzung eines Passwortmanagers auf. Eines der Ziele des Passwortmanagers ist es, Passwörter mit einer hohen Entropie und einem kryptografisch sicherem Zufall zu erzeugen, um somit eine Resistenz gegen zuvor dargestellten Angriffe zu erwirken. Die zuvor genannten Angriffe sind nämlich weitaus schneller durchzuführen, wenn kurze oder von Menschen generierte Passwörter verwendet werden. Diese Passwörter müssen daher vertraulich behandelt werden und auch abgelegt werden, sodass ein unbefugter Zugriff durch Dritte nicht möglich ist. Für ein höheres Maß an Komfort können die Passwörter über mehrere Geräte hinweg bereitgestellt werden, sodass der Nutzer sowohl unterwegs als auch zu Hause Passwörter abrufen kann. Der Nutzer generiert starke Passwörter für die benutzten Plattformen, wobei die Merkbarkeit der Passwörter irrelevant ist. Es kann aus einem großen Zeichenraum geschöpft werden und die Zeichen sind, im Gegensatz zu vom Nutzer generierten Passwort, vollständig zufällig. Der Nutzer merkt sich ein Passwort, mit dem der Manager die gespeicherten Passwörter entschlüsseln kann. Für diesen Zweck wird dann aber ein komplexes und langes Passwort verwendet. Dadurch, dass der Nutzer sich keine weiteren Passwörter merken muss, die komplex und lang sein sollen, kann er sich einfacher ein einziges langes und komplexes Passwort merken (Huth, Orlando, & Pesante, 2012, S. 2). Zwar kann der Nutzer sich mehrere komplexe

[6]IETF: https://tools.ietf.org/html/rfc7914, abgerufen am 01.12.2018

aber kurze Passwörter merken, diese weisen aber eine geringere Entropie auf, sodass diese unsicher sind. Die formulierten Ziele werden im Folgenden kurz und prägnant zusammengefasst:

1. Vergabe eines komplexen und langen Master-Passworts
2. Vertrauliche Behandlung und Speicherung der Daten
3. Erzeugung von Passwörtern mit hoher Entropie und sicherem Zufall
4. Hohe Verfügbarkeit

Als Nachteile stellt das Bundesamt für Sicherheit in der Informationstechnik (BSI) ein aufwendiges Verfahren zur Neuvergabe eines Passwortes sowie die unsichere Verwahrung notierter Kennwörter heraus. Daher empfiehlt das BSI jedem Nutzer dringend die Nutzung eines Passwortmanagers (Bundesamt für Sicherheit in der Informationstechnik, 2016, S. 3825). Auch die amerikanische Behörde CERT empfiehlt einen Passwortmanager und liefert hilfreiche Hinweise, die bei der Auswahl eines Managers zu beachten sind (Huth u. a., 2012).

Alle die zuvor genannten Ziele wie auch Probleme adressiert ein Passwortmanager. Ein Passwortmanager ist ein Verwaltungstool für Passwörter, welches wiederum mit einem Master-Passwort gesichert wird. An dieses Passwort gelten natürlich höchste Anforderungen, sodass die gespeicherten Passwörter optimal geschützt sind. Diese müssen eine gewisse Länge aufweisen und komplex sein, sodass das erste Ziel erfüllt wird. Dem zweiten Ziel entsprechend, werden die Passwörter sicher verschlüsselt hinterlegt und sind ohne Master-Passwort nicht abrufbar. Die Passwörter werden nicht wie bei Web-Plattformen mit einer kryptografischen Hash-Funktion verarbeitet, da auf diese ein späterer Zugriff möglich sein muss, der nur durch eine Ver- und anschließende Entschlüsselung gewährleistet wird. Der Manager nutzt zur Verschlüsselung der Daten adäquate Verschlüsselungsmechanismen, die dem aktuellen Standard entsprechen. Bei der Verarbeitung des Master-Passwortes wird auf eine adäquate Hash-Funktion zurückgegriffen, die Precomputation attacks effektiv erschwert. Diese Hash-Funktion verfügt über einen möglichst starken Salt und einen konfigurierbaren Kostenfaktor, der es ermöglicht auf die steigende Leistungsfähigkeit der Computer zu reagieren. Ziel 2 kann jedoch nur mit Hilfe von Ziel 1 erfüllt werden, denn nur ein starkes Passwort mit einer hohen Entropie verhindert effektiv Brute-Force-Angriffe. Ein Passwortmanager hat für den Nutzer zudem verschiedene Vorteile. Der Nutzer kann für jedes Portal ein sicheres zufälliges und langes Passwort generieren und dieses im Passwortmanager ablegen. Natürlich muss das Passwort auch in dem betreffenden Portal in das generierte Passwort geändert werden. Viele Passwortmanager sind auf

mehreren Plattformen verfügbar, sodass der Nutzer über verschiedene Geräte hinweg immer alle aktuellen Passwörter zur Verfügung hat. Die Nutzung bedeutet für den Nutzer aber anfänglich erhöhten Aufwand. Sämtliche Passwörter müssen eingetragen, gegebenenfalls auf der Plattform geändert werden. Dabei darf kein Passwort vergessen werden. Wird ein Passwort geändert, muss dies auch im Passwortmanager aktualisiert werden. Bekannte Passwortmanager bieten Browser-Erweiterungen an, die Passwortänderungen erkennen und vorschlagen, das Passwort ebenfalls im installierten Manager zu ändern. Wird das Master-Passwort vergessen, verliert der Nutzer alle seine gespeicherten Passwörter, was ein zusätzliches Risiko darstellt.

Obwohl viele Hersteller eine sichere Nutzung der Passwörter propagieren, sollte man diesen Aussagen nicht vollstes Vertrauen schenken. Vor allem bei proprietären Anwendung lässt sich diese Aussage nur schwer überprüfen. Deshalb untersuchten Silver, Jana, Boneh, Chen, und Jackson populäre Passwortmanager auf bekannte Verwundbarkeiten und stellten darunter auch Mängel fest. Dabei standen zum einen bei Web-Passwortmanagern das Einfüllen der Daten in den Browser im Vordergrund (Silver u.a., 2014, S. 453), zum anderen aber auch Verwundbarkeiten bei der Authentifizierung (Li, He, Akhawe, & Song, 2014). Ein weiteres Team tauchte tiefer in die Materie ein und untersuchte genau, wie die Daten gespeichert werden und führten Angriffe gegen die Datenhaltung aus (Gasti & Rasmussen, 2012). So konnten Schwachstellen bei dem Datenbankformat der alten KeePass-Version 1.X und den in Browsern integrierten Managern festgestellt werden. Zu dem Ergebnis, dass gespeicherte Passwörter in Browsern nur unzureichend geschützt werden, kamen auch Zhao und Yue. Passwörter werden mit veralteten Algorithmen verschlüsselt oder das Schlüsselmaterial lässt sich einfach auslesen (Zhao & Yue, 2014, S. 8-9). Heraus stach auch die Idee einer anderen Architektur, getauft „Kamouflage“, die das Entwenden von Passwörtern durch Köderpasswörter drastisch erschweren soll (Bojinov, Bursztein, Boyen, & Boneh, 2010). Dieses Architekturkonzept wird im Verlauf dieser Thesis näher betrachtet.

Auf dem Markt sind verschiedene Applikationen vorhanden, die zudem eine unterschiedliche Plattformunterstützung aufweisen. Einen groben Überblick über die vorhandenen Anwendungen liefert Tabelle 1.1. Eine detaillierter Vergleich wird in der Masterthesis erarbeitet.

Bei der Untersuchung der Quelltexte der quelloffenen Passwortmanager fällt auf, dass nur Padlock die gleiche Code-Basis für mobile Plattformen als auch Desktop-

[7] LastPass: https://www.lastpass.com/de, abgerufen am 07.04.2018
[8] KeePass: https://keepass.info, abgerufen am 07.04.2018
[9] 1Password: https://1password.com, abgerufen am 07.04.2018
[10] Padlock: https://padlock.io/, abgerufen am 07.04.2018
[11] Passbolt: https://www.passbolt.com/, abgerufen am 07.04.2018

Anwendung	Plattform	Lizenz	Betriebsmodus
LastPass[7]	iOS, Android, Windows Phone, Browser	Proprietär	Online mit Synchronisierung
KeePass[8]	Windows (MacOS, Linux, Android und iOS über inoffizielle Portierungen)	GPLv2	Offline
1Password[9]	MacOS, Windows, iOS, Android	Proprietär	Offline mit Synchronisierung
Padlock[10]	Windows, MacOS, Linux, ChromeOS, Android, iOS	GPLv3	Offline mit Synchronisierung
Passbolt[11]	Browser	AGPLv3	Online mit Synchronisierung

Tabelle 1.1: Überblick bekannter Passwortmanager

Betriebssysteme nutzt. Dabei wird für die mobile Plattform das Cordova-Framework[12] genutzt, welches, vereinfacht dargestellt, eine Web-View in einer App aufbaut und die App, die als Webseite geschrieben wurde, anzeigt. Native Apps bieten jedoch den Vorteil der viel besseren Performanz.

Code-Sharing ist ein wichtiger Teil der Thesis. Bei Code-Sharing handelt es sich um einen eleganten Ansatz, um Code-Duplikation zu vermeiden und die Wartung zu vereinfachen, indem Code von mehreren Plattformen genutzt wird, dessen Idee Hand in Hand mit der Cross-Platform-Entwicklung geht. Die Motivation der Cross-Platform-Entwicklung ist es, mit einer gelernten Programmiersprache eine Applikation zu entwickeln, die auf möglichst vielen Plattformen genutzt werden kann. Das Ziel ist es, möglichst wenig Wartungsaufwand zu erzeugen und somit die Effizienz der Entwicklung zu steigern. Ebenso wird die Qualitätssicherung vereinfacht, indem Tests nicht erneut für jede Plattform implementiert werden müssen. Ein sehr gutes Beispiel für Code-Sharing ist der Passwortmanager Padlock, der in dieser Thesis detailliert betrachtet wird. Es wurde ein Sicherheits-Audit der Software durch cure53[13] durchgeführt und einige Schwachstellen gefunden. Diese konnten zeitnah geschlossen werden, da der Code sowohl für mobile Geräte als auch Desktop-Geräte genutzt werden konnte. Das Audit zeigt auch auf, dass viele Fehler zugleich alle Plattformen betreffen. Die Behebung eines Problems fließt somit direkt in alle Plattformen ein. Mit Code-Sharing kann also verhindert werden, dass Fehler oder Schwachstellen nur auf einer Plattform existieren, wie es der Fall wäre, wenn die gleiche Funktion in verschiedenen Sprachen für unterschiedliche Plattformen implementiert würde, was die Wartung und Fehlersuche vereinfacht. Weitere Details zum Audit der Software werden in Kapitel 3.2.4 erörtert. Dieses Audit zeigt aber auch sehr gut die Nach-

[12]Apache: https://cordova.apache.org/, abgerufen am 29.11.2018
[13]Cure53: https://cure53.de/, abgerufen am 26.07.2018

teile der gemeinsamen Code-Teile, denn das Audit offenbart Schwachstellen in allen Plattformen und nicht nur einer Einzigen. Bis die Schwachstellen behoben werden, ist der Passwortmanager folglich über alle Anwendungen verwundbar. Nichtsdestotrotz können auch Fehler bei allen Plattformen einer Anwendung ohne Code-Sharing auftreten, wenn zu Beginn schlechte Designentscheidungen getroffen wurden, oder Sicherheitsgrundregeln ignoriert werden.
Bei einem Blick auf GitHub fällt auf, dass viele Entwickler bekannter Open-Source-Projekte auf vollständig getrennten Source-Code setzen und die Anwendungen separat warten. Erst durch die Cross-Platform-Entwicklung ist es möglich einen hohen Anteil an Code-Sharing zu erreichen. Bei der separaten Entwicklung der Anwendungen zeigt sich, dass die Zeiten zum Beheben von Problemen auf den jeweiligen Plattformen weit auseinander gehen (Zhou, Neamtiu, & Gupta, 2015, S. 4). Am schnellsten wurden die Probleme auf den mobilen Plattformen behoben, wohingegen die Bearbeitung unter Desktop-Plattformen, möglicherweise auch durch die Anpassung für verschiedene Desktop-Betriebssysteme, mehr Zeit in Anspruch nimmt. Diese Zeit kann durch Code-Sharing minimiert werden, da sich auch die Komplexität der Anwendung enorm reduziert. Zwar ist der Strukturierungsaufwand der Anwendung zu Beginn höher, jedoch zahlt sich Code-Sharing auf lange Sicht des Projektes aus. Es entfällt zudem das Einarbeiten in andere Sprachen und Konzepte.

Entsprechend dem Kerckhoffschen Prinzip birgt Open Source Software (OSS) wichtige Vorteile. Kerckhoff war ein niederländischer Kryptograf des 19. Jahrhunderts, der die These aufstellte, dass ein kryptografisches System auch sicher sein solle, selbst wenn das ganze System, außer dem Schlüssel, bekannt sei (Kerckhoff, 1883). Dem gegenüber steht das Prinzip „Security through Obscurity“, frei übersetzt: Sicherheit durch Unklarheit. Die Entwicklung einer Applikation als OSS erhöht Code-Qualität und Sicherheit der Anwendung, da andere Personen den Quellcode untersuchen und Schwachstellen aufdecken sowie direkt reparieren können. Gerade bei kryptografischen Anwendungen fällt es schwer den Herstellern proprietärer Software zu vertrauen. Bei OSS kann der Nutzer den Quellcode vorher einsehen und überprüfen. Der Nutzer hat sogar die Möglichkeit die Anwendung selbst zu kompilieren. Dadurch muss sich der Anwender nicht auf bereits kompilierte Anwendungen, die durch Dritte zu Verfügung gestellt werden, verlassen und weiß, dass die Anwendung nicht manipuliert wurde.

1.2 Wissenschaftlicher Beitrag

Die Wartung der separaten Anwendungen für die verschiedenen Plattform ist ein sehr zeitaufwändiger als auch fehlerbehafteter Prozess. Anwendungen sind dann meist

nicht im Einklang miteinander, was Design und Funktionen betrifft. Beide Anwendungen müssen separat von einander getestet werden und setzen somit Sprachkenntnisse für die native Programmierung für die jeweilige Plattform voraus. Verschiedene Entwicklertools versuchen eine Brücke zwischen verschiedenen Plattformen zu errichten. Dabei gibt es verschiedene Ansätze, wie beispielsweise die erwähnte WebView, jedoch ist auch eine Transpilierung von JavaScript in einen nativen Code für Android und iOS möglich.

Wie eingangs im Abschnitt „Motivation“ bereits erwähnt, ist die Zahl der Passwortmanager, die auf eine gemeinsame Code-Basis setzen, sehr gering. Interessant ist es daher zu untersuchen, wie eine Cross-Platform-Applikation mit möglichst viel Code-Sharing strukturiert und implementiert wird. Diese Arbeit soll die Möglichkeiten, aber auch die Grenzen der Cross-Platform-Entwicklung, durch die Entwicklung eines eigenen Passwortmanagers, der die zuvor genannten Ziele eines Passwortmanagers erfüllt, aufzeigen. Als Open-Source-Software können somit auch andere Entwickler von den gesammelten Erfahrungen sowie dem Quellcode profitieren und weitere Verbesserungen durch andere Entwickler eingepflegt werden (Perens, 1999, S. 2). Beide Aspekte, sowohl Entwicklung als OSS als auch die gemeinsame Code-Basis, tragen zur Verbesserung der Sicherheit bei. Dies äußert sich darin, dass, wenn ein Fehler gefunden wird, dieser sofort korrigiert werden kann und direkt in alle Plattformen einfließt. Ein Patch für ein Sicherheitsproblem muss also nicht für jede Plattform individuell geschrieben werden. Bis jetzt ist nur ein Angriff[14] auf den Online-Passwortmanager LastPass aus Juni 2015 bekannt geworden. Erfolgreiche Angriffe auf lokale Passwortmanager, also welche, die die Passwörter lokal speichern, sind nicht bekannt.

Die gewählte Open-Source-Lizenz nimmt die Entwickler je nach Typ mehr oder minder in die Verantwortung, den modifizierten Code veröffentlichen zu müssen, wie beispielsweise bei der GPLv3, bei der eine Veröffentlichung verpflichtend ist (Kojima, 2016, S. 10; Institut für Rechtsfragen der Freien und Open Source Software (München), 2005, S. 1). Andere bekannte Lizenzen sind beispielsweise Apache 2.0, BSD oder die Mozilla Public License (Fitzgerald, 2006).

1.3 Zielsetzung

Zentrales Ziel der Masterthesis ist es, einen Cross-Platform-Passwortmanager zu entwickeln, der auf bekannten Desktop-Plattformen sowie dem mobilen Betriebssystem Android ausgeführt werden kann. Dazu wird zuerst eine Architektur konzipiert. Das

[14] Zeit: http://www.zeit.de/digital/datenschutz/2015-06/lastpass-passwortmanager-hack-sicherheit, abgerufen am 07.04.2018

Konzept beinhaltet eine Code-Struktur sowie einen modularen Aufbau und das Datenbankschema. Es müssen folgende Funktionalitäten implementiert werden, die innerhalb der Thesis berücksichtigt werden:

Ziele hinsichtlich IT-Sicherheit:

- Sicherung der Datenbank hinsichtlich Vertraulichkeit und Integrität durch adäquate kryptografische Verfahren
- Sicherung des kryptografischen Schlüsselmaterials hinsichtlich Vertraulichkeit und Integrität durch adäquate kryptografische Verfahren und einem Master-Passwort
- Schließen der Datenbankverbindung bei Inaktivität
- Guter Zufall für Passwort-Generator

Ziele hinsichtlich Funktionalität:

- Synchronisation über Cloud-Dienst
- Passwort-Generator für neue Passwörter
- Ohne bestehende Internet-Verbindung funktionsfähig
- Angabe eines Profils (Voller Name, Geburtsdatum) sodass Passwörter mit Hilfe dieser Daten intelligenter bewertet werden können
- Aufzeichnen von Authentifizierungsversuchen

Ziele hinsichtlich Benutzbarkeit:

- Aufbau einer simplen und modernen Oberfläche
- Anzeige gespeicherter Passwörter mit Label, Notizen, Ablaufdatum
- Standardmäßig sind Passwörter unkenntlich gemacht und können auf Knopfdruck angezeigt werden
- Passwort-Generator auch nutzbar ohne gerade ein neues Passwort in der DB zu hinterlegen
- „Credit“-Seite, die über Rechtliches bezüglich der Lizenz informiert

Ziele hinsichtlich Qualität:

- Testen der Kernfunktionalitäten

Weitere Ziele:

- Identifizierung Vor- und Nachteile Authentifizierungsarten
- Vergleich bekannter Passwortmanager
- Aufbau einer modularen Architektur

Im Rahmen der Thesis soll zudem überprüft werden, inwieweit eine Erweiterbarkeit über alle Plattformen hinweg implementiert werden kann und welchen Einschränkungen dies unterliegt. Denkbar wäre in Zukunft zum Beispiel eine Zwei-Faktor-Authentifizierung über den Fingerabdruck. Abschließend soll ein kurzer Vergleich der bekanntesten Lizenzen angestrebt und eine passende Lizenz ausgewählt werden.

1.4 Struktur der Arbeit

Die Arbeit befasst sich eingangs mit den Grundlagen zur Authentifizierung in Kapitel 2. Darunter fallen zuerst die Definitionen aller wichtigen Begriffe, gefolgt von der Analyse von Vor- und Nachteilen verschiedener Authentifizierungsarten und der genaueren Betrachtung der Mehr-Faktor-Authentifizierung. In Kapitel 3 wird eine adäquate Lizenz für den zu entwickelnden Passwortmanager ausgewählt. Anschließend werden bekannte Passwortmanager miteinander verglichen, um Erkenntnisse aus diesem Vergleich in die Implementierung einfließen zu lassen. Kapitel 4 thematisiert folglich die Konzeption und Umsetzung der definierten Ziele in dem zu entwickelnden Passwortmanager. Es wird die Architektur beschrieben, sowie auf einige wichtige Implementierungsdetails eingegangen, als auch auf die damit einhergehenden Probleme. Das Kapitel wird mit Hinweisen zum Betrieb der Anwendung und einer bebilderten Kurzanleitung abgeschlossen. Die Evaluierung der Anwendung mit Fokus auf das betriebene Code-Sharing findet in Kapitel 5 statt. Zusätzlich werden noch mögliche, aber nicht dringliche, Erweiterungen vorgestellt und die Architekturentscheidung im Vergleich zur „Kamouflage“-Architektur betrachtet. Abschließend wird ein Fazit gezogen und ein Ausblick über wichtige, dringend zu implementierende, Ergänzungen gegeben.

Kapitel 2

Grundlagen zu Authentifizierungssystemen

In diesem Kapitel werden die theoretischen Grundlagen zur Authentifizierung erläutert. Dazu müssen in Kapitel 2.1 die elementaren Begriffe definiert und abgegrenzt werden, darunter auch Begriffe aus der Informations-Sicherheit, die die Grundlage der ISO/IEC-27000-Reihe darstellen. Vertiefend wird in Kapitel 2.3 die Mehr-Faktor-Authentifizierung, auch mit Hinblick auf die verschiedenen Ausprägung dieser, betrachtet.

2.1 Begriffsdefinitionen

2.1.1 Authentifikation

Gemäß ISO/IEC 27000 ist die Authentifizierung der Prozess der Überprüfung von charakteristischen Merkmalen, ob diese mit vorgegebenen Merkmalen übereinstimmen („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 2; Swoboda u. a., 2008, S. 15-16).

2.1.2 Schwachstelle

Eine Schwachstelle eines Systems ist eine sicherheitsrelevante Schwäche, die durch eine oder mehrere Bedrohungen ausgenutzt und das System somit verwundet werden kann („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 11). Schwachstellen können beispielsweise durch Fehler in der Konzeption, bei den Algorithmen oder bei dem Betrieb des Systems auftreten[1] (Stoneburner, Goguen, & Feringa, 2012, B-13).

[1] BSI Glossar: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, abgerufen am 18.11.2018

2.1.3 Verwundbarkeit

Eine Verwundbarkeit ist eine ausgenutzte Schwachstelle, bei der Sicherungsmaßnahmen umgangen werden, um Systeme unberechtigt zu kompromittieren, auszuspähen oder zu manipulieren[2] (Stoneburner u. a., 2012, B-13).

2.1.4 Bedrohung

Eine Bedrohung ist ein mögliches eintretendes Ereignis, das durch die Ausnutzung von Schwachstellen Schäden am System verursachen kann („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 11; Stoneburner u. a., 2012, B-13). Diese haben das Ziel die drei schützenswerten Ziele der Informationssicherheit „Verfügbarkeit“, „Integrität“, „Vertraulichkeit“, die auch als das CIA-Dreieck bekannt sind, zu verletzen[3].

2.1.5 Angriff

Ein Angriff bezeichnet den Vorgang des Ausnutzens oder den Versuch des Ausnutzens einer Schwachstelle mit dem Ziel Informationen zu sammeln, zu manipulieren oder zu löschen („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 1; Stoneburner u. a., 2012, B-1), um dies zum eigenen Vorteil zu nutzen[4].

2.1.6 CIA-Dreieck

Das CIA-Dreieck nach Peltier, dargestellt in Abbildung 2.1, basiert auf den drei Schutzzielen Vertraulichkeit (**C**onfidentiality), Integrität (**I**ntegrity) und Verfügbarkeit (**A**vailability). Laut Peltier, 2013, Seite 317 sind Informationen gut geschützt, wenn alle drei Schutzziele nicht verletzt werden.

Nachfolgend werden die Begriffe der drei Schutzziele definiert.

Vertraulichkeit Das Schutzziel Vertraulichkeit hat zum Ziel, dass Informationen nur für autorisierte Instanzen in einer zulässigen Weise verfügbar sind („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 2; Stoneburner u. a., 2012, B-3).

[2] BSI Glossar: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, abgerufen am 18.11.2018

[3] BSI Glossar: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, abgerufen am 18.11.2018

[4] BSI Glossar: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, abgerufen am 18.11.2018

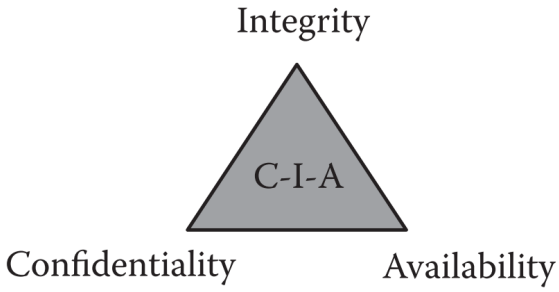


Abbildung 2.1: CIA-Dreieck nach Peltier (Peltier, 2013, S. 317)

Integrität Integrität beschreibt die Korrektheit und einwandfreie Funktionsweise von Daten bzw. Systemen[5] („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 5). Integrität bezeichnet zudem den Schutz vor ungewollter Manipulation oder Löschung von Informationen (Stoneburner u. a., 2012, B-3).

Verfügbarkeit Verfügbarkeit beschreibt die Sicherstellung eines zeitnahen und zuverlässigen Zugriffs auf Informationen (Stoneburner u. a., 2012, B-2). Diese Informationen dürfen nur für berechtigte Instanzen nutz- und abrufbar sein („ISO/IEC 27000: Information technology – Security techniques“, 2018, S. 2).

2.1.7 Passwort

Ein Passwort ist eine für Dritte unbekannte Zeichenfolge. Die Zeichenfolge kann aus Buchstaben, Ziffern oder Sonderzeichen bestehen. Passwörter werden durch Regeln, sogenannte Policies, beschränkt. Diese Regeln setzen eine gewisse Passwortlänge sowie das Vorkommen und die Menge von Groß-, Kleinbuchstaben, Ziffern und Sonderzeichen voraus, haben aber auch sehr starken Einfluss auf die Usability und die Merkbarkeit eines Passwortes (Stobert & Biddle, 2014). Der NIST-Standard SP 800 63B (Grassi u. a., 2017, S. 13) definiert Anforderungen und Empfehlungen für Entwickler von Passwortsystemen im Umgang mit Passwörtern, als auch Empfehlungen für Nutzer solcher Systeme bezüglich der Wahl eines starken Passworts nach heutigem Stand der Technik. Passwörter sollen laut NIST zudem immer mit Datenbanken aus bereits veröffentlichten Daten-Leaks verglichen werden, denn viele Nutzer tendieren dazu, ihre vorher ausgedachten Passwörter nur um die geforderte Zeichen zu erweitern oder vorhersehbare Ersetzungen der Zeichen im Passwort durchzuführen. Ein prominentes Beispiel der jüngeren Vergangenheit stellt die Plattform „Have I been pwned“[6] dar. Die Plattform sammelt Passwörter von veröffentlichten Datenbeständen und stellt diese über eine Schnittstelle auch anderen Plattformen zur Verfügung. Die

[5] BSI Glossar: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, abgerufen am 18.11.2018
[6] Have I been pwned: https://haveibeenpwned.com/About, abgerufen am 15.11.2018

Passwörter sind in einer geschützten Form abrufbar, sodass nur ein Rückschluss auf die Existenz des Passwortes in der Datenbank möglich ist. Der Missbrauch der Daten wird somit erschwert, da die Passwörter nicht im Klartext vorliegen.

2.1.8 Entropie

Die Entropie ist das Maß der Ungewissheit eines Passworts und gibt somit Aufschluss über die Stärke eines Passwortes. Je höher die in Bits gemessene Entropie, desto schwieriger ist das Passwort durch einen Angreifer rechnerisch zu rekonstruieren (Newton u. a., 2013, S. 9; Ma, Campbell, Tran, & Kleeman, 2010, S. 586). Beeinflusst wird die Entropie durch die Anzahl der Zeichen und die verwendeten Symbolgruppen, wie Klein- und Großbuchstaben, Zahlen und Sonderzeichen (Taha, Ahmed, E. Moktar, H. Salim, & M. Abdullah, 2013, S. 498). Alle Entscheidungen, die ein Mensch bei der „Generierung“ eines Passwortes trifft, verringern die Entropie, da das Passwort einem System entspricht, das nur eine verringerte Anzahl an möglichen Passwörtern bietet. Eine möglichst hohe Entropie kann daher nur durch einen echten Zufall erreicht werden (Newton u. a., 2013, S. 109).
Für eine kryptografische Anwendung ist es wichtig, dass zwingend kryptografisch sichere pseudozufällige Zeichenketten generiert werden (Müller, Krummeck, & Romsy, 2016, S. 12). Dass der Generator kryptografisch sicher ist, bedeutet, dass die generierte pseudozufällige Zeichenkette nicht von einer echtzufälligen Zeichenkette unterschieden werden kann und der Zustand des Generators für einen Außenstehenden nicht ersichtlich ist (Kelsey, Schneier, Wagner, & Hall, 1998, S. 4). Wären die generierten Zufallszeichen nicht kryptografisch sicher, könnte sich das Ergebnis des Zufallsgenerators zum Beispiel durch einen Angreifer berechnen lassen, was die Sicherheit vieler Verschlüsselungsverfahren bedrohen würde. Zufällig generierte Passwörter wären somit keineswegs mehr zufällig und verlieren ihren Vorteil der Unberechenbarkeit. Ebenfalls dem Kerckhoffschen Prinzip folgend, sollten die zuvor genannten Eigenschaften bestehen, obwohl die Funktionsweise des Zufallsgenerators bekannt ist.

2.1.9 Passwortmanager

Ein Passwortmanager ist eine Softwareanwendung, die Authentifikationsgeheimnisse, wie Passwörter oder PIN-Nummern und eventuell zugehörigen Nutzernamen, mit sicheren und aktuellen kryptografischen Verfahren sichert und abspeichert (Huth u. a., 2012). Dies ist heutzutage erforderlich, da der Nutzer bei vielen Plattformen im Internet registriert ist, die alle ein sicheres und einzigartiges Passwort erfordern. Ein sicheres Passwort zeichnet sich, wie zuvor genannt, durch eine hohe Entropie aus. Ein solches sicheres Passwort bietet aber einen schlechten Komfort, da die Merkbarkeit

mehrerer beispielsweise 32 Zeichen langer und untereinander unterschiedlicher Zeichenketten nicht gewährleistet werden kann.
Die Entschlüsselung der gespeicherten Daten ist nur der Person gestattet, die sich mit einem vorher gesetzten zwingend starken Master-Passwort authentisieren kann (Bundesamt für Sicherheit in der Informationstechnik, 2016, S. 3825). Ein Passwortmanager reduziert den Merkaufwand des Nutzers, indem nur ein sicheres und langes Master-Passwort gemerkt werden muss, das einen Teil des Schlüsselmaterials darstellt. Die gespeicherten Passwörter können nun eine höhere Entropie aufweisen, da diese zufällig durch Software generiert werden und auch eine erhöhte Länge aufweisen können, da die Merkbarkeit keine übergeordnete Rolle mehr spielt.

2.1.10 Open Source

Open Source bedeutet wortwörtlich aus dem Englischen übersetzt „freie Quellen“. Strikt der Übersetzung nach, ist eine Open-Source-Anwendung also eine Software, von der der Quelltext für die Öffentlichkeit vorliegt. Viel mehr steckt hinter dem Begriff Open Source eine philosophische Frage, der sich auch die GNU Community sowie die Open Source Initiative (OSI) angenommen haben. Die Begriffsdefinition der OSI[7] deckt sich mit der Definition der „freien Software“ der GNU Community (Perens, 1999, S. 2). Denn nur das Veröffentlichen des Quelltextes erlaubt nicht die Modifikation oder Weiterverteilung des Quelltextes. Es müssen verschiedene Freiheiten, wie die zuvor genannten, oder die Nutzung jedweder Art gewährt werden, damit aus der Anwendung eine freie Software oder vollständige Open-Source-Software wird. Diese Freiheiten werden durch eine Lizenz gesichert, die darüber hinaus keine Personen, Gruppen oder Technologien diskriminieren bzw. ausschließen darf. Ein prominentes Beispiel für eine Lizenz, die diese Kriterien erfüllt, ist die GNU GPLv3[8].
Das Kerckhoffsche Prinzip setzt voraus, dass Algorithmen und damit der Quellcode kryptografischer Anwendungen offengelegt werden müssen. Denn so können die zugrunde liegenden Algorithmen überprüft, korrigiert und weiterentwickelt werden. Zudem hat der Nutzer so die Möglichkeit von der korrekten Implementierung der relevanten Algorithmen zu überzeugen.

[7] OSI: https://opensource.org/osd, abgerufen am 01.12.2018
[8] GNU: https://www.gnu.org/licenses/gpl.html, abgerufen am 15.11.2018

2.2 Vor- und Nachteile verschiedener Authentifizierungsarten

Passwörter sind zentrales Element vieler Authentifizierungssysteme. Authentifikationsmerkmale lassen sich dabei in drei Kategorien separieren: wissensbasierte, besitzbasierte und eigenschaftsbasierte Authentifikationsmerkmale (Tietz, Pelchen, Meinel, & Schnjakin, 2017, S. 21-24).

Das klassische Beispiel *Passwort* gehört zur Kategorie der wissensbasierten Authentifikationsmerkmale (Eckert, 2014, S. 468; Swoboda u. a., 2008, S. 150). Passwörter unterliegen strenger Geheimhaltung und sind somit nur dem eigentlichen Besitzer bekannt. Beispielsweise Web-Plattformen verwenden dieses Authentifikationsmerkmal, um sicherzustellen, dass die Person diejenige ist, für die sie sich ausgibt.
Da das Passwort das Kernelement dieser Thesis ist, wird im Folgenden der Prozess der passwortbasierten Authentisierung betrachtet. Zu Beginn vergibt der Nutzer ein Passwort zum Beispiel für eine Web-Plattform oder ein Programm auf dem Computer. Dieses Passwort wird abgespeichert, jedoch nicht im Klartext, sondern nachdem es von einer Hash-Funktion verarbeitet wurde. Eine Hash-Funktion ist eine Einwegfunktion die eine Zeichenkette auf eine Andere abbildet. Dabei muss diese Abbildung eindeutig und kollisionsfrei sein. Das bedeutet, dass wenn ein Passworthash berechnet wird, erzeugt die Funktion immer den gleichen Hash, der auch nicht durch ein anderes Wort als Eingabe erzeugt werden darf. In Kapitel 1.1 wurde bereits die Entwicklung der Verfahren zur Passwortsicherheit dargestellt und wird nun mit Hinblick auf die möglichen Angriffsarten zusammengefasst. *Precomputation-Angriffe*, zu denen beispielsweise der Wörterbuchangriff oder auch Lookup-Angriffe mit Rainbow-Tables gehören, werden effektiv durch die Nutzung von Salt und Pepper erschwert, denn ein Angreifer kann die Berechnung aller möglichen Passwörter erst dann durchführen, wenn Daten bereits entwendet wurden und nicht etwa vorher, was den Angriff durch den hohen Zeitaufwand unpraktikabel macht. Alle Angriffe auf Hashing-Verfahren werden durch ein langes sowie komplexes Passwort und durch Verfahren mit großer Hash-Länge erschwert. Dies wirkt sich vor allem positiv auf die Resistenz gegenüber „Brute-Force-Angriffen“ aus. Um zudem der stetig steigenden Rechenleistung der Computer entgegenzuwirken, werden Hashing-Verfahren mit einem einstellbaren Kostenfaktor genutzt.
Ein wichtiger Vorteil der wissensbasierten Merkmale ist es deshalb, dass diese einfach widerrufen werden können. Der Nutzer kann eine neue PIN-Nummer oder Passwort wählen und somit den Zugang mit den vorherigen Daten invalidieren, sodass der Zugriff nur noch mit den neuen Daten möglich ist (Eckert, 2014, S. 471). Das be-

deutet folglich, dass zeitlich intensive Angriffe wirkungslos sind, wenn der Nutzer sein Passwort ändert, bevor der Angriff abgeschlossen ist, da der Angreifer den zeitintensiven Angriff erneut beginnen müsste. Den Vorteilen stehen aber auch einige Nachteile gegenüber. Denn starke Passwörter mit echter Zufälligkeit und einer großen Länge weisen eine sehr schlechte Merkbarkeit auf. Ein weiteres Problem ist, dass das Merkmal nicht vor Verlusten geschützt ist (O'Gorman, 2003, S. 2024). Ein Angreifer könnte an das Passwort gelangen und dann dieses zur Authentifikation nutzen. Die dargestellte, nicht vorhandene Pflicht der Anwesenheit der Person, die sich authentifiziert, ist ein weiterer Nachteil. Zwar sehen Nutzer das Teilen das Passworts mit Verwandten oder gar Freunden als sehr komfortabel an um gemeinsam eine Plattform zu nutzen, jedoch ist dies in der IT-Sicherheit ein sehr kritischer Faktor (O'Gorman, 2003, S. 2024).

Das besitzbasierte Authentifikationsmerkmal *Smart-Card* ist zum Beispiel bei der Auszahlung einer bestimmten Summe Geld von einem Bankautomaten relevant. Eine Smart-Card ist eine Kunststoffkarte, die einen Kontakt und einen integrierten Mikro-Chip aufweist. Auf diesem Chip können Informationen hinterlegt werden, wie beispielsweise Authentifikationsdaten (Trojahn, 2016, S. 13).
Hier liegt einer der wichtigen Vorteile gegenüber wissensbasierten Authentifikationsmerkmalen. Der Nutzer muss sich keine Zugangsdaten merken, weshalb Smart-Cards auch häufig zur Authentifizierung der Nutzer an Laptops in der Firmenumgebung genutzt werden. Verliert ein Nutzer seine Zugangskarte, kann diese deaktiviert werden, sodass diese vom Authentifizierungssystem nicht mehr akzeptiert wird. Ein Missbrauch ist so dann nicht mehr möglich (Schilling, 2018, S. 43). Bleibt der Diebstahl aber erst einmal unbemerkt und die Karte weiterhin aktiv, kann der Angreifer sich als der ursprünglicher Eigentümer authentifizieren. Die persönliche Anwesenheit kann hier ebenfalls nicht sichergestellt werden.
Bezugnehmend auf das vorangegangene Beispiel der Smart-Card bei einem Geldautomaten, besitzt der Kunde eine Bank-Karte, die ihm eindeutig zugeordnet ist und somit authentifiziert. Zusätzlich kommt aber auch die PIN-Nummer als Teil der sogenannten *Zwei-Faktor-Authentifizierung* hinzu (Schilling, 2018, S. 44). Denn mit der Karte lässt sich schwer überprüfen, ob derjenige, der gerade die Karte besitzt, wirklich der Eigentümer des Kontos ist. Ein Angreifer muss also neben der Inbesitznahme der Karte zusätzlich auch noch den PIN für den Zugriff in Erfahrung bringen.

Das Merkmal *Fingerabdruck* der eigenschaftsbasierten oder biometrischen Merkmale ist zurzeit im Smartphone- und Notebook-Bereich sehr verbreitet. Bei dem Fingerabdruck werden spezielle Eigenschaften wie Papillarlinien, Porenstruktur und

Punkte betrachtet (Eckert, 2014, S. 496). Aufgrund dieser individuellen und einzigartigen Ausprägung des Fingerabdrucks kann ein Hash errechnet werden, der dem Nutzer eindeutig zugewiesen werden kann. Der Besitzer erhält erst vollen Zugriff auf die Funktionen des Smartphones, wenn dieser sich mit einem Fingerabdruck auf dem Sensor authentifiziert.
Da der Fingerabdruck eine für jede Person individuelle Ausprägung hat, bietet diese Eigenschaft verschiedene Vorteile hinsichtlich der IT-Sicherheit. Im Vergleich zu den wissens- und besitzbasierten Authentifikationsmerkmalen, sind die biometrischen Merkmale besser vor Verlust geschützt und benötigen die persönliche Anwesenheit des zu Authentifizierenden. Der Nutzer muss sich keine Zugangsdaten merken, und erhält somit den Komfort, der von besitzbasierten Authentifikationssystemen bekannt ist. Zu beachten ist, dass biometrische Merkmale nicht widerrufbar sind, da der Nutzer die Merkmale immer besitzt (Trojahn, 2016, S. 13).
Die Ergebnisse werden in Tabelle 2.1 in Anlehnung an die Arbeit von O'Gorman kompakt in einer Übersichtstabelle dargestellt (O'Gorman, 2003, S. 2024).

Eigenschaft	**Eigenschaftsbasierte Merkmale**	**Wissensbasierte Merkmale**	**Besitzbasierte Merkmale**
Widerruf		•	•
Merkbarkeit	•		•
Anwesenheit	•		
Gemeinsame Benutzung	•		
Verlust	•		

Tabelle 2.1: Vor- und Nachteile verschiedener Authentifikationsmerkmale (O'Gorman, 2003, S. 2024)

2.3 Mehr-Faktor-Authentifizierung

Vor allem bei wissens- und besitzbasierten Merkmalen kann durch die mögliche Nichtanwesenheit des zu Authentisierenden nicht sicher nachgewiesen werden, dass die Person die ist, für die sie sich ausgibt.
Bei einer klassischen Authentisierung bei einer Web-Plattform werden der Nutzername und ein Passwort genutzt, auch bekannt als Ein-Faktor-Authentifizierung, da die erfolgreiche Authentisierung nur von einer einzigen Komponente abhängig ist (Aloul, Zahidi, & El-Hajj, 2009). Die Zwei-Faktor-Authentifizierung (2FA) ist eine Ausprägungen der Mehr-Faktor-Authentifizierung[9]. Bei der Mehr-Faktor-Authentifizierung

[9]Security Insider: https://www.security-insider.de/was-ist-eine-zwei-faktor-authentifizierung-2fa-a-631495/, abgerufen am 01.12.2018

werden mehrere voneinander unabhängige Authentifikationsmerkmale genutzt, wie beispielsweise Smart-Cards, USB-Sticks oder PIN-Codes. Ein beliebtes Beispiel ist hier, wie bereits erwähnt, die Smart-Card des Bankunternehmens in Kombination mit dem wissensbasierten Merkmal PIN[10].

Häufig sind für Authentifikationssysteme sogenannte TAN-Listen verfügbar, die als Einwegkennwörter fungieren[11]. Die ausgedruckte TAN-Liste stellt somit ein besitzbasiertes Merkmal dar, da der Nutzer sich die TANs nicht merken muss. Ähnlich operieren heutzutage viele Zwei-Faktor-Authentifizierungssysteme beliebter Web-Plattformen. Zuvor wird eine Telefonnummer für einen Anruf oder eine SMS hinterlegt[12]. Wenn der Nutzer sich bei einer Plattform anmelden möchte, kontaktiert die Web-Plattform die Telefonnummer über die gewählte Art und Weise, um den Code zu übermitteln. Dieser Code ist einmalig und weist eine begrenzte Lebensdauer auf. Der Nutzer gibt die Nutzerdaten und zusätzlich den Code ein, um Zugriff auf die Plattform zu erhalten (Aloul u. a., 2009).

Der Vorteil ist offensichtlich, denn ein Angreifer muss nicht nur in den Besitz eines Passwortes kommen, sondern auch noch in den Besitz des zweiten Faktors, beispielsweise ein Smartphone, ein USB-Stick oder eine Smart-Card. Möglich ist auch ein zweiter Faktor in Form eines NFC-Tags[13] oder eines biometrischen Merkmals, wie dem Fingerabdruck. Sowohl NFC- als auch Fingerabdrucksensoren sind in modernen Smartphones verbaut. die sich somit für die Authentifikation nutzen lassen können. Die Komplexität des Anmeldeprozesses wird dadurch etwas erhöht, was aber mit dem Gewinn an Sicherheit vereinbar ist. Viele Personen haben ihr Smartphone immer dabei oder wollen sich direkt auf dem Smartphone anmelden, sodass dieser Umstand heutzutage keine Hürde mehr ist.

Die Mehr-Faktor-Authentifizierung muss natürlich auch immer im jeweiligen Kontext betrachtet werden. Mit Hinblick auf die in dieser Thesis erarbeiteten Anwendung, die lokal installiert wird und bei der das Versenden von SMS kostenpflichtig ist, bieten sich eher der USB-Stick, NFC-Tag oder der Fingerabdruck als zweiter Faktor an. Der Schlüssel kann dann auf den Speichermedien und NFC-Tags hinterlegt werden, der Fingerabdruck bietet sich vor allem auf mobilen Geräten wie Smartphones durch die einheitliche Application Programming Interface (API) an.

[10] SearchSecurity: https://www.searchsecurity.de/definition/Zwei-Faktor-Authentifizierung, abgerufen am 01.12.2018

[11] PC Welt: https://www.pcwelt.de/ratgeber/Wichtige_Dienste_per_Zwei-Faktor-Authentifizierung_schuetzen-Sicherheit-8679969.html, abgerufen am 03.12.2018

[12] Blog Botfrei: https://blog.botfrei.de/2015/11/2-faktor-authentifizierung-was-ist-das-eigentlich/, abgerufen am 01.12.2018

[13] Heise: https://www.heise.de/mac-and-i/meldung/NFC-als-zweiter-Faktor-fuer-das-iPhone-4055166.html, abgerufen am 01.12.2018

Kapitel 3

Analyse

Dieses Kapitel stellt die Basis für die Implementierung des Passwortmanagers dar. In Kapitel 1.1 wurden bereits erste Prinzipien eines guten Passwortmanagers erörtert. Dazu gehört die Entwicklung als Open-Source-Software, Cross-Platform-Verfügbarkeit und das Betreiben von Code-Sharing. Bei der Entwicklung von Open-Source-Software spielt die Wahl einer adäquaten Lizenz eine große Rolle. Deshalb werden in Kapitel 3.1 wichtige Auswahlkriterien festgelegt und mögliche Kandidaten vorgestellt. Anschließend wird eine Lizenz ausgewählt und die Auswahl anhand der gewählten Kriterien begründet. Um auch die rechtliche Relevanz der Open-Source-Lizenzen aufzuzeigen, werden Ereignisse der jüngsten Vergangenheit diskutiert. Die in der Motivation bereits genannten Prinzipien eines guten Passwortmanager waren nicht nicht tiefgreifend genug. Deshalb widmet sich das Kapitel 3.2 der Entwicklung weiterer Prinzipien, die von einem guten Passwortmanager erfüllt werden sollten, und der detaillierten Einzelbetrachtung der Manager. Abschließend werden wichtige Erkenntnisse aus dem Vergleich für den eigenen Passwortmanager erörtert und Anforderungen für Implementierungsdetails, die Technologiewahl und Konzeptionsphase definiert.

3.1 Lizenzauswahl

Die Lizenzauswahl ist eine der letzten Etappen, bevor eine Anwendung veröffentlicht wird. Der Vergleich der Passwortmanager zeigt, dass quelloffene Anwendungen dadurch einfach auf ihre Sicherheit untersucht werden können. Dem Kerckhoffschen Prinzip folgend, soll eine kryptografische Applikation quelloffen sein, damit die kryptografische Datenverarbeitung überprüft und Fehler frühzeitig erkannt werden können. Kryptografische Systeme sollen durch die Sicherheit der Algorithmen überzeugen und nicht durch den Umstand, dass die Verfahren nicht bekannt sind. Deshalb ist es zwingend erforderlich, dass eine adäquate Open-Source-Lizenz gefunden wird, damit der Status als quelloffene Software gewahrt bleibt und die Anwendung in sicherheits-

kritischen Aspekten gepflegt sowie weiterentwickelt werden kann.

3.1.1 Kriterien

Um die geeignete Lizenz für den Passwortmanager auszuwählen, müssen zuerst Kriterien, anhand derer die Lizenzen verglichen werden, spezifiziert werden. Die Kriterien werden vom Kerckhoffschen Prinzip abgeleitet, da dieses Prinzip wichtige Eigenschaften für kryptografische Systeme definiert. Wichtig ist dabei zu aller erst, dass die Lizenz für lokal betriebene Anwenderprogramme geeignet ist. Einige Lizenzen fokussieren sich beispielsweise auf Bibliotheken, die aber nicht für ein Anwenderprogramm, das in dieser Thesis entwickelt werden soll, geeignet sind. Ziel der Auswahl der Open-Source-Lizenz ist es, dass möglichst viele Leute von den Erkenntnissen und Konzepten dieser Arbeit profitieren können. Deshalb ist es wichtig, dass, falls die Applikation modifiziert wird, der Quellcode ebenfalls öffentlich gemacht wird. Damit folgen auch Modifikationen der Anwendung dem Kerckhoffschen Prinzip. Dabei ist es irrelevant, ob die Änderungen über den Upstream des ursprünglichen Repositories bereitgestellt, oder in einem separaten Repository zur Verfügung gestellt werden. Zudem muss das veränderte Produkt unter der gleichen Lizenz veröffentlicht werden. Damit wird sichergestellt, dass jegliche Modifikation des Produktes nicht mit weniger strikten Lizenzen lizenziert werden dürfen. Damit der freie Code auch wirklich frei bleibt, ist es wichtig, dass der Code nicht in proprietäre Anwendungen integriert werden darf, um die freie Software zu fördern und zudem zu garantieren, dass Verbesserungen auch an andere Nutzer weitergegeben werden.
Die Nutzung der Software soll nicht eingeschränkt werden, also kommerzielle sowie private Nutzung und auch Modifikation erlaubt sein. Dies erhöht nicht nur die Reichweite, da sowohl Unternehmen als auch Privatpersonen angesprochen werden. Dabei müssen natürlich auch die anderen zuvor genannten Regeln eingehalten werden. Da die Software sicherheitskritisch ist, muss natürlich die Frage der Haftung und der Gewährleistung betrachtet werden. Die Lizenz soll den Anspruch auf Gewährleistung und Haftung gegenüber dem Entwickler explizit ausschließen, sodass keine rechtliche Verfolgung geschehen kann.

Die Kriterien werden nun kurz und prägnant zusammengefasst:

- Für lokal betriebene Anwenderprogramme
- Modifikation uneingeschränkt möglich
- Modifizierter Code muss ebenfalls veröffentlicht werden
- Modifiziertes Produkt muss unter selber Lizenz verbreitet werden

- Darf nicht in proprietäre Anwendungen integriert werden
- Uneingeschränkte private und kommerzielle Nutzung
- Freisprechung von Haftbarkeit und Gewährleistung für Funktionalität der Anwendung

3.1.2 Kandidaten

Im nachfolgenden Teil werden die Kandidaten vorgestellt und ihre jeweiligen Kerneigenschaften dargelegt. Anschließend wird eine Auswahl auf Basis der zuvor genannten Kriterien getroffen. Abschließend sollen kurz rechtliche Probleme mit Open-Source-Lizenzen in der jüngeren Vergangenheit beleuchtet werden.

GNU GPLv3 Die GNU GPLv3[1] (GNU General Public License Version 3) ist eine der meist verbreitetsten Lizenzen. Die Lizenz enthält sowohl Rechte als auch Auflagen für den Nutzer der lizenzierten Software. Der Nutzer erhält das Recht, so lange die Lizenz dabei intakt bleibt, Kopien von der Software zu erstellen, Modifikationen vorzunehmen, die Software privat und kommerziell zu nutzen oder auch weiterzuverteilen, wie folgendem Auszug zu entnehmen ist:
„*You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force.*“
Die GPL gewährt auch das Recht die genutzten Patente der Software mitzunutzen (Abschnitt 11, „Patents“). Modifikationen müssen protokolliert und kenntlich gemacht werden, damit Fehler nur an Autoren der betroffenen Version angesprochen werden. Der Abschnitt in der Lizenz lautet wie folgt:
„*For both users' and authors' sake, the GPL requires that modified versions be marked as changed [...].*“
Die Auflagen oder Pflichten, die die GPL dem Nutzer auferlegt, werden auch *Copyleft* genannt. Der Nutzer wird verpflichtet, dass die Software, falls verändert und vertrieben, ebenfalls veröffentlicht werden muss. Dabei muss die modifizierte Software, oder die Software, die eine GPLv3-lizenzierte Software enthält, auch unter GNU GPLv3 Lizenz stehen (Abschnitt 5c, „Conveying Modified Source Versions“). Aufgrund dieser Pflichten gilt die GPL im Allgemeinen als eine „strenge“ Lizenz. Des Weiteren muss die verwendete GPL-Software inklusive der Lizenz in einer Notiz erwähnt werden (Abschnitt 5d, „Conveying Modified Source Versions“). Die Software, die unter GPLv3 lizenziert wurde, darf nicht in proprietäre Anwendungen integriert werden, da sonst der Entwicklung der Open-Source-Welt geschadet werde, wie dem letzten Abschnitt der Lizenz entnommen werden kann:

[1] GNU Project: https://www.gnu.org/licenses/gpl-3.0-standalone.html, abgerufen am 18.11.2018

„*The GNU General Public License does not permit incorporating your program into proprietary programs.*“
Abschnitt 15 „Disclaimer of Warranty“ der Lizenz befreit den Autor der Software von Haftbarkeit und Gewährleistung für die Funktionalität der Anwendung.
Ein populäres Beispiel für eine Software unter der GPL, aber jedoch in Version 2, ist der „Linux-Kernel“.

GNU LGPLv3 Die GNU LGPLv3[2] (GNU Lesser General Public License Version 3) ist eine Modifikation der GPLv3. Der ursprüngliche Name der LGPLv3 war „Library General Public License Version 3“, da die Einschränkung der GPL dazu führte, dass die Software nicht in proprietäre Anwendung integriert werden kann, denn die proprietäre Software müsste sonst auch unter GPL lizenziert werden. Mit der LGPL darf die freie Software in andere Software integriert werden, auch wenn diese proprietär ist:
„*You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work [...].*“. Es muss dann aber zwingend in der Anwendung ersichtlich sein, dass die Bibliothek verwendet und unter welcher Lizenz die Bibliothek veröffentlicht wurde.
Unter der LGPLv3 ist beispielsweise die UI-Bibliothek „Qt“ lizenziert.

GNU AGPLv3 Die GNU AGPLv3[3] ist ebenfalls von der GPLv3 abgeleitet und steht für „GNU Affero General Public License Version 3“. Diese Lizenz ergänzt eine Klausel für Software, die nur auf den Servern betrieben und für die eine Netzwerkverbindung benötigt wird. Bei der AGPLv3 gilt die Nutzung einer veränderten Software als Weiterverbreitung, weshalb der Quellcode offengelegt werden muss. Diese Anwendungen müssen den Quelltext zur Verfügung stellen, falls bei der Nutzung der Server-Software Kopien auf dem eigenen Gerät von AGPLv3-lizenzierten Dateien angelegt werden (Abschnitt 6, „Conveying Non-Source Forms“). Die GPLv3 ist in diesem Aspekt nicht exakt und würde eine Bereitstellung des Quelltextes außerhalb der Applikation, beispielsweise über GitHub, erlauben.
Die bekannte Cloudlösung „Nextcloud“ und ihr Vorfahre „Owncloud“ stehen unter GNU AGPLv3 Lizenz.

BSD Die Lizenz Berkeley Software Distribution (BSD)[4] gilt als eine freiere Lizenz in der Community, denn diese enthält kein Copyleft. Bei der BSD-Lizenz wurden ver-

[2]GNU Project: https://www.gnu.org/licenses/lgpl-3.0-standalone.html, abgerufen am 18.11.2018
[3]GNU Project: https://www.gnu.org/licenses/agpl.txt, abgerufen am 18.11.2018
[4]Open Source Initiative: https://opensource.org/licenses/BSD-3-Clause, abgerufen am 18.11.2018

schiedene Versionen veröffentlicht, jedoch sind nur die „2-Clause“- und „3-Clause“-BSD-Lizenzen aktuell. Die ursprüngliche BSD-Lizenz aus 1990 enthielt noch eine Werbeklausel und ist als „4-Clause“-BSD-Lizenz bekannt. Das Werbematerial musste eine spezifische Notiz enthalten, die darauf hinwies, dass die Software Quellcode von der Universität Berkley und weiteren Entwicklern enthielt. Diese wurde 1999 entfernt und seitdem wird die aktuelle „3-Clause“-Lizenz genutzt. Sowohl die bereits angesprochene „4-Clause“-BSD-Lizenz als auch die simplifizierte „2-Clause“-BSD-Lizenz finden in der praktischen Anwendung nur selten Aufmerksamkeit, weshalb im Folgenden die „3-Clause“-Lizenz näher betrachtet wird.
Die BSD-Lizenz gliedert sich in drei kompakt formulierte Regeln. Laut den ersten beiden Regeln muss bei Nutzung oder Weiterverbreitung, sei es in binärer Form oder als Quelltext, eine Copyright-Notiz hinterlegt werden. Der Hinweis kann bei der Verbreitung in binärer Form auch in der Dokumentation oder anderweitigem Material platziert werden. Der Entwickler kann den Code des Weiteren beliebig nutzen, verändern und vertreiben. Die Software muss nicht erneut veröffentlicht werden und die Lizenz muss im Falle der Veröffentlichung gleich bleiben.
Abgeschlossen wird der Lizenztext durch den Hinweis, dass der Entwickler der Software keine Funktionalität gewährleistet und für diese auch nicht haftbar gemacht werden kann. Der betreffende Abschnitt lautet:
„*THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS „AS IS“ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED*“
Bekannte Software, die unter BSD lizenziert wurde, ist die In-Memory-Datenbank „Redis“.

MIT Die MIT-Lizenz[5] ist sehr kurz gefasst und räumt dem Nutzer, genau wie die GPLv3, das Recht ein, die Software beliebig zu nutzen, zu modifizieren, oder zu verbreiten. Die Lizenz enthält kein Copyleft, sodass eine erneute Veröffentlichung bei Veränderung nicht notwendig ist. Wie bei der BSD-Lizenz, bleibt die MIT-Lizenz bestehen, wenn die Software weiterverteilt oder modifiziert wird. Wie bei allen anderen Lizenzen kann der Nutzer auch hier nicht für die Funktionalität haftbar gemacht werden, ebenso bietet der Entwickler oder der Verbreiter keine Gewährleistung für die Funktionalität. Auch hier muss eine Copyright-Notiz vorhanden sein, entweder mitgeliefert als Text-Datei oder in dem Programm aufrufbar. Die Lizenz erlaubt durch die Freiheit auch die Software in proprietäre Anwendungen zu integrieren.

[5] Open Source Initiative: http://www.opensource.org/licenses/MIT, abgerufen am 18.11.2018

Viele Programmiersprachen, Bibliotheken und Frameworks nutzen die MIT-Lizenz. So auch die in dieser Thesis verwendeten Bibliotheken *Electron*, *Node.js* und *React*.

Apache 2.0 License Die Apache 2.0 Lizenz[6] ähnelt, in den für den hier angestrebten Vergleich festgelegten Kriterien, der MIT- und BSD-Lizenz, sodass die Software beliebig genutzt, verändert und weiterverbreitet werden darf (Abschnitt 2, „Grant of Copyright License"). Auch das von der GPL bekannte Copyleft ist nicht vorhanden. Die Apache-Lizenz erlaubt die Nutzung von Patenten und schreibt eine Auflistung der Modifikationen vor, verbietet aber explizit die Benutzung von Markenzeichen, sogenannten „Trademarks". Wie bei anderen Lizenzen auch muss eine Copyright-Notiz eingepflegt werden und der Entwickler kann rechtlich für Fehler mit der Software nicht belangt werden.
Die von Google entwickelte Python-Bibliothek für Deep-Learning *TensorFlow* ist beispielsweise unter Apache 2.0 lizenziert.

Mozilla Public License 2.0 Die Mozilla Public License 2.0[7], auch häufig MPL genannt, enthält, im Vergleich zur GPL, ein schwächeres Copyleft. Dieses unterscheidet die Art der Modifikation an der Software. Werden bestehende Dateien verändert und handelt es sich um eine Verbreitung des Quellcodes, so müssen diese ebenfalls unter der Mozilla Public License lizenziert werden. Neue Dateien können aber beliebig lizenziert werden, solange die gewählte Lizenz mit den Lizenzen des benutzten Codes vereinbar ist:
„*You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software.*"
Dadurch soll die Kombination unterschiedlich lizenzierter Quellcodes vereinfacht werden. Wird eine ausführbare Anwendung vertrieben, müssen die unter der MPL lizenzierten Code-Teile veröffentlicht werden. Jeglicher andere Code, der durch einen Entwickler hinzugefügt wurde, kann dann beispielsweise unter eine proprietäre Lizenz gestellt werden (Abschnitt 3.2, „Distribution of Executable Form").
Zum Beispiel die Office-Suite *LibreOffice* ist unter der MPL lizenziert.

Unlicense Die Unlicense-Lizenz[8] überschreibt die Software der Öffentlichkeit und entzieht dem Autor jedwede Rechte an dem Code. Der Code darf privat als auch kommerziell genutzt werden. Das Weiterverbreiten ist ebenso gestattet, eine Veröffentlichung des Codes muss nicht erfolgen. Diese Rechte werden in der Lizenz durch folgenden Text eingeräumt:

[6] Apache: http://www.apache.org/licenses/LICENSE-2.0, abgerufen am 18.11.2018
[7] Mozilla: https://www.mozilla.org/en-US/MPL/2.0/, abgerufen am 18.11.2018
[8] Unlicense: https://unlicense.org/, abgerufen am 18.11.2018

„*Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.*“
Änderungen müssen nicht dokumentiert werden, ebenso sind keine Copyright-Notizen vonnöten. Die Garantie und Haftung werden durch die Lizenz im letzten Abschnitt ebenfalls ausgeschlossen. Die *Creative Commons Lizenz*, die das Werk ebenfalls der Öffentlichkeit zur Verfügung stellt (explizit CC0-Lizenz), verfolgt das selbe Ziel, wird aber aufgrund des Lizenz-Textes zur Nutzung empfohlen.

Die Tabelle 3.1 stellt den Lizenzen alle Vergleichskriterien gegenüber, sodass die Ergebnisse besser ersichtlich sind.

Tabelle 3.1: Lizenzen im Vergleich

Eigenschaft	**GPLv3**	**LGPLv3 (Bibliotheken)**	**AGPLv3 (Netzwerk)**	**BSD**	**MIT**	**Apache 2.0**	**MPL**	**Unlicense**
Für lokal betriebene Anwenderprogramme	•	–	–	•	•	•	•	•
Modifikation uneingeschränkt möglich	•	•	•	•	•	•	•	•
Modifizierter Code muss ebenfalls veröffentlicht werden	•	•	•	–	–	–	•	–
Modifiziertes Produkt muss unter selber Lizenz verbreitet werden	•	•	•	–	–	–	–	–
Darf nicht in proprietäre Anwendungen integriert werden	•	–	•	–	–	–	–	–
Uneingeschränkte private und kommerzielle Nutzung	•	•	•	•	•	•	•	•
Freisprechung von Haftbarkeit und Gewährleistung für Funktionalität der Anwendung	•	•	•	•	•	•	•	•

• Kriterium vollständig erfüllt
– Kriterium nicht erfüllt

3.1.3 Auswahl

Für die Anwendung der Thesis wurde die **GNU GPLv3** ausgewählt. Ausschlaggebend ist hier nicht nur, dass die Lizenz hervorragend zum Vertrieb einer lokalen Anwendung geeignet ist, sondern auch, dass mit der Lizenz ein wertvoller Beitrag zur Open-Source-Community geleistet wird. Denn durch das Copyleft wird sichergestellt, dass Weiterentwicklungen und Modifikationen ebenfalls veröffentlicht werden. Elementar ist auch, dass Veränderungen ebenfalls unter GPLv3-Lizenz stehen müssen

und somit die Lizenz fortbesteht. Dies ist ein sehr wirksames Mittel, um Fortschritt zu sichern und zu verhindern, dass Open-Source-Software weiterentwickelt wird und in proprietären Anwendungen genutzt wird. Vor allem das Copyleft wird in der Community stark diskutiert. Einige Nutzer empfinden die GPL als streng, da es viele Pflichten für die Entwickler auferlegt, die beispielsweise bei der BSD-Lizenz nicht vorhanden sind. Dieses stellt aber nicht nur Transparenz für den Entwickler sicher, sondern auch für den Verbraucher, der zu jeder Zeit auch von einer modifizierten Software den Quellcode einsehen kann.
Der Vergleichstabelle 3.1 kann entnommen werden, dass die GPLv3 klar hervorsticht, aber auch einige Gemeinsamkeiten mit der LGPLv3, AGPLv3 und MPL aufweist. Sowohl LGPLv3 als auch AGPLv3 eignen sich aufgrund der Spezialisierung auf Bibliotheken bzw. Serveranwendungen nicht für den hier zu entwickelnden Passwortmanager, der als lokal betriebene Anwendung konzipiert ist. Dadurch, dass die LGPLv3 für Bibliotheken angedacht ist, die eventuell auf proprietären Code zugreifen müssen oder in proprietären Code integriert werden kann, erfüllt sie den Grundgedanken nach einer Stärkung der Open-Source-Welt, indem möglichst viel Code veröffentlicht wird, nicht.
Die Mozilla Public License 2.0 erlaubt es ebenfalls, dass der Code in proprietäre Anwendungen integriert und auch geändert werden darf. Diese Eigenschaft und das fehlende Copyleft ermöglichen es, dass Code nach Veränderung ggf. nicht veröffentlicht werden muss oder nur für eine proprietäre Anwendung weiterentwickelt wird, was dem Open-Source-Gedanken nachhaltig Schaden zufügt. Des Weiteren gibt es bei der MPL keine Anforderungen, wie Änderungen zu dokumentieren sind. Die GPLv3 fordert eine Protokollierung der gemachten Änderungen an der Software bei der Veröffentlichung. Dies fördert die Transparenz, sodass genau verfolgt werden kann, welche Dateien vom ursprünglichen Projekt enthalten sind.
Aus den genannten Gründen wird der Passwortmanager unter der GPLv3-Lizenz entwickelt.

3.1.4 Rechtliche Probleme mit Open-Source-Lizenzen in der Vergangenheit

Die GPL erhielt in letzter Zeit viel mediale Aufmerksamkeit. Denn Linux-Kernel-Entwickler Patrick McHardy verklagte zahlreiche Firmen vor Gericht, wenn diese ein Linux-System für Verbraucherwaren anpassten und verkauften, aber den Quellcode nicht veröffentlichten[9]. Dies kann auf die kleine Größe der Entwicklungsabteilung und eventuell Unwissenheit oder sogar auf Vorsatz zurückgeführt werden. Die Community

[9]DerStandard: https://www.derstandard.de/story/2000075504111/gerichtsverfahren-um-linux-abmahnungen-verunsichert-branche, abgerufen am 02.12.2018

sieht die Verfahrenswelle gespalten, einige sehen McHardy im Recht, der in ihren Augen versucht das geltende Recht durchzusetzen und Firmen für Open-Source-Lizenzen zu sensibilisieren, andere empfinden dies als persönliche Bereicherung oder sind gar GPL-Verweigerer[10].
Hier manifestiert sich die in Kapitel 3.1.2 genannte Auffassung zu den „strengen“ und „freien“ Lizenzen. Oft wird erwähnt, dass eine OSS-Lizenz nur frei sei, wenn es keine Einschränkungen oder Pflichten bei der Nutzung und Weiterentwicklung gäbe. Dementsprechend wird von GPL-Verweigerern oft auf die BSD-, Apache-, oder MIT-Lizenz verwiesen. Die Wahl der GPLv3 für Open-Source-Software ist aber die richtige, da die Weiterentwicklung nicht durch Vertrauen auf denjenigen, der die Software modifiziert, gesichert werden kann. Der Begriff „freie Lizenz“ wird im Sinne von „freier Software“ und nicht „Open Source“ verstanden, was jedoch zwei unterschiedlich weit gefasste Begriffe sind, siehe Begriffsdefinition von Open Source in Kapitel 2.1.10.
Interessant ist auch der Umgang mit der GPL-Lizenz von Hardware-Herstellern zum Beispiel bei dem ARM-SoC-Hersteller *Allwinner*. Dieser soll Meldungen zufolge in das verkaufte Produkt LGPLv3-Software integriert und modifiziert, aber nicht veröffentlicht haben[11]. Als dies bekannt wurde, wurden Funktionsnamen geändert, um die Nutzung dieser Bibliotheken zu verschleiern. Der Quellcode wurde bis heute nicht veröffentlicht und Allwinner hat kein Stellungnahme zu der Thematik veröffentlicht[12].
Ein weiteres Beispiel ist die gerichtliche Auseinandersetzung von Entwicklern der Anwendung `iptables` mit einer Firma, die die unter der GPL lizenzierte Software in einer proprietären Anwendung nutze (Metzger, 2015, S. 204). Der Richter hat der Unterlassungsklage stattgegeben, sodass die Firma den Quellcode der Anwendung offenlegen musste. Dies zeigt für den Prototypen, dass die GPLv3-Lizenz leider nicht immer sicherstellen kann, dass auch alle Entwickler diese befolgen. Nichtsdestotrotz nimmt die GPLv3 die Entwickler mehr in die Pflicht als andere Lizenzen, sodass der zurückfließende Code-Anteil höher ist als bei den Konkurrenten, die die Veröffentlichung auf freiwilliger Basis vorsehen. Die Zukunft wird aber zeigen, ob die Entwickler zunehmend die Bedingungen der GPL akzeptieren, oder es weitere Gerichtsverfahren zur Durchsetzung der GPL geben wird.

[10]Pro Linux: https://www.pro-linux.de/news/1/25090/patrick-mchardy-gpl-durchsetzung-zur-pers%C3%B6nlichen-bereicherung.html, abgerufen am 02.12.2018
[11]Golem: https://www.golem.de/news/arm-soc-allwinner-soll-lgpl-verletzungen-verschleiern-1503-113086.html, abgerufen am 01.12.2018
[12]Golem: https://www.golem.de/news/arm-soc-allwinner-verschleiert-lizenzverletzungen-noch-weiter-1503-113184.html, abgerufen am 02.12.2018

3.2 Vergleich Passwortmanager

Auf dem Markt sind schon zahlreiche Passwortmanager etabliert. Die Autoren des Magazins c't haben bereits fünfzehn bekannte Passwortmanager verglichen, setzten dort jedoch eher den Fokus auf Funktionen, die entweder kostenlos oder kostenpflichtig zur Verfügung stehen, und auf einen möglichst einfachen Betrieb (Poimann, 2018). Im Nachfolgenden werden fünf Manager in einer Detailansicht vorgestellt. In der Einleitung wurden bereits erste Prinzipien definiert, die hier für die Vorstellung der einzelnen Manager genutzt werden. Daher wird bei der Einzelbetrachtung der Manager zunächst die Plattformverfügbarkeit und anschließend die Art der Datenhaltung diskutiert. Des Weiteren wird die Lizenz und damit einhergehend die Möglichkeit der Einsicht in den Quellcode betrachtet. Abschließend wird, da einige Passwortmanager kostenpflichtig sind, die Preisstruktur dargelegt, gefolgt von der Analyse von veröffentlichten Schwachstellen und Sicherheitsaudits. Anschließend werden weiterreichende Vergleichskriterien technischer Natur in Kapitel 3.2.6 festgelegt, anhand derer die Passwortmanager verglichen werden, die aber auch für die Implementierung des eigenen Passwortmanagers relevant sind.

3.2.1 LastPass

Plattformen LastPass ist ein bekannter Online-Passwortmanager. Der Manager stellt hierbei eine Browser-Erweiterung dar, die sich mit dem Online-Dienst verbindet. Für die bekannten Browser wie Chrome, Opera, Firefox und Internet Explorer sind Erweiterungen vorhanden. Für die mobilen Plattformen bietet LastPass eine App ohne Browser-Integration an. Durch die Erweiterung hat der Nutzer die Möglichkeit direkt im Browser auf seine Passwörter zuzugreifen. Der Browser erkennt über die URL die Plattform, auf der der Nutzer sich anmelden möchte und füllt automatisch Nutzername und Passwort in die Texteingabefelder ein. Für den Nutzer erhöht sich so der Komfort, da die Anzahl der Klicks, die ein Benutzer zum Einloggen bei einer Plattform benötigt, erheblich sinkt.

Datenhaltung Zwar sind alle Passwörter durch die Synchronisierung jederzeit verfügbar, werden aber immer auf den firmeneigenen Servern abgelegt. Der Nutzer hat nicht die Möglichkeit den Speicherort der Daten zu wechseln. Ebenfalls muss der Nutzer darauf vertrauen, dass der Anbieter vertrauenswürdig mit den Daten umgeht und vor Angriffen sichert. Nicht nur bei der Speicherung der Daten lässt der Entwickler Transparenz vermissen, auch der Quellcode der Anwendung ist nicht Open Source und somit für Andere nicht einsehbar.

Lizenz und Einsicht in Quellcode Dadurch, dass LastPass einer proprietären Lizenz unterliegt, kann keine Einsicht in den Quellcode unternommen werden.

Kosten LastPass kostet monatlich 2$ für Privatpersonen für eine Einzellizenz und 4$ für eine Familienlizenz für bis zu sechs Nutzer.

Schwachstellen und Audits In den letzten Jahren sind zahlreiche Schwachstellen aufgedeckt worden, die mit offenem Quellcode dem Kerckhoffschem Prinzip folgend wohl vermeidbar gewesen wären. Zudem kann der Quellcode nicht auf weitere, noch nicht behobene, Schwachstellen untersucht werden. Im Jahre 2011 wurden verdächtige Aktivitäten auf den Servern von LastPass beobachtet, der sich im erhöhten Downstream-Traffic im Vergleich zum Upstream geäußert hat. Der Hersteller bemühte sich um Beschwichtigung der Kunden und verwies auf das sichere Verschlüsseln und Hashing der Passwörter. Nichtsdestotrotz mussten alle Nutzer ihr Master-Passwort ändern. Laut Hersteller konnte nicht ermittelt werden, welcher Angriffsvektor genutzt wurde, was genau den erhöhten Netzwerkverkehr verursachte oder welche Daten gelesen wurden[13]. Vier Jahre später wird eine weitere Aktivität publik, bei der Passworthashes, Salts und E-Mail-Adressen entwendet wurden. Auch hier verweist LastPass auf die Sicherheit der gespeicherten Daten, empfiehlt aber das Ändern des Master-Passworts. Als Reaktion auf die Vorfälle wurde ein Bug-Bounty-Programm gestartet. Dies animiert Entwickler und Tester den Passwortmanager auf Schwachstellen zu untersuchen und für ein Preisgeld einzureichen[14][15]. Im Juli 2016 wurden Fehler in der Browser-Erweiterung von LastPass bekannt[16]. Ein Fehler im URL-Parser sorgte dafür, dass Angreifer auf einer Seite LastPass dazu bewegen konnten, Login-Daten einer anderen Domäne einzufügen. Die URL `http://example.com/@twitter.com/@test.php` sorgte bei dem URL-Parser, der nur das letzte Vorkommen des @-Symbols berücksichtigt, um dieses Zeichen mit `%40` zu kodieren, sodass `twitter.com` die Domäne ist, für die Passwörter in LastPass gesucht werden. Folglich wurde die Domäne `example.com` als Nutzername behandelt und ignoriert[17]. Weitere Schwachstellen wurden aber nicht detailliert beschrieben. Es war Angreifern wohl möglich über Phishing und manipulierte Webseiten an Nutzerdaten zu gelangen, oder im Firefox alle Funktionen der LastPass-Erweiterung im

[13] LastPass: https://blog.lastpass.com/2011/05/lastpass-security-notification.html/, abgerufen am 23.07.2018

[14] LastPass: https://blog.lastpass.com/2015/06/lastpass-security-notice.html/, abgerufen am 23.07.2018

[15] Ars Technica: https://arstechnica.com/information-technology/2015/06/hack-of-cloud-based-lastpass-exposes-encrypted-master-passwords/, abgerufen am 23.07.2018

[16] LastPass: https://blog.lastpass.com/2016/07/lastpass-security-updates.html/, abgerufen am 24.07.2018

[17] Detectify: https://labs.detectify.com/2016/07/27/how-i-made-lastpass-give-me-all-your-passwords/, abgerufen am 24.07.2018

Hintergrund zu benutzen. Darüber, wie die Verwundbarkeiten exakt ausgenutzt und diese geschlossen wurden, hält sich LastPass bedeckt.
Im Jahr 2017 sind erneut Schwachstellen gefunden worden, der Fehler des URL-Parsers des vergangenen Jahres wurde nicht in allen aktiven Branches eingepflegt, sodass der Patch für die inzwischen als veraltet markierte und nicht mehr weiter entwickelte Version 3.X nachgereicht wurde. Im Onbording-Prozess konnten zudem Nutzerdaten entwendet werden. Auch hier wurde ein Patch veröffentlicht[18]. Wenige Tage später wurde eine weitere Schwachstelle bekannt, durch die Zugriff auf Funktionen von LastPass möglich war. LastPass spricht hier von zwei Arten von JavaScript im Browser: „Content script“ und dem regulären JavaScript, das durch den Betreiber der Seite eingepflegt wurde. „Content scripts“ haben nur Zugriff auf den DOM und erhalten keinen Zugriff auf Variablen oder Funktionen des regulären Skripts. Umgekehrt gilt das auch für das reguläre Skript. Diese Abgrenzung konnte jedoch überwunden werden, sodass eine umfängliche Neustrukturierung der Anwendung notwendig wurde[19][20].
Ebenfalls im März 2017 untersuchte das Fraunhofer Institut in Darmstadt zahlreiche Passwortmanager auf Schwachstellen und wurde bei LastPass an drei Stellen fündig[21]. Dabei wurde die Android-App von LastPass untersucht und aufgedeckt, dass kryptografische Schlüssel hart kodiert, das Master-Passwort, zwar verschlüsselt, aber einfach einsehbar war. Zudem sei die App anfällig für Man-In-The-Middle-Angriffe gewesen. Die integrierte Suche im App-Browser implementiert eine Funktion, die Vorschläge über Suchbegriffe ausgibt, die aber über einen ungeschützten HTTP-Request an Google ermittelt werden. Ein Angreifer kann beispielsweise den genutzten WLAN-Access-Point kompromittieren und dadurch Request als auch Response manipulieren, um den Nutzer auf manipulierte Webseiten zu führen. Alle drei Schwachstellen wurden LastPass gemeldet und zeitnah behoben.

3.2.2 KeePass

Plattformen KeePass ist ein populärer Open-Source-Passwortmanager mit einer großen Nutzergemeinde, dessen erste Version im November 2003 veröffentlicht wurde. Die aktuelle Version ist in C# implementiert und unter Windows lauffähig. Durch die große Community wurden ebenfalls Versionen für MacOS, Linux sowie Android

[18] LastPass: https://blog.lastpass.com/2017/03/important-security-updates-for-our-users.html/, abgerufen am 24.07.2018
[19] Tavis Ormandy: https://twitter.com/taviso/status/845717082717114368, abgerufen am 25.07.2018
[20] LastPass: https://blog.lastpass.com/2017/03/security-update-for-the-lastpass-extension.html/, abgerufen am 25.07.2018
[21] GHacks: https://www.ghacks.net/2017/03/02/security-issues-found-in-nine-password-managers-for-android-lastpass-dashlane/, abgerufen am 25.07.2018

und iOS bereitgestellt. Mittels Browser-Erweiterung durch unabhängige Entwickler kann eine ähnliche Funktionalität gewährleistet werden, wie mit LastPass. Mittels der Erweiterung erhält der Nutzer lesenden Zugriff auf die KeePass-Datenbank und kann Anmeldedaten in das Formular einfügen. Überprüft wurde die Vesion 2.39 von KeePass.

Datenhaltung KeePass bietet die Möglichkeit der Synchronisierung. Dabei werden zwar standardmäßig keine Dienste wie Dropbox oder Google Drive unterstützt, jedoch aber zahlreiche Protokolle wie File Transfer Protocol (FTP), SSH File Transfer Protocol (SFTP), FTP over SSL (FTPS) oder WebDAV. Dadurch kann der Nutzer selbst einen Server betreiben, der diese Protokolle zu Verfügung stellt und hat somit vollen und alleinigen Zugriff auf die Passwortdatenbank.
Die Software steht kostenlos zur Verfügung, ebenso wie die durch die Community entwickelten Applikationen. Die Anzahl der Nutzer ist hierbei nicht limitiert, Nutzer müssen sich das Master-Passwort aber teilen.

Lizenz und Einsicht in Quellcode KeePass verfügt über eine GPLv3 Lizenz. Verbreitete Lizenzen werden in Kapitel 3.1 genauer erläutert und verglichen. Nutzer können den Quellcode von KeePass sowie den inoffiziellen KeePass-Varianten einsehen und Einblicke in die Datenverarbeitung erhalten.

Kosten Dadurch, dass KeePass mit einer freien Lizenz vertrieben wird, kann jeder diesen Passwortmanager kostenlos ohne Einschränkungen nutzen. Neue Funktionen stehen somit sofort allen Nutzern zur Verfügung.

Schwachstellen und Audits Bezüglich Sicherheit wird bei KeePass eine Webseite angeboten, die publik gewordene Probleme und Fehler beschreibt und die Lösung derer verdeutlicht[22]. Das aktuelle KDBX-Format teilt eine KDBX-Datei in einen unverschlüsselten Header und einen verschlüsselten Body auf. Der Header kann so manipuliert werden, sodass die gespeicherten Daten nicht mehr lesbar sein können. Dieses Problem wurde durch eine „Header authentication“ gelöst, welche prüft, ob der Header valide und integer ist. Der Entwickler von KeePass weist aber darauf hin, dass nicht alle anderen Anwendungen, die das KDBX-Format unterstützen, beispielsweise die Implementierungen durch Dritte für Android, mit der Header-Änderung umgehen können, oder diese überhaupt berücksichtigen. Hier hätte eine Cross-Platform-Anwendung mit Code-Sharing für die Nutzer große Vorteile, denn die Änderung des Formats wäre gleich für alle Plattformen verfügbar gewesen. Der Nutzer muss auch nicht auf den Entwickler der Dritt-Anbieter-App warten, bis dieser, falls überhaupt,

[22]KeePass: https://keepass.info/help/kb/sec_issues.html, abgerufen am 25.07.2018

ein Update herausbringt. Des Weiteren wird die auf GitHub verfügbare Anwendung „KeeFarce“[23] thematisiert. KeeFarce kann durch DLL-Injection Passwörter im Klartext aus dem Arbeitsspeicher von KeePass auslesen. Der Entwickler von KeePass weist darauf hin, dass es gegen diese Art des Angriffes keine Lösung gibt, da dies einen spezialisierten Angriff auf den Passwortmanager darstellt. Gegen generische Spyware, wie beispielsweise Keylogger, wurde die Two-Channel Auto-Type Obfuscation (TCATO) implementiert. TCATO kopiert sensible Daten in die Zwischenablage und fügt die Daten an gewünschter Stelle ein, sodass der Keylogger nur die Tastenkombinationen für Kopieren und Einfügen ermitteln können. Auch die gern genutzte Anfälligkeit für Man-In-The-Middle-Angriffe beim Update-Prozess[24] wurde durch den Wechsel auf HTTPS und die Signierung der Pakete mittels RSA4096 (RFC 8017[25]) und SHA512 gelöst.
Die europäische Komission hatte jüngst ein Sicherheits-Audit von KeePass im Oktober 2016 durchgeführt und die Ergebnisse des Audits veröffentlicht (FOSSA Europäische Komission, 2016). Das Audit attestierte KeePass in der Version 1.3.5 14 Befunde, die anhand der 218 getesteten Punkte laut Prüfer als wenig angesehen werden können. Keine der Befunde überstieg die mittlere Schwere, sechs der Befunde sind informativer Natur. Die Tester monierten unter anderem, dass bei einer Funktion nicht geprüft wird, ob der String mit \0 endet, denn diese „Zero-Bytes“ terminieren einen String. Für die Funktion in KeePass bedeutet dies, dass es zu einem Buffer-Over-Read kommt, der die Anwendung zum Absturz bringen kann, falls der String nicht mit einem „Zero-Byte“ beendet wird. Vereinzelt wird auch auf Werte von Variablen zugegriffen, die zuvor nicht initialisiert wurden. Die drei weiteren Befunde der mittleren Dringlichkeit weisen unter anderem darauf hin, dass unsichere Zufallsgeneratoren wie `rand()` genutzt werden. Diese werden zwar nicht für sicherheitskritische Bereiche genutzt, werden aber aufgrund der Wichtigkeit trotzdem erwähnt. Das Audit hinterlässt einen durchweg positiven Eindruck. Es wurden keine schwerwiegenden Probleme gefunden und die genannten Befunde wurden innerhalb kürzester Zeit behoben.

3.2.3 1Password

Plattformen 1Password ähnelt LastPass, bietet aber viele Vorteile von KeePass. Neben einer Anwendung für Windows, MacOS, Android sowie iOS, sind auch Browser-Erweiterungen für eine nahtlose Integration in Anmeldeprozesse verfügbar. Eine Applikation für Linux-basierte Betriebssysteme ist nicht vorhanden.

[23] GitHub: https://github.com/denandz/KeeFarce, abgerufen am 25.07.2018
[24] Inc: https://www.inc.com/will-yakowicz/security-vulnerability-keepass-password-manager.html, abgerufen am 25.07.2018
[25] IETF: https://tools.ietf.org/html/rfc8017, abgerufen am 25.07.2018

Datenhaltung 1Password bietet Synchronisierung über den eigenen Dienst, aber auch die Möglichkeit die Daten lokal abzulegen oder über WiFi, iCloud und Dropbox zu synchronisieren. Der Nutzer kann somit entscheiden, wo die Nutzerdaten abgelegt werden.

Lizenz und Einsicht in Quellcode Die Anwendung ist, ebenfalls wie LastPass, proprietär, sodass der Quellcode nicht einsehbar ist.

Kosten Die Lizenz für Einzelpersonen kostet monatlich etwa 3$, eine Familienlizenz für bis zu sechs Personen 4,99$.

Schwachstellen und Audits 1Password wurde ebenfalls durch das Fraunhofer Institut für Sicherheit in der Informationstechnik (SIT) in Darmstadt überprüft[26]. Dabei wurde ein Fehler entdeckt, dass durch den URL-Parser Anmeldedaten, die für eine Unterdomäne A gespeichert wurden, auch bei Unterdomäne B eingefügt wurden. Ebenfalls wie bei LastPass wurden alle Webseitenanfragen des eingebauten Browsers über HTTP anstatt HTTPS durchgeführt. Der eingebaute Browser erlaubte zudem auch das `file:///`-Schema, sodass auch auf die privaten Dateien der Anwendung, unter anderem auch die Passwort-Datenbank, aufgerufen werden konnte. Des Weiteren wurden auch Titel und URL eines gespeicherten Passwortes nicht verschlüsselt. Hier wurden dem Hersteller der Software die gefundenen Fehler übermittelt und zeitnah durch diesen behoben.

3.2.4 Padlock

Plattformen Padlock ist ein Open-Source-Passwortmanager, der im Februar 2014 veröffentlicht und in der Version 2.7.2 überprüft wurde. Padlock bedient sich aktueller Web-Technologien, um die Applikationen für Windows, Linux, MacOS sowie Android und iOS bereitzustellen. Es ist ebenfalls eine Chrome-Web-App vorhanden, sodass der Passwortmanager in Chrome und in ChromeOS verwendet werden kann.

Datenhaltung Padlock operiert lokal und kann auch mit dem Padlock Cloud Service verbunden werden. Dieser, vom Hersteller zur Verfügung gestellte Dienst, ist kostenlos und nicht verpflichtend. Zudem ist der Server ebenfalls quelloffen, sodass der Nutzer hohe Transparenz genießt, was mit den Daten passiert, obwohl diese nicht auf dem eigenen Server gespeichert werden.

[26]Team SIK: https://team-sik.org/trent_portfolio/password-manager-apps/, abgerufen am 15.11.2018

Lizenz und Einsicht in Quellcode Die Anwendung ist unter GPLv3 lizenziert, sodass der Quellcode eingesehen und verändert werden darf. Dabei wird aber durch die Lizenz vorausgesetzt, dass diese Änderungen ebenfalls der Community zur Verfügung gestellt werden.

Kosten Die Client-Anwendung als auch die Server-Applikation sind kostenlos verfügbar. Auch weitere Funktionen sind für alle Nutzer direkt und kostenlos nutzbar.

Schwachstellen und Audits Padlock wurde durch das Unternehmen Cure53 ebenfalls im April 2016 eines Audits unterzogen (Heiderich, Aranguren, Inführ, & Fässler, 2016). Überprüft wurde die App für Android und iOS, die Chrome-Browsererweiterung und den Cloud-Server, um Passwörter synchronisieren zu können. Das Urteil fällt nicht so positiv aus, wie bei KeePass. Es ergaben sich 17 Befunde, von denen neun mittlerer und vier hoher Dringlichkeit sind. In allen getesteten Anwendungen von Padlock wurde ein nicht sicherer Pseudozufallszahlengenerator für kryptografische Verarbeitungen genutzt. Alle drei Anwendungen sind zudem auch durch Denial of Service (DNS)-Angriffe verwundbar. Das an den Server durch die Client-Anwendung übermittelte JSON-Objekt bietet ein Feld `iter` mit dem die Anzahl der Runden für die Verschlüsselung festgelegt wird. Ein Angreifer kann das Feld manipulieren und viel höhere Zahl wählen. Dies hat zum einen die Auswirkung, dass die Entschlüsselung bei heutigem Stand der Technik in naher Zukunft nicht abgeschlossen wird. Durch die falsche Iterationsanzahl kann zum anderen auch nie das ursprüngliche Passwort ausgegeben werden, wodurch durch diese Manipulation Datensätze korrumpiert werden. Unter Android und iOS beklagen die Tester, dass kein Pinning von öffentlichen Schlüsseln für HTTPS-Verbindungen geschieht, ohne das der Angreifer mit einem gefälschten Zertifikat einen Man-In-The-Middle-Angriff durchführen kann. Wenn die Synchronisierung genutzt wird, wurden relevante Informationen wie Server-URL, Krypto-Parameter, Email-Adresse des Cloud-Accounts und das zugehörige Token im Klartext in den Einstellungen abgelegt. Durch diese Daten hat der Angreifer die Möglichkeit den Passwortspeicher, der auf dem Server gespeichert ist, mit falschen Parametern zu verschlüsseln oder die Daten unbrauchbar zu machen. Spezifisch bei Android fehlt der Schutz vor Screenshots. Dies liegt darin begründet, dass Passwörter standardmäßig nicht unkenntlich gemacht sind und dass im Quellcode ein Flag nicht aktiv gesetzt wird. Nur unter Android ist die Padlock-App anfällig für „Tapjacking"-Angriffe. Beim Tapjacking[27] wird über der eigentlichen App durch eine dritte böswillige Anwendung ein transparentes Overlay angezeigt, das die Nutzerinteraktionen aufnimmt sowie protokolliert und für andere Zwecke nutzt, bevor die

[27] DevKnox: https://blog.devknox.io/tapjacking-android-prevent/, abgerufen am 26.07.2018

Befehle an die dahinter liegende Anwendung weitergeleitet werden. Auch dies lässt sich durch das Setzen eines Flags und das Implementieren eines Event-Handlers lösen. Die Server-Implementierung weist auch einige Fehler auf. Beispielsweise erhält der Nutzer unterschiedliche Fehlermeldungen ob ein Nutzer nicht aufgefunden wurde oder ob das Passwort nicht korrekt ist. Dies lässt für den Angreifer Rückschlüsse zu, welcher Teil der Anmeldedaten nicht korrekt ist. Dies kann gelöst werden, indem der Fehlertext für beide Fehlerarten identisch ist. Der Server weist eine schlechte Konfiguration auf, sodass dieser über das SSL3.0-Protokoll bzw. schwache TLS-Konfiguration für POODLE[28]- respektive Logjam[29]-Angriffe verwundbar ist. Diese und einige weitere Befunde wurden dem Entwickler eingereicht. Dieser behob die Probleme zeitnah und informierte über den Status mit einem Informationstext, wie das Problem jeweils gelöst wurde[30].

3.2.5 Passbolt

Plattformen Passbolt verfolgt einen anderen Ansatz bei den unterstützten Plattformen als die bisher vorgestellten Passwortmanager. Zentrales Element des Managers ist das Docker-Image des Servers, das einen Betrieb auf einer Vielzahl an Geräten und Betriebssystemen ermöglicht. Der Server kann durch Docker auf vielen Linux-Distributionen, MacOS und Windows betrieben werden. Dieses Docker-Image stellt zudem einen Web-Server bereit, sodass ein Abruf der Passwörter durch die Browser von Desktop- und Mobilgeräten geschehen kann. Ergänzend werden auch Browser-Erweiterungen für Firefox und Chrome bereitgestellt. Andere Plattformen oder Browser werden durch die Entwickler nicht unterstützt. Überprüft wurde die Anewndung in der Version 2.1.0.

Datenhaltung Die Datenhaltung geschieht zentral durch den Server. Der Nutzer hat den vollen und alleinigen Zugriff auf die Daten, sofern die Anwendung auf dem eigenen Server zur Verfügung gestellt wird. Nachteilig wirkt sich aber auch der Mangel an nativen Applikationen mit lokaler Datenhaltung aus, denn so kann der Manager nur genutzt werden, wenn eine Internetverbindung besteht. Passbolt stellt zur Zeit eine nur unzureichend dokumentierte REpresentational State Transfer (REST)-API zur Verfügung, mit der es schlecht möglich ist, einen eigenen Client für Passbolt zu implementieren, der beispielsweise eine Offline-Funktion enthält. Ein Blick auf GitHub zeigt, dass die API vor der Browser-Erweiterung stark im Vordergrund steht und gemeinsam mit dem Server gut gewartet wird.

[28]US-CERT: https://www.us-cert.gov/ncas/alerts/TA14-290A, abgerufen am 26.07.2018
[29]Weak Diffie-Hellman and the Logjam Attack: https://weakdh.org/, abgerufen am 26.07.2018
[30]Padlock: https://padlock.io/pentest-1604-notes/, abgerufen am 26.07.2018

Lizenz und Einsicht in Quellcode Passbolt ist ein weiterer quelloffener Manager, dessen Bestandteile Server, Docker-Image und Browser-Erweiterung jeweils unter AGPLv3 lizenziert sind.

Kosten Passbolt ist in der Community Edition kostenlos. Es können in dieser Edition zwar unbegrenzt viele Nutzer oder Instanzen des Servers angelegt werden, dennoch ist der Funktionsumfang begrenzt. Wer viele Komfortfunktionen und alle neue Funktionen sofort benutzen möchte, muss eine Lizenz für mindestens 19€ pro Monat erwerben.

Schwachstellen und Audits Über Passbolt sind keine Penetration-Tests oder Sicherheits-Audits veröffentlicht worden. Auch aufgedeckte Schwachstellen sind nicht auffindbar.

3.2.6 Vergleichskriterien

Für einen guten Vergleich sind die Kriterien essenziell. Der Vergleich soll zum einen die vorhandenen Manager untereinander vergleichen, aber auch zum anderen die Kriterien beleuchten, die für den Prototypen dieser Thesis interessant sind. Im Fokus dieser Thesis liegt die *gemeinsame Codebasis*. Deshalb ist es sehr interessant zu beleuchten, wie die Applikationen implementiert sind, sofern Informationen bekannt sind. Die *Codequalität* lässt Aussagen über die Sicherheit des Passwortmanagers zu. Die verwendeten *kryptografischen Verfahren* entscheiden darüber, ob eine Software Daten sicher, oder nicht sicher ablegen. Ebenso unterscheidet sich der *Funktionsumfang*, die die Passwortmanager bieten. Interessant wäre auch der Vergleich der Usability der verschiedenen Passwortmanager, jedoch liegt diese nicht im Fokus dieser Thesis. Es sind jedoch wissenschaftliche Arbeiten zur Überprüfung der Usability von klassischen Passwortmanagern verfügbar (Karole, Saxena, & Christin, 2011). Ein wichtiges Kriterium ist vor allem der Punkt *Lizenz* und damit einhergehend das Kriterium *Preis*. Bei diesem Vergleichskriterium ist der Betrieb für eine Person interessant, weshalb bei kostenpflichtigen Passwortmanagern immer das Abonnement für Einzelpersonen als Basis gewählt wurde.

3.2.7 Vergleich

Nun werden die Passwortmanager auf die zuvor genannten Kriterien hin untersucht. Fundierte Aussagen zum Code lassen sich leider nur bei den Passwortmanagern treffen, bei denen der Quelltext einsehbar ist.

Code-Sharing

LastPass ist eine proprietäre Anwendung und dadurch bleibt der Zugriff auf den Quelltext leider verwehrt. Die verbreiteten Anwendungen lassen durch die zugrunde liegende Programmiersprache aber eventuell Rückschlüsse auf das Code-Sharing zu. Die Software für den Desktop wird nur als Browser-Erweiterung verbreitet, für die mobile Plattformen iOS und Android ist jeweils eine native App verfügbar. Sowohl bei der Browser-Erweiterung als auch der Mobilanwendung ist Code-Sharing in einer Cross-Plattform-Anwendung jeweils möglich, weshalb keine definitive Aussage zum Thema Code-Sharing bei LastPass getroffen werden kann.
KeePass ist wie bereits erwähnt für Windows verfügbar und unterstützt in der Version 2.X auch Mono[31]. Mono ist eine Open Source Implementierung von Microsofts .NET-Framework, sodass KeePass auch unter Linux und MacOS betrieben werden kann. Demnach liegt der Anteil des Code-Sharings für die Desktop-Plattformen bei 100%. Plattform-spezifischer Code fällt hier nicht an, da die .NET-Implementierung alle Sonderfälle bei den Betriebssystemen berücksichtigt. Der Code kann aber nicht mit den durch die Community entwickelten Anwendungen geteilt werden. Somit kann eine durch Dritte bereitgestellte Software Schwachstellen aufweisen, die bei der Desktop-Implementierung, auch durch eventuell besseres Fachwissen der Entwickler, nicht auftreten.
Ähnlich wie bei LastPass, verhält es sich auch bei 1Password. Die identische Optik und Releasezeiten von großen Major-Versionen, die sich für MacOS und Windows nur um wenige Tage unterscheiden, könnten ein Indiz für eine Cross-Plattform-Anwendung mit Code-Sharing sein. Der Umstand, dass aber keine native Anwendung für Linux angeboten wird, obwohl eine Anwendung für das ebenfalls UNIX-basierte MacOS besteht, stützt die These, dass jede Anwendung individuell entwickelt wurde. Die Browser-Erweiterung ist für Linux-Nutzer also verpflichtend. Bei der Browser-Erweiterung wäre Code-Sharing möglich, leider lässt sich diese Aussage nicht anhand von Quelltext überprüfen. Code-Sharing bei den mobilen Anwendungen wird scheinbar auch nicht praktiziert, da sich die Versionsnummern der beiden Applikationen unterscheiden und auch die Releasezeiten völlig unterschiedlich sind, obwohl auch sicherheitskritische Änderungen vorhanden waren, die mehrere Plattformen betreffen könnten.
Padlock nutzt eine gemeinsame Codebasis für alle Plattformen. Für die Desktop-Plattformen wird *Electron* und für die mobilen Plattformen *Cordova* genutzt. Dadurch, dass beide Frameworks eine WebView bereitstellen, lässt sich sehr viel Code für Logik und Bedienoberfläche teilen. Der Anteil des Code-Sharings ist aber nicht so hoch wie bei KeePass. Ein guter Ansatzpunkt, um die Umsetzung des Code-Sharings

[31] Mono: https://www.mono-project.com/, abgerufen am 09.07.2018

zu begutachten, ist das Dateisystem. Hier zeigt sich, dass plattformspezifischer Code implementiert wurde, da Electron eine andere API und Pakete verwendet, als Cordova. Dennoch pflegen die Entwickler den Code in einer einzelnen Datei, was den falschen Eindruck erweckt, dass sämtlicher Quellcode für alle Plattformen genutzt wird.
Die Applikation Passbolt, die einen anderen Ansatz verfolgt, setzt sich aus einer Browser-Erweiterung und einem Server als Docker-Image zusammen. Die Browser-Erweiterung nutzt in großem Umfang gemeinsamen Code, überschreibt einige Dateien aber mit browserspezifischem Code. Eine vollständig gemeinsame Codebasis nutzt Passbolt hingegen bei der Server-Implementierung. Durch Docker, das eine virtualisierte Umgebung für verschiedene Applikationen bereitstellt, wird eine Abstraktionsschicht hinzugefügt, sodass Passbolt immer mit einem Debian 9 Image betrieben wird. Durch Docker kann diese virtualisierte Umgebung auf vielen Plattformen (Linux, MacOS, Windows) und Architekturen (x86, x64, ARM) genutzt werden. Plattformspezifische Sonderfälle werden auch hier durch Docker gelöst, sodass der Code nur einmal geschrieben werden muss. In Tabelle 3.2 werden die Eigenschaften der Passwortmanager hinsichtlich des Code-Sharings zusammengefasst.

Eigenschaft	**KeePass**	**Padlock**	**Passbolt**
Plattformen	Windows, Mono	Windows, MacOS, Linux, Android, iOS, ChromeOS	Browser (Firefox, Chrome), Windows, MacOS, Linux
Code-Sharing im Bereich	UI, Core, native API	UI, Core	UI, Core, native API
Technologie	C# mit .NET	Web-Technologien (HTML, JS, CSS) mit Polymer in Electron und Cordova	Browser-Erweiterung in JavaScript; Server als Docker-Image in PHP mit CakePHP

Tabelle 3.2: Code-Sharing im Vergleich

Code-Qualität

Um die Qualität des Codes auch bezüglich Sicherheit bewerten zu können, müssen gewisse Metriken zurate gezogen werden. Dazu werden hier die Design Prinzipien von Saltzer und Schroeder zurate gezogen (E. Smith, 2012). Der Informatiker Jerome Saltzer stellte 1974 eine Liste von Sicherheitsmechanismen, die in dem Time-Sharing-Betriebssystem „Multics“ vertreten waren und leitete auf dieser Basis fünf Design-Prinzipien ab. Ein Jahr später wurden die Prinzipien mit Hilfe von Michael Schroeder erweitert. Das Dokument stellt wichtige Regeln auf, die bei der Entwicklung von si-

cheren Softwaresystemen berücksichtigt werden sollten. Die Prinzipien befassen sich dabei sowohl mit Anwendungsdesign und dem Rechtesystem, als auch mit dem offenen Design nach Kerckhoff und der psychologischen Akzeptanz der Anwendung. Im folgenden werden die Regeln aufgeführt, nach denen die Passwortmanager geprüft werden. Eine der Prinzipien „Least common mechanism“ wird in dem Vergleich nicht betrachtet. Das Prinzip befasst sich mit geteilten Ressourcen eines Computers mit mehreren Nutzern, bei denen sich der Aktionen eines Nutzers auch auf den Anderen auswirken können, falls die Bereiche der CPU oder des Arbeitsspeichers nicht wirksam voneinander getrennt sind (E. Smith, 2012, S. 21). Durch die begrenzte Zeit und an Ermangelung an Mitteln zur Überprüfung dieses Design-Prinzips, werden die im Folgenden formulierten Design-Prinzipien betrachtet.

1. **Economy of mechanism:** Einfaches Anwendungsdesign, damit Tests und Validierung einfacher durchgeführt werden können. Überprüft wird dies durch Betrachtung des Quellcodes (Variablen- und Funktionsnamen, Ordner- und Dateistruktur, Dokumentation).

2. **Fail-safe defaults:** Standardmäßig sollen Rechte eingeschränkt sein und nur bei Bedarf gewährt werden. Dies kann dem Quellcode entnommen werden, indem auf Passwortabfragen oder Dateisystemrechte geprüft wird, oder durch Ausprobieren der Anwendung und des Versuchs des Zugriffs als fremder Nutzer.

3. **Complete mediation:** Regelmäßiges Abprüfen der Rechte ist notwendig, damit kein Zugriff auf Daten erfolgen kann, zu dem der Nutzer keine Berechtigung hat. Dies ist vor allem wichtig bei Anwendungen, die mehrere Benutzerkonten und ein Teilen von Daten mit anderen Nutzern anbieten. Dies kann durch den Quellcode überprüft werden oder durch Ausprobieren der Anwendung mit mehreren Nutzern.

4. **Open design:** In Anlehnung an das Kerckhoffsche Prinzip und an Shannons These, dass „der Angreifer das System kennen sollte“ (Shannon, 1949), befanden auch Saltzer und Schroeder die Offenheit eines kryptografischen Systems für äußerst wichtig. Dies lässt sich anhand der Lizenz eines Programms überprüfen.

5. **Separation of privilege:** Mehrere Faktoren sichern den Zugriff auf ein System. Dies kann durch die Suche im Quellcode oder Aufsuchen der Funktion „Mehr-Faktor-Authentifikation“ überprüft werden.

6. **Least privilege:** Dem Nutzer sollen nur so wenig Rechte gewährt werden wie möglich. Geprüft wird dies, indem bei mehrbenutzerfähigen Passwortmanagern

unerlaubte Aktionen durchgeführt werden können, wie ein Zugriff auf nicht mehr geteilte Passwörter.

7. **Psychological acceptability:** Nutzer müssen die Notwendigkeit von Aktionen und Empfehlungen verstehen, damit die Nutzer diese zur eigenen Sicherheit durchführen und befolgen. Überprüft wird dies anhand von Beschreibungstexten und Dokumentationen der Funktionen.

8. **Work factor:** Der Aufwand soll durch Anpassung eines Work Factors veränderbar sein. Dies lässt sich mittels einer Durchsicht des Quellcodes nach Algorithmen wie „bcrypt“, „scrypt“, „PBKDFv2“ oder „Argon2“ überprüfen.

9. **Compromise recording:** Das System sollte Zugriffe und Angriffe protokollieren. Überprüfen lässt sich das durch die Bedienung der Anwendung, ob eine Darstellung der stattgefundenen Zugriffe möglich ist.

Zusätzlich zu bekannten Design Prinzipien existieren für verschiedene Programmiersprachen sogenannte „Coding Guidlines“. Diese Guidelines sollen den Entwickler bei der Verarbeitung von sensiblen Information unterstützen. Ein Beispiel für gut gepflegte Guidlines sind die „Secure Coding Guidelines“ von Oracle[32] für die Sprache Java. Diese lassen sich leider nicht vollständig und nur selten auf andere Programmiersprachen übertragen, weshalb in folgenden Coding Guidelines für die jeweilige Sprache herausgesucht wurden.

Das Projekt KeePass ist wie erwähnt in C# implementiert. Zuerst werden hier die Design Prinzipien von Saltzer und Schroeder zurate gezogen (E. Smith, 2012), da Microsoft in den zugehörigen Coding Guidlines für C# leider nur wenige Themen thematisiert. Code-Beispiele sind ebenfalls selten.
Breiter aufgestellt sind die Design Prinzipien von Saltzer und Schroeder. Die erste Regel „Economy of mechanism“ fordert ein einfaches Anwendungsdesign, sodass Testen und Validieren einfacher durchgeführt werden kann. Die Dokumentation der Funktionen und Variablen wurde nur teilweise durchgeführt, die Struktur ist nicht dokumentiert und nur schwer zu durchschauen. An Stellen, an denen Code-Dokumentation eingepflegt wurde, wurden sinnvolle Funktions- sowie Variablennamen gewählt. Das genaue Gegenteil ist bei den undokumentierten Stellen der Fall. Vor allem dort hätte aber eine Dokumentation stattfinden müssen, um die Wartbarkeit und das Verständnis zu erhöhen. Zwar hat der Entwickler erste Schritte in Richtung einfachem und verständlichem Design der Anwendung unternommen, unterlässt es aber Tests für die

[32]Oracle: http://www.oracle.com/technetwork/java/seccodeguide-139067.html, abgerufen am 17.11.2018

zahlreichen Komponenten der Anwendung zu verfassen. Dies ist sehr verwunderlich, da es sich bei einem Passwortmanager um eine sensible Anwendung handelt, bei der Softwarefehler fatale Auswirkungen bis hin zum Verlust aller Daten haben können. Die weiteren Prinzipien sind „Fail-safe defaults“ und „Complete mediation“. Die Prinzipien zielen auf einen sinnvollen Standard von Zugriffsrechten und das regelmäßige Prüfen jener Rechte ab. Standardmäßig hat kein Nutzer Zugriff auf die verschlüsselten Daten und nur Personen, die das Master-Passwort kennen, können die verschlüsselten Daten entschlüsseln. Somit gilt die erste Regel der zuvor Genannten als erfüllt. Regelmäßiges Abfragen der Rechte zum Lesen der Datei im Sinne des Prinzips „Complete mediation“ ist aber nicht notwendig, da ein Abbild der Datenbank nach einmalig Zugriff in den Arbeitsspeicher geladen wird. Die Rechte zum Zugriff auf die Datenbankdatei verwaltet KeePass selber, wenn beispielsweise eine Änderung in die Datenbank geschrieben werden soll. Das Prinzip „Open Design“ folgt dem Kerckhoffschen Prinzip, dass kryptografische Anwendungen und Algorithmen öffentlich zugänglich sein sollten. Auch dieses Prinzip erfüllt KeePass. Die nächsten beiden Regeln „Separation of privilege“ und „Least privilege“ werden ebenfalls erfüllt. Diese beschreiben zum einen die Mehr-Faktor-Authentifizierung und die Empfehlung, dem Nutzer nicht sofort alle Rechte einzuräumen. Da es aber keine Unterscheidung zwischen einem Standard-Nutzer und einem höher privilegierten Nutzer in KeePass gibt, ist dieser Punkt hinfällig. Als weitere Regel definiert „Psychological acceptability“, dass Nutzer den Sinn der sicherheitsrelevanten Vorgaben verstehen müssen, da sie sonst nicht korrekt umgesetzt bzw. genutzt werden. Nutzer, die sich für einen Passwortmanager entscheiden, sind sich in der Regel bewusst, dass eine hohe Passwortsicherheit gefordert wird, da ansonsten der Zweck des Passwortmanagers hinfällig wäre. Die Passwortstärke wird zudem auch bewertet, sodass der Nutzer auch hier eine Hilfestellung erhält. In diesem Sinne erfüllt KeePass auch dieses Design Prinzip. Die Design Prinzipien wurden aber noch um zwei weitere Punkte erweitert, die heute relevanter denn je sind. Das erste der beiden Prinzipien ist der „Work factor“, der den gesamten Aufwand zum Errechnen des korrekten Passwortes beschreibt. Der Workload lässt sich sowohl bei der AES-basierten Schlüsselableitungsfunktion als auch bei Argon2 einstellen. Das Prinzip „Compromise recording“ fordert das Aufnehmen von Ereignissen und Zugriffen in einem Log, sodass Zugriffe verfolgbar sind. Diese Funktion ist in KeePass nicht enthalten, zudem existiert auch kein Plugin, welches diese Funktion nachrüstet.

Die Richtlinien von Microsoft beziehen sich auf das korrekte Behandeln von Nutzereingaben, Vermeidung von Race Conditions sowie das dynamische Generieren und Ausführen des Codes. Eingaben werden „escaped“ und verschlüsselt in Objekten abgelegt, sodass keine Passwörter als Klartext im Arbeitsspeicher auffindbar sind. Race

Conditions für Dateizugriffe werden zudem vermieden, indem die Datenbank in ein C#-Objekt überführt wird.

Der Konkurrent Padlock ist in JavaScript für alle Plattformen implementiert. Es werden zuerst die Design Prinzipien von Saltzer und Schroeder überprüft. Auch für Node.js und JavaScript gibt es keine offiziellen „Secure Coding Guidelines“, weshalb stattdessen die Mozilla Secure Coding Guidelines[33] betrachtet werden.
Die Code-Struktur erscheint sinnvoll und enthält wenige langen Dateien, was die Übersicht in beider Hinsicht verbessert und auch durch die eingeschränkte Funktionalität im Vergleich zu KeePass begründet ist. Zudem werden sinnvolle Variablen- und Funktionsnamen genutzt, die Dokumentation ist aber leider lückenhaft. Tests wurden implementiert und stellen die Softwarequalität sicher, welche in Kombination mit der einfachen Struktur das erste Prinzip der Design Principles erfüllen. Ebenso wie KeePass werden hier „Fail-safe defaults“ und „Complete mediation“ implementiert. Der Nutzer hat ohne das korrekte Master-Passwort keinen Zugriff auf die Daten. Desweiteren wird in Padlock ebenfalls der gesamte Datensatz im Arbeitsspeicher gehalten, sodass nur ein einmaliger Zugriff beim Öffnen der Datei erfolgt. Ein Ändern der Rechte durch die Anwendung ist nicht vorgesehen, da diese Applikation nicht mehrere Nutzer unterstützt und folglich keine Rechte verwaltet werden müssen. „Open design“ erfüllt auch Padlock, denn der Code ist frei einsehbar und nach GPLv3 lizenziert. Padlock bietet keine Mehr-Faktor-Authentifizierung, wodurch das Prinzip „Separation of privilege“ nicht erfüllt wird. Dadurch, dass es nur eine Rechtestufe gibt und der Manager nur für den Einzelnutzer-Betrieb ausgelegt ist, müssen keine Rechteebenen verwaltet werden. Lese- und Schreibrechte werden durch das Betriebssystem verwaltet. Stimmen diese nicht, kann der Passwortmanager die Daten nicht lesen und gibt eine Fehlermeldung aus. Das Prinzip „Least privilege“ wird damit erfüllt. Einen Beitrag zur „Psychological acceptability“ bietet der Manager hier nicht. Erklärungen warum und wie genau einige Schritte ausgeführt werden müssen, sucht man vergebens. Der aber für kryptografische Verfahren wichtige „Work factor“ ist hier implementiert, sodass sich der Aufwand zum Berechnen eines Hashes durch den Entwickler mit fortschreitender Entwicklung in der Computer-Industrie regulieren lässt. Leider können keine Zugriffe protokolliert werden, was das Prinzip „Compromise recording“ jedoch empfiehlt.
Die von Mozilla gepflegten Guidelines richten sich zuerst an Web-Entwickler, die kritische Anwendungen für den Browser entwickeln. Da Padlock zwar mit Web-Technologien arbeitet, aber keine Anwendung für den Browser ist, müssen die Guidelines im Vorfeld gefiltert werden. Im folgenden werden nun die Guidelines betrachtet, die für Padlock relevant sind. Mozilla gibt hier Empfehlungen zur Wahl des Pass-

[33] Mozilla Wiki: https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines, abgerufen am 10.7.2018

worts. Dieses soll mindestens acht Zeichen und Ziffern enthalten. Zudem sollten die Passwörter mit einer Blacklist verglichen werden, sodass nicht schwache Wörter gewählt werden können. Ein Abgleich mit einer Blacklist findet bei Padlock nicht statt. Die Inspektion des Quellcodes ergab zudem, dass zwar das Passwort auf Stärke geprüft wird, es aber dennoch möglich ist ein schwaches Passwort zu nutzen. Dies sollte aber unter keinen Umständen möglich sein. Ein weiterer entscheidender Punkt ist der Umgang mit abgelehnten Login-Daten. Gescheiterte Login-Versuche sollen eine generische Fehlermeldung anzeigen, die keine Rückschlüsse zulässt, welche Eingabe von Nutzername und Passwort genau fehlerhaft ist. Auch die Passwörter der gescheiterten Passwörter sollten analysiert werden, sodass es sich bei grundlegend verschiedenen Plattformen um einen Angriffsversuch handeln könnte. Bei schnell aufeinander folgenden Zugriffsversuchen kann eine Wartezeit zwischen den Passworteingaben eingeführt werden. Padlock zeigt bei einem falschen Passwort gar keinen Hilfetext an, sondern lässt, im Stile von Apples MacOS, ein Symbol wackeln, um zu zeigen, dass der Anmeldevorgang nicht erfolgreich war. Nach mehreren erfolglosen Versuchen wird ein Popup angezeigt, ob die Datenbank zurückgesetzt werden soll oder weiter versucht werden soll. Zwar ist der erste Teil des Kriteriums im Umgang mit Passwörtern erfüllt, jedoch findet keine Untersuchung der Passwörter, geschweige denn eine Protokollierung der Zugriffsversuche, statt. Der letzte wichtige Punkt der Mozilla Guidelines, der hier angewandt werden kann, ist der der Eingabenvalidierung. Diese findet hier nicht statt, da hier kein Potential für eine SQL-Injection besteht. Das Speicherformat entspricht der JavaScript-Object-Notation, sodass die Daten einfach geparst und in den Arbeitsspeicher geladen werden können.

Die dritte Anwendung im Vergleich ist Passbolt. Die Applikation besteht aus einem Paket für den Server, das in der Community Edition kostenfrei selbst gehostet werden kann. Die Server-Anwendung ist in PHP implementiert, die Browser-Anwendung in JavaScript. Auch hier werden die Design Prinzipien nach Saltzer und Schroeder herangezogen und anschließend noch die „PHP Sicherheitsinformationen“ betrachtet.
Wie bereits bekannt, zielt das erste Prinzip „Economy of mechanism“ auf einen einfach strukturierten und damit auch einfach testbaren Code ab. Sowohl Browser-Erweiterung als auch Server-Anwendung sind hervorragend dokumentiert und strukturiert. Für die Server-Applikation, die hier das wichtigste Element aufgrund der Datenhaltung darstellt, sind zahlreiche PHP-Tests vorhanden. Damit wird dem Prinzip vollends Folge geleistet. Die beiden weiteren Regeln zielen auf die Verwaltung von Rechten ab. Wie bei den anderen beiden Open-Source-Passwortmanagern ist der Zugang standardmäßig verwehrt und wird erst nach Eingabe des korrekten Master-Passworts gewährt. Dadurch, dass der Manager von mehreren Benutzern verwendet

werden kann, können hier auch Rechte entzogen werden. Standardmäßig können Nutzer, falls Passwörter freigegeben werden, diese nur lesen und nicht bearbeiten. Auf ungeteilte Passwörter haben die anderen Nutzer keinen Zugriff, was dem Prinzip „Fail-safe defaults“ von Saltzer und Schroeder entspricht. Dadurch, dass der Nutzer bei Passbolt nie auf alte Datenquellen zugreift und durch einen sogenannten ein Event-System immer von Änderungen an den Daten benachrichtigt wird, wird dem Nutzer im Sinne des Prinzips „Complete mediation“ sofort das Recht entzogen auf einen nicht mehr geteilten Datensatz zuzugreifen. Die Anfangs niedrig gesetzten Rechte erfüllen auch das Prinzip „Least privilege“. Passbolt ist Open-Source-Software mit AGPLv3-Lizenz, was auch im Sinne des Prinzips „Open design“ ist. Passbolt nutzt das sichere Verfahren OpenPGP (IETF RFC 4880[34]) und dessen Implementierung GnuPG. Die generierten Schlüssel in OpenPGP lassen sich mit einem Passwort schützen, sodass eine Zwei-Faktor-Authentifizierung möglich ist (PGP Corporation, 2003, S. 32). Eine Entschlüsselung kann nur mit dem privaten Schlüssel und dem Passwort für den Schlüssel erfolgen. Der Nutzer wird bei der Wahl des Master-Passworts beraten und somit auch für die Notwendigkeit eines starken Passwortes sensibilisiert. Auch hier erfüllt Passbolt das Prinzip „Psychological acceptability“. Durch die Nutzung von OpenPGP kann kein Wortfaktor eingestellt werden und Log-Files werden nur für HTTP-Anfragen geschrieben, jedoch keine, die genauer auf fehlgeschlagene Login-Versuche hinweisen. Damit erfüllt Passbolt die letzten beiden Prinzipien leider nicht. Nun werden die „PHP Sicherheitsinformationen“ betrachtet[35]. Auch hier muss nach Einsatzzweck gefiltert werden, sodass im Folgenden nur für einen Passwortmanager relevante Regeln näher analysiert werden. Eine Regel, die in allen Guidelines empfohlen wird, ist die Nutzereingaben vor der Verarbeitung zu bereinigen, sodass keine gefährlichen Eingaben bearbeitet werden. Dies wird von Passbolt bei jeder Eingabe, sei es ein Passwort oder eine geänderte Einstellung, hervorragend umgesetzt, sodass auch SQL-Injections unterstützt durch „Prepared statements“ effektiv vermieden werden. Natürlich wird in den Guidelines noch die Notwendigkeit der kryptografischen Verarbeitung von Passwörtern oder sensitiven Daten zum Speichern in Datenbanken erwähnt. Auch dies setzt Passbolt hervorragend um, indem OpenPGP für die Verschlüsselung genutzt wird. Passwörter werden für die Nutzer nicht in der MySQL-Datenbank abgelegt, denn das Passwort wird zur Nutzung des privaten Schlüssels direkt in OpenPGP genutzt. Leider sind die PHP Sicherheits-Informationen sehr veraltet, sodass weitere Empfehlungen über gar nicht mehr vorhandene Funktionen dokumentiert sind.

Abschließend lässt sich sagen, dass die neuen Entwicklungen besser strukturiert und dokumentiert sind. Zudem sind Tests vorhanden. KeePass merkt man auch im

[34]IETF: https://tools.ietf.org/html/rfc4880, abgerufen am 02.12.2018
[35]PHP: https://secure.php.net/manual/de/security.php, abgerufen am 26.07.2018

Hinblick auf den Quellcode das Alter an. Die Entwickler haben bei der Codepflege und der Dokumentation noch einiges nachzuholen. Vor allem sind Tests nötig, damit Änderungen sich nicht negativ auf die Operabilität der Anwendung auswirken. Der Vergleich der Anwendungen wie diese die Design Prinzipien von Saltzer und Schroeder erfüllen, wird in Tabelle 3.3 aufbereitet.

Tabelle 3.3: Code-Qualität im Vergleich

Design Prinzip	**KeePass**	**Padlock**	**Passbolt**
Economy of mechanism	Struktur etwas undurchsichtig, keine Tests, teilweise dokumentiert	Tests implementiert, Dokumentation teilweise umgesetzt	•
Fail-safe defaults	•	•	•
Complete mediation	•	•	•
Open design	•	•	•
Separation of privilege	•	–	•
Least privilege	•	•	•
Psychological acceptability	•	–	•
Work factor	•	•	–
Compromise recording	–	–	–

• Kriterium vollständig erfüllt
– Kriterium nicht erfüllt

Kryptografische Verfahren

Die kryptografischen Verfahren sind die Kernkompetenzen eines Passwortmanagers und stellen die Sicherheit der Authentifikationsdaten eines Nutzers sicher. Bei vier der fünf Passwortmanager ist AES mit 256 Bit Schlüssellänge (NIST FIPS 197) der Verschlüsselungsalgorithmus der Wahl. LastPass nutzt laut eigener Aussage AES in Kombination mit PBKDF2 (NIST SP 800-132) SHA256 und Salted Hashes. Wofür die Algorithmen genutzt und wie die Authentifikationsdaten abgelegt werden, ist nicht bekannt. Höchstwahrscheinlich wird mit PBKDF2 ein Schlüssel für AES abgeleitet. SHA256 nimmt dabei die Funktion der pseudozufälligen Funktion ein.
Ebenfalls bedeckt hält sich 1Password. Es wird AES 256 Bit zur Verschlüsselung genutzt. Zur Ableitung des Schlüssels wird laut Sicherheitsdatenblatt[36] PBKDF2-HMAC-SHA256 genutzt. Zusätzlich zum Master Passwort wird auch noch ein „Secret

[36] 1Password: https://1password.com/files/1Password%20for%20Teams%20White%20Paper.pdf, abgerufen am 18.07.2018

Key“ generiert, der als zusätzlicher Faktor in der Verschlüsselung genutzt wird und immer beim Nutzer verbleibt. Dies bedeutet auch, dass der Zugang zu den Daten im Falle eines Verlustes verwehrt bleibt. Dieser Secret Key ist 34 Buchstaben und Ziffern lang, dessen Blöcke mit Bindestrichen separiert werden. 1Password garantiert für den Secret Key eine Entropie von 128 Bit, die in Kombination mit einem starken Master-Passwort noch höher ausfällt. Bearbeitet wird dieser Schlüssel mit HMAC-based Extract-and-Expand Key Derivation Function (HKDF)[37] nach RFC 5869, bevor dieser auf dem Gerät abgelegt wird. Der Schlüssel grenzt sich aber von der Zwei-Faktor-Authentifizierung ab, da der Secret Key im Vergleich zu One-Time-Passwörtern immer gleich ist und aktiv zur Verschlüsselung genutzt wird, wobei die zwingende Verfügbarkeit die gleiche Wirkung wie der zweite Faktor auf die Sicherheit erzielt. Zusätzlich zu den Faktoren Passwort und Secret Key, wurde eine Zwei-Faktor-Authentifizierung implementiert, sodass eine Anmeldung mit Passwort auch einen PIN-Code erfordert. Als einziger hier vorgestellter Passwortmanager, bietet 1Password eine Drei-Faktor-Authentifizierung.

KeePass nutzt zur Verschlüsselung der Datenbankdatei, wie die anderen Manager, AES256 und ChaCha20 (RFC 7539). Zur Ableitung des Schlüssels wird das sichere Argon2 (IETF draft-irtf-cfrg-argon2-03) genutzt, das 2015 die „Password Hashing Competition“ gewonnen hat. Dieses zeichnet sich nicht nur durch den vorhandenen Kostenfaktor aus, sondern auch durch die weiteren Parameter Arbeitsspeicher, Parallelismus, Zeit und Länge des Salts. Diese weiteren Parameter decken somit auch die zu erwartende Weiterentwicklung der Geräte hinsichtlich Speichergröße und Rechenzeit mit mehreren Prozessorkernen ab, indem auch dafür konfigurierbare Parameter vorhanden sind. Weitere Verfahren können über Erweiterungen nachgerüstet werden. Um die Datenintegrität sicherzustellen, wird HMAC-SHA256 genutzt. Dies konnte der Dokumentation entnommen und musste deshalb nicht im Quellcode nachgeschlagen werden. Dem Quellcode kann aber entnommen werden, dass sowohl ChaCha20 als auch Argon2 selbst implementiert wurden. Diese Entscheidung verwirrt, denn eine gut gewartete Referenz-Implementierung von Argon2 liegt bereits in der Sprache C vor, die in C# genutzt werden kann. Für HMAC-SHA256 und AES256 nutzt der Entwickler die Kryptografie-Komponente von .NET.

Anders sieht es bei Padlock aus. Offiziell wird angegeben, dass AES256 verwendet wird. Glücklicherweise liegt der Quelltext vor und dieser kann auf die verwendeten Algorithmen hin untersucht werden. Laut Quelltext sind neben 256 auch 128 und 192 Bit als Schlüssellängen möglich, die aber nicht mehr den aktuellen Sicherheitsstandards entsprechen. Dem Quelltext kann auch entnommen werden, dass PBKDF2 mit SHA256 genutzt wird, um den Schlüssel für AES zu generieren. Für alle kryptografi-

[37] IETF: https://tools.ietf.org/html/rfc5869, abgerufen am 18.07.2018

schen Funktionen nutzt Padlock auf der Desktop-Plattform die node.js Crypto-API, auf der mobilen Plattform hingegen wird „sjcl“, die „Stanford JavaScript Crypto Library“, in der Version `1.0.7` angesprochen. Am 10.11.2018 wurde eine neue Version veröffentlicht, in der einige Schwachstellen und Fehler behoben wurden. Eine neue Version von Padlock, die die aktuellen Versionen der genutzten Pakete beinhaltet, wurde bis Dato (Stand 18.11.2018) nicht veröffentlicht. Im Gegensatz zu KeePass wird bei Padlock keine kryptografische Funktion selbst implementiert.
Einen interessanten Ansatz vertritt Passbolt, bei dem OpenPGP für die Verschlüsselung verwendet. Mit dem privaten Schlüssel, den nur der legitime Nutzer besitzt, können die Daten ver- und entschlüsselt werden. Der private Schlüssel kann zusätzlich mit einem Passwort gesichert werden, sodass auch der Verlust nicht zu Missbrauch führt. Ähnliches Schema wollte 1Password vermutlich mit dem Secret Key emulieren, sodass es noch einen weiteren Faktor gibt, mit dem der Zugriff erst möglich ist. OpenPGP wird hier nicht selbst implementiert, sondern die Entwickler nutzen dafür das Paket „openpgp-php“ in der aktuellen Version `0.3.0`, die im April 2017 veröffentlicht wurde. Tabelle 3.4 zeigt wichtige Eigenschaften der Manager hinsichtlich der kryptografischen Verfahren in einer übersichtlichen Form auf.

Funktionsvielfalt

LastPass und 1Password sind im Funktionsumfang, ausgeschlossen sind verfügbare Apps, sehr ähnlich. Beide bieten Zwei-Faktor-Authentifizierung sowie 1 GB Dokumentenspeicher, zusätzlich zu den in der Anzahl unlimitierten Authentifikationsdaten. Natürlich ist in beiden Anwendungen auch ein Passwort-Generator und eine Passwortbewertung bereits existenter Passwörter integriert. Beide bieten erweiterten Support für ihre Anwendungen. LastPass bietet die Möglichkeit Passwörter für Dritte freizugeben, sowie eine Notfallzugriffsfunktion. 1Password bietet die Möglichkeit die Passwörter für einen gewissen Zeitraum vom mobilen Gerät zu entfernen. Dies ist zum Beispiel bei Reisen sinnvoll, sodass im Falle des Verlustes keine Passwörter auf dem Gerät vorhanden sind. Beide liefern gute Hilfestellungen zur Inbetriebnahme und Erläuterungen für Funktionen.
KeePass begeistert mit einer hervorragenden Dokumentation der vorhandenen Funktionen. Diese ist nicht nur im Programm selber aufrufbar, sondern kann auch online im Browser eingesehen werden. Die Dokumentation ist auch erforderlich, denn KeePass ist gefüllt mit zahlreichen Funktionen, sei es direkt integrierte Funktionen oder welche, die über Erweiterungen eingepflegt werden können. KeePass kann ebenfalls Anmeldedaten in Programme einfügen. Durch Nutzung der Zwischenablage ist ein Schutz vor Keylogger vorhanden, da diese nur die Tastenkombination für Kopieren und Einfügen ermitteln können. Da KeePass lokal installiert wird, ist der Dokumen-

Merkmal	**LastPass**	**1Password**	**KeePass**	**Padlock**	**Passbolt**
Verschlüsselung Nutzerdaten	AES 256 Bit	AES 256 Bit	AES 256 Bit	AES 256 Bit	OpenPGP
Schlüsselableitung	PBKDFv2	PBKDF2-HMAC-SHA256	Argon2	PBKDFv2 mit SHA-256	keine
Datenintegrität	Unbekannt	Unbekannt	HMAC-SHA-256	–	–
Crypto-Libs und Version	Unbekannt	Unbekannt	`.NET` Crypto; `ChaCha` eigene Implementierung; `Argon2` eigene Implementierung	`sjcl` in `v1.0.7`; `node.js` Crypto	`openpgp-php` in `v0.3.0`
Besonderheiten	–	Secret Key für erhöhte Entropie und als weiterer Faktor	Weitere Verschlüsselungs- und Hash-Algorithmen können über Plugins nachgerüstet werden	–	Authentifizierung über GPGAuth

Tabelle 3.4: Kryptografische Verfahren in Passwortmanagern im Vergleich

tenspeicher nur durch die Speicherkapazität des Gerätes oder des Remote-Hosts für beispielsweise WebDAV beschränkt. Support erhält der Nutzer in der Community, eine Hotline gibt es nicht. Erweiterungen können verschiedene Verschlüsselungsmechanismen, Adapter für Synchronisierung, oder auch Unterstützung für verschiedene Dateiformate für Im- und Export sein.
Padlock bewirbt sich selbst als minimalistischen Passwortmanager, der er auch tatsächlich ist. Die Bedienung ist einfach, da auch nicht viele Funktionen vorhanden sind. Es können Daten gespeichert und kategorisiert werden. Eine Synchronisierung mit dem Online-Dienst von Padlock ist ebenfalls möglich. Die Anwendung sperrt sich nach längerer Inaktivität. Eine Zwei-Faktor-Authentifizierung ist nicht möglich, ebenso sind keine Plugins installierbar. Falls dennoch Fragen zur Benutzung aufkommen, gibt es ein wenig Hilfe auf der Webseite, die aber nicht die rudimentären Funktionen

Merkmal	**LastPass**	**1Password**	**KeePass**	**Padlock**	**Passbolt**
Passwort-Generator	•	•	•	•	•
Mehr-Faktor-Authentifizierung	•	•	•	–	Passwort und privater Schlüssel
Multi-User	•	•	•	–	•
Erweiterungen	–	–	•	–	–
Datenkapazität	1GB	1GB	Begrenzt durch Datenspeicher des Geräts	Begrenzt durch Datenspeicher des Geräts	Begrenzt durch Datenspeicher des Servers
Importformate	CSV, zahlreiche Datenbanken anderer Manager	1Password-Format	CSV, XML, zahlreiche Datenbanken anderer Manager	CSV, LastPass, SecuStore	–
Exportformate	CSV	1Password-Format (.1pif), CSV	CSV	CSV, proprietäres Format	–
Synchronisierung zu zugehörigem Cloud-Service	•	•	selbstgehostet über SFTP, FTPS, WebDAV, SMB (Dropbox, Google Drive und mehr über Plugins)	•	•

• Kriterium vollständig erfüllt
– Kriterium nicht erfüllt

Tabelle 3.5: Funktionsvielfalt im Vergleich

erläutert.
Der Passwortmanager Passbolt kann entweder über ein Installationspaket oder über einen Docker-Container in Betrieb genommen werden. Installationsskripte oder ein fertiges Paket, das auch alle Abhängigkeiten spezifiziert und installiert, ist nur den kostenpflichtigen Abonnements vorbehalten. Dazu gehören auch Telefon- und Email-Support. Der Support der kostenlosen Community-Edition beschränkt sich auf den Support durch die Community. Grundfunktionen, wie das Teilen von Passwörtern, Favoriten, Filtern und Nutzerverwaltung, sind Bestandteil aller Versionen. In der kostenpflichtigen Lizenz erhält der Nutzer Zugriff auf Import, Export, Kategorisierung sowie LDAP-Integration. Die überschaubaren Funktionen der Community-Version sind ausreichend erklärt. Löblich muss man hervorheben, dass auch die technische Dokumentation der Authentifizierung mittels OpenPGP detailliert beschrieben wird. Eine Schnittstellendokumentation der REST-API befindet sich in Planung. Plugins sind auch hier nicht zur Installation vorgesehen. Für einen besseren Überblick können die Ergebnisse des Vergleiches der Tabelle 3.5 entnommen werden.

Lizenz

Nur drei der fünf vorgestellten Anwendungen stehen unter einer Open Source Lizenz. LastPass und 1Password stehen unter einer proprietären Lizenz und somit bleibt ein Blick auf den Quelltext der Anwendung verwehrt. Die drei Übrigen, KeePass, Padlock und Passbolt, stehen unter einer Lizenz der GPL-Familie. KeePass steht unter der älteren GNU GPLv2, Padlock unter der verbreiteten GNU GPLv3 und Passbolt unter der GNU AGPLv3. Vor allem bei kryptografischen Anwendungen ist es laut Kerckhoffschem Prinzip wichtig, dass der Quellcode der Anwendungen für die Öffentlichkeit zugänglich ist, sodass Schwachstellen früh geschlossen werden können. Weiterer Vorteil des offenen Quellcodes ist es, dass sichergestellt werden kann, dass die Sicherheit wirklich durch adäquate kryptografische Verfahren gesichert wird und nicht durch mangelnde Informationen des Angreifers, wie es bei proprietären Anwendungen der Fall ist. Tabelle 3.6 fasst die Ergebnisse zusammen.

Eigenschaft	**LastPass**	**1Password**	**KeePass**	**Padlock**	**Passbolt**
Lizenz	Proprietär	Proprietär	GPLv2	GPLv3	AGPLv3

Tabelle 3.6: Passwortmanager-Lizenzen im Vergleich

Preis

Dass der Quellcode des Passwortmanagers öffentlich verfügbar ist, bedeutet nicht zwangsläufig, dass der Betrieb kostenlos ist. Bei einigen, in dieser Thesis aber nicht

vorgestellten, Applikationen ist nur die Server-Anwendung unter Open Source Lizenz, die Applikation für die Endgeräte muss aber kostenpflichtig erworben werden. Dies ist aber bei keinem der hier thematisierten Passwortmanagern der Fall, denn entweder ist der Betrieb komplett kostenfrei oder komplett kostenpflichtig.
Wie bereits erwähnt, kostet die Einzelpersonlizenz von *LastPass* 2$, für *1Password* sind monatlich 2,99$ zu entrichten. Natürlich müssen noch die diversen Funktionen verglichen werden, die für den Preis angeboten werden, was im Abschnitt „Funktionsvielfalt“ näher betrachtet wird. die Tabelle 3.7 fasst die Preise der Passwortmanager zusammen.

Eigenschaft	**LastPass**	**1Password**	**KeePass**	**Padlock**	**Passbolt**
Preis	2$/Monat	2,99$/Monat	kostenlos	kostenlos	kostenlos

Tabelle 3.7: Preise im Vergleich

3.2.8 Ergebnisse

Der Vergleich beleuchtete viele Eigenschaften eines guten Passwortmanagers. Das Code-Review der quelloffenen Manager lieferte einen interessanten Einblick in die Funktionsweise und Designentscheidungen, die jeweils getroffen wurden. Im Folgenden werden die gewonnenen Erkenntnisse für den eigenen Passwortmanager konkretisiert. Es sollen Kriterien formuliert werden, welche der eigene Passwortmanager zwingend erfüllen soll. Dieses Kapitel strebt keine Wahl bestimmter Algorithmen oder Technologien an. Die begründete Auswahl der Pakete wird in Kapitel 4, nach einer Vorstellung der jeweiligen Kandidaten, getroffen.

Kryptografie und Sicherheit

Überaus interessant ist die Entscheidung von Passbolt OpenPGP für die Verschlüsselung einzusetzen. OpenPGP ist nicht nur eine State-of-the-Art-Technologie, sondern implementiert mit einem passwortgeschützten privaten Schlüssel eine wirksame Zwei-Faktor-Authentifizierung (Lucas, 2006, S. 75). Padlock wählte im Vergleich dazu einen anderen Ansatz. Die Datei, die die gespeicherten Passwörter enthält, beinhaltet diese als JSON-String, bei denen sensible Attribute mittels AES verschlüsselt wurden. Dadurch, dass JSON-Objekte, durch die technische Basis von Padlock, einfach gelesen werden können, hat der Entwickler mit dieser Variante die Kompatibilität zwischen den verschiedenen Plattformen sichergestellt. *Infolgedessen soll eine Technologie bzw. Algorithmus gefunden werden, die nach heutigen Standards State-of-the-Art ist und eine Kompatibilität zwischen allen bedienten Plattformen zuverlässig erlaubt.*

Zur Schlüsselableitung werden von den Mitbewerbern verschiedene Algorithmen verwendet, häufig PBKDFv2 und von KeePass Argon2. Nur bei KeePass konnte durch Einsicht in den Quellcode herausgefunden werden, dass Argon2 selber implementiert wurde. Dies birgt Risiken, denn es besteht die Möglichkeit, dass der Entwickler sich nicht an die dokumentierte Referenzimplementierung gehalten hat und somit eine nicht entdeckte Schwachstelle entstanden sein kann. Zudem muss, falls eine Schwachstelle im Design der Referenzimplementierung bekannt wird, ebenfalls die Implementierung in KeePass evaluiert werden. Dies sorgt für hohe Wartezeiten, falls diese Schwachstellen überhaupt behoben werden. *Deswegen soll bei dem eigenen Passwortmanager auf die eigene Implementierung sicherheitskritischer Algorithmen verzichtet werden und stattdessen gut gewartete und dokumentierte Bibliotheken genutzt werden.*

Bei der Wahl des Verfahrens zur Schlüsselableitung ist es wichtig, dass ein *Kostenfaktor eingestellt werden kann*, wie bei PBKDFv2, bcrypt, scrypt und Argon2, sodass den Design Prinzipien von Saltzer und Schroeder entsprochen wird. Zudem sollen die *Parameter für den gewählten Algorithmus den heutigen Standards entsprechen.*

Im Sicherheitsaudit von Padlock wurden auch Probleme beleuchtet, die bei Padlock nur auf Android zum Tragen kommen. Beispielsweise wurden Passwörter nicht unkenntlich gemacht, oder der Manager sperrt den Zugang nicht automatisch nach einer gewissen Zeit. *Diese Punkte soll der in dieser Thesis zu implementierende Manager für mobile Geräte und, nach Möglichkeit, für Desktop-Betriebssysteme ebenfalls erfüllen, um die Verwundbarkeit durch Spähangriffe zu minimieren.*

Code-Sharing

Eine weitere Erkenntnis ist, dass kaum einer der Mitbewerber auf eine gemeinsame Codebasis setzt. Viel mehr werden Apps mit nativen Sprachen bevorzugt. Erwähnenswert ist zudem, dass einige Dienste auch gar keinen Client als Anwendung anbieten, sondern sich direkt in den Browser integrieren. Browser-Erweiterungen können Code-Sharing bieten, da durch die gemeinsame Programmiersprache JavaScript der Core-Code geteilt werden kann.
Vorbild für Code-Sharing ist Padlock, denn mit Hinblick auf Code-Sharing wird nicht nur der Core-Code geteilt, sondern auch die Komponenten der Nutzeroberfläche, was nur durch die Kombination von Electron mit Cordova ermöglicht wird. Da die eigene Implementierung aufgrund von Bediengeschwindigkeit und User Experience von Cordova absieht und auf React Native setzt, kann nur Code-Sharing des Core-Codes betrieben werden. Dies bedeutet, dass möglichst viele *Event-Handler, die sich nicht direkt auf Bedienoberfläche beziehen, zwischen den Plattformen geteilt werden sollen*, sodass Views möglichst wenig eigene Logik enthalten. Die Entwickler von Padlock nutzen für das Dateisystem beispielsweise spezifischen Code für Desktop oder mobile

Endgeräte. Dadurch ergeben sich für den anderen Code unterschiedliche Schnittstellen, die im Code an zahlreichen Stellen über `if/else`-Verzweigungen unterschieden werden. Dies erhöht unnötig die Komplexität des Codes und verschlechtert die Wartbarkeit, da plattformspezifischer Code nicht mehr vom gemeinsamen Code getrennt wird. *Aus diesem Grund soll in dem Passwortmanager dieser Thesis immer eine Schnittstelle in der Anwendung für plattformspezifischen Code geben.* Dies bietet den Vorteil, dass der spezifische Code in einem eigenen Bereich abgekapselt ist und die Rückgabetypen sowie Methodensignaturen identisch für alle Plattformen sind.
Code-Sharing bietet aber nicht nur Vorteile bei der Strukturierung und Wartbarkeit des Codes, sondern Schwachstellen lassen sich auch schneller für alle Plattformen beheben. Erneut dient Padlock hier als positives Beispiel. Durch das Sicherheitsaudit von Cure53 wurden Schwachstellen nachgewiesen, wie beispielsweise, dass Login-Daten für die Cloud unverschlüsselt auf dem Gerät abgespeichert wurden (Heiderich u. a., 2016, S. 18), oder ein unsicherer Zufallsgenerator für zufällige Passwörter genutzt wurde (Heiderich u. a., 2016, S. 3). Da der Entwickler detailliert Aufschluss darüber gibt, welche Änderung welche Schwachstelle behebt, kann über GitHub leicht nachvollzogen werden, ob die Schwachstelle in dem geteilten Code vorhanden ist. Die beiden zuvor genannten Schwachstellen hat der Entwickler mit einigen wenigen Änderungen im Core-Code beheben können[38][39]. Dies zeigt, dass die Sicherheit einer Anwendung enorm von Code-Sharing profitiert, da wichtige Änderungen direkt in beiden Plattformen implementiert werden können. Des Weiteren bietet das Verwenden der gleichen Bibliotheken auf beiden Plattformen entscheidende Vorteile. In Padlock wurde der Zufallsgenerator nur zur Erzeugung von zufälligen Passwörtern genutzt, aber nicht zum Erzeugen des notwendigen Zufalls für die Verschlüsselung von Daten mittels AES. Denn für die Verschlüsselung wurde sowohl auf dem Desktop als auch bei der mobilen Anwendung die Bibliothek „sjcl“ verwendet, die sich bereits eines sicheren Zufallsgenerators bedient. Die Sicherheit der Passwörter war schlussendlich nicht von der Wahl des unsicheren Zufallsgenerators betroffen.

Funktionsumfang

Vor allem die kommerziellen Passwortmanager erfreuen sich großer Beliebtheit. Ein Grund dafür könnte sein, dass im Falle von LastPass der Aufwand für den Kunden zur Inbetriebnahme auf allen Geräten sehr gering ist. Im Falle von KeePass ist man hinsichtlich der Funktionen zwar hervorragend aufgestellt, bei den durch die Community zur Verfügung gestellten Implementierungen für die verschiedenen Plattformen

[38] GitHub: https://github.com/MaKleSoft/padlock/commit/4b01a36ad90c, abgerufen am 17.11.2018
[39] GitHub: https://github.com/MaKleSoft/padlock/commit/22bffa5232b0, abgerufen am 17.11.2018

fehlt es aber an Transparenz. Dadurch und durch den Umstand, dass einige Funktionen über Plugins nachgerüstet werden müssen, sind viele Nutzer abgeschreckt.
Um den Aufwand zur Inbetriebnahme zu verringern soll in dieser Thesis die Benutzeroberfläche der Anwendung auf allen Plattformen identisch sein. Da eine Crossplatform-Anwendung implementiert wird, ist der Funktionsumfang ebenfalls exakt gleich, was für den Nutzer keine Umstellung von Plattform zu Plattform bedeutet. Dadurch, dass alle zur Verfügung gestellten Anwendungen dem gleichen Quellcode entstammen, muss nicht, wie im Falle von KeePass, auf die Anpassung der Android-App durch andere Entwickler gewartet werden, sondern Änderungen und auch Verbesserungen werden konsistent und konsequent in alle Plattformen integriert.

Code-Qualität

Die verglichenen Manager zeigen darüber hinaus Defizite in den Bereichen Test und Dokumentation auf. Vor allem der Blick als dritte Person auf die Anwendung, um die Verschlüsselungsalgorithmen einzusehen, betont die Wichtigkeit guter Code-Struktur und -Dokumentation. *In dieser Thesis sollen daher Tests für die wichtigen Code-Komponenten die Funktionalität der Anwendung sicherstellen und die Dokumentation dafür sorgen, dass Variablen und Funktionen gut beschrieben sind.* Hier kommt erneut der Vorteil des Code-Sharings zum Vorteil, denn Tests können eventuell nur einmalig implementiert werden, die Code-Dokumentation wird an betreffenden Stellen nur einmalig angefertigt. Die Dokumentation geschieht einmal als JavaScript-Doc-Kommentar zu Beginn einer Funktion oder Klasse und ggf. innerhalb einer Funktion, um deren Funktionsweise näher zu erläutern.

3.3 Vorgehen bei der Literaturrecherche

Zu Beginn der Masterthesis wurde zunächst potentielle Literatur zum ausgewählten Thema gesichtet. Hierfür wurden insbesondere relevante Monographien der Hochschulbibliothek Düsseldorf sowie der Universitätsbibliothek Köln ausgeliehen. Zusätzlich wurde die Internetsuchmaschine „Google“ verwendet.
Durch diese zuerst ungerichtete Literaturrecherche zeigte sich, dass die vorgeschlagene Literatur zu einzelnen gesuchten Themenbereichen, wie „Open-Source-Lizenz“ oder „Authentifikation“, zu ungenau war. Dies hatte zwar zum einen den Vorteil, dass sich der Autor einen ersten Einblick in das Themengebiet verschaffen konnte, zum anderen hatte dies jedoch auch zur Folge, dass schlussendlich keine konkreten Aussagen hinsichtlich des Forschungsthemas gemacht werden konnten. Der Autor kam demnach zur Einsicht, dass eine qualitativ, systematische Literaturrecherche angewendet werden musste, da sie die Möglichkeit bietet, viele Aspekte des Themas vor

dem Hintergrund des aktuellen Wissensstandes zu erfassen.
Aus diesem Grund wurde in einem ersten Arbeitsschritt sämtliche Literatur der ungerichteten Literatursuche bearbeitet. Hierfür wurden entsprechende Unterkategorien relevanter Themenfelder für die Literaturrecherche gebildet, wie die Tabelle 3.8 zeigt.

#	**Thema**
Unterkategorie 1	IT-Sicherheit
Unterkategorie 2	Authentifikation
Unterkategorie 3	Kryptografische Verfahren
Unterkategorie 4	Open-Source-Lizenzen
Unterkategorie 5	Passwortmanager
Unterkategorie 6	Lösungsansatz

Tabelle 3.8: Kategorien wichtiger Themenfelder bei der Literaturrecherche

Unterkategorie 1 „IT-Sicherheit“ wurde verfasst, um alle wichtigen Begriffe der IT-Sicherheit zu sammeln, die für das Verständnis dieser Thesis wichtig sind. „Authentifikation“ wurde als zweite Unterkategorie gewählt, da es sich bei Passwörtern, die ein zentrales Element dieser Arbeit sind, um ein Authentifikationsmerkmal handelt, das eingeordnet und mit anderen Merkmalen verglichen werden muss. Die dritte Unterkategorie wurde „Kryptografische Verfahren“ genannt, weil Passwörter zur Authentisierung bei Systemen genutzt werden, die sensitive Daten mittels kryptografischer Verfahren schützen. Da die Entwicklung des Passwortmanagers als Open-Source-Anwendung geschieht, werden alle diesbezüglichen Recherchen in der Unterkategorie „Open-Source-Lizenzen“ zusammengefasst. Des Weiteren wurde die Unterkategorie „Passwortmanager“ geschaffen, die die Recherche zu existenten Passwortmanagern umfasst. Als zusätzliche Unterkategorie wurde das Unterkapitel 6 „Lösungsansatz“ eingeführt, da sich bei einer ersten Auswertung zeigte, dass neben der problemorientierten thematischen Bearbeitung auch Lösungsansätze abgeleitet werden konnten.

In einem zweiten Arbeitsschritt wurde eine Schlag- und Stichwortliste für die Literaturrecherche erstellt. Sämtliche Schlag- und Stichworte, die bereits bei der ersten Literatursuche verwendet wurden, wurden den verschiedenen Unterkategorien zugeordnet. Des Weiteren wurden diese um zusätzlich relevant erscheinende Begriffe aus den jeweiligen Aufsätzen und Texten ergänzt. Durch die detaillierte Sichtung verschiedener Verschriftlichungen, war es dem Autor gelungen eine Autorenliste von Experten des Fachgebietes zu erstellen. Diese ist dem Anhang A zu entnehmen, skizziert aber nur grob die Ergebnisse bei den einschlägigen Datenbanken, da die Anforderung einen Recherchebericht zu erstellen, erst nach der Recherche aufkam. Zu Recherchezwecken wurden, neben den bereits aufgeführten Bibliotheken, ebenfalls folgende Datenbanken herangezogen:

- Google Scholar

- Researchgate
- USENIX
- Springer Link
- IEEE Computer Society

Nach einer ersten Vorauswahl konnten mindestens 50 zum Themengebiet passende Paper zusammengestellt werden. Nach Durchsicht der genannten Datenbanken konnten für die Thesis schlussendlich 30 wissenschaftliche Arbeiten ausgewählt werden. Die hohe Trefferanzahl bei der Suche nach wissenschaftlichen Arbeiten verdeutlicht auch hier nochmal die Relevanz des Themas dieser Arbeit. Insbesondere bei den Datenbanken Researchgate und Springer Link konnte eine Vielzahl relevanter Erkenntnisse für diese Thesis gewonnen werden.
Insgesamt wurde sich in den Suchanfragen auf die Kombination von höchstens drei Suchbegriffen beschränkt, da sich auch hier gezeigt hat, dass die Trefferquote für thematisch relevante Literatur so am höchsten war und damit auch die Wahrscheinlichkeit, alle relevanten Aspekte des Forschungsbereiches zu erfassen. Weitere Datenbanken wurden aufgrund des eingeschränkten zeitlichen Rahmens nicht berücksichtigt.
Nach der Erstellung der Schlag- und Stichwortliste sowie der Auswahl von Datenbanken für die systematische Literaturrecherche wurden Einschluss- und Ausschlusskriterien definiert, um eine angemessene Erreichung der Zielsetzung zu gewährleisten.
Eines der gewählten Einschlusskriterien stellte ein zeitlicher Faktor dar. In die Recherche für die Basisliteratur wurde lediglich Literatur eingeschlossen, die ab 2005 bis 2017 veröffentlicht wurde. Insgesamt scheint diese acht Jahre umfassende Zeitspanne geeignet, um einerseits alle Entwicklungen einzubeziehen, die zum aktuellen Stand des Forschungsbereichs beigetragen haben, andererseits aber auch um ein gewisses Maß an Aktualität und Übersichtlichkeit zu gewähren. Ausgeschlossen wurde Literatur, bei der bereits im Titel oder in der Zusammenfassung erkennbar war, dass keine thematische Relevanz besteht. Ebenso wurde jene Literatur ausgeschlossen, bei der bereits ein aktuellerer Beitrag zum gleichen Thema vorlag.

Kapitel 4

Prototyp

Im Rahmen der Thesis wird ein Prototyp entwickelt, der auf Windows, MacOS, Linux sowie Android lauffähig sein soll. Zu Beginn des Kapitels wird Motivation für Technologie und Design in Kapitel 4.1 dargelegt, in dem die Entscheidungen der Tools in Bezug auf die vorher definierten Ziele betrachtet werden. Das anschließende Kapitel 4.2 beleuchtet die Architektur der Anwendung und legt wichtige Zusammenhänge zwischen den Bausteinen der Applikation dar. Vertiefend werden in Kapitel 4.3 Implementierungsdetails diskutiert, die die Sicherheit, Erweiterbarkeit und Probleme bei der Implementierung betreffen. Darauf folgend wird die Funktionsweise in Kapitel 4.5 näher erläutert. Dieses Kapitel hat dabei den Anspruch die Anwendung aus Benutzersicht kurz im Sinne einer bebilderten Tour zu beleuchten.
In der Einleitung und dem Kapitel 3.2.8 wurden, auf Basis des Vergleichs existierender Passwortmanager, Anforderungen für einen guten Passwortmanager definiert, die bei der Implementierung des Passwortmanagers berücksichtigt werden sollen und für die Implementierung des Prototyps relevant sind. Diese werden nun kurz und kompakt zusammengefasst.

- Code-Sharing für Quellcode, der sich nicht auf die Benutzeroberfläche bezieht
- Oberfläche soll bei allen Plattformen identisch sein
- Schnittstellen sollen für plattformspezifischen Code implementiert werden
- Kryptografische Algorithmen sollen nach aktuellem Standard gewählt werden, die die Kompatibilität zwischen den Plattformen sicherstellen
- Algorithmus der Schlüsselableitung soll über einen Kostenfaktor verfügen
- Es soll auf eine eigene Implementierung sicherheitskritischer Algorithmen verzichtet werden
- Tests sollen Funktionalität sicherstellen

- Passwörter bei der Eingabe sollen unkenntlich gemacht werden
- Zugriff soll nach gewisser Zeit auf den Manager gesperrt werden

4.1 Motivation hinsichtlich Technologie und Design

Einleitend wurden bereits die wichtigen Kriterien für den Passwortmanager, der den Arbeitstitel „Safeword“ trägt, zusammengefasst. Der Name Safeword setzt sich aus dem englischen Wort „Safe“, für Schließfach, und „Password“ zusammen. Kern des Managers ist das Code-Sharing, das nur mit Tools erreicht werden kann, die eine Cross-Platform-Entwicklung ermöglichen. Ein Tool, das alle Smartphone- und Desktopbetriebssysteme ohne plattformspezifischen Code ansprechen kann, existiert leider nicht, weshalb jeweils eines für Desktop und Mobile gewählt werden muss. Dabei ist zu beachten, dass Code-Sharing nur mit Programmiersprachen betrieben werden kann, die beide Tools unterstützen. Im Desktopbereich sind „Qt“[1] und „Electron“[2] weit verbreitet. Electron bietet den Vorteil, dass die Anwendung auf allen Desktop-Betriebssystemen immer gleich aussieht und zahlreiche Pakete zur Nutzung vorhanden sind. Electron ermöglicht die Programmierung mit Web-Technologien, also HTML, CSS und JavaScript. Da Qt Programmierkenntnisse von C++ voraussetzt, über die der Autor der Masterthesis nicht verfügt, wurde Electron für die Desktopbereich ausgewählt. Dennoch sieht Qt sehr vielversprechend aus und sollte für zukünftige Projekte in Betracht gezogen werden. Da Electron mit JavaScript programmiert wird, muss das Tool zur Programmierung der mobilen Anwendung ebenfalls mit JavaScript programmierbar sein, wie „React Native“[3], VueNative[4] oder „Cordova“[5]. React Native bietet den Vorteil, dass im Gegensatz zu Cordova native Komponenten genutzt werden können, die flüssige Animationen mit 60 Bildern pro Sekunde erlauben. Viele Unternehmen wie Microsoft oder Facebook nutzen React Native um ihre Anwendungen für Android und iOS bereitzustellen, zudem existiert eine große Community um React Native. VueNative basiert auf React Native, implementiert aber nicht die React-Komponenten, sondern welche aus Vue.js. Da VueNative noch sehr jung ist, eine weitere große Abhängigkeit hinzufügt und aufgrund weiterer genannter Vorteile, wurde React Native für die Implementierung der mobilen Anwendung gewählt.
Für eine gute User-Experience fordert Google die Einhaltung der Material Guidelines, die Regeln um Empfehlung zu Gestaltung und Aufbau von Android-Anwendungen im Material Design definieren. Um sowohl die Konformität zu den Material Guidelines als

[1] Qt: https://www.qt.io/, abgerufen am 29.11.2018
[2] GitHub: https://github.com/electron/electron, abgerufen am 29.11.2018
[3] GitHub: https://github.com/facebook/react-native, abgerufen am 29.11.2018
[4] GitHub: https://github.com/GeekyAnts/vue-native-core, abgerufen am 04.12.2018
[5] Apache: https://cordova.apache.org/, abgerufen am 29.11.2018

auch die mögliche Kompatibilität zu iOS zu wahren, wurde für die Nutzeroberfläche der mobilen Anwendung das Paket „NativeBase“[6] ausgewählt. NativeBase erweitert React Native um Komponenten, die sowohl unter iOS als auch Android genutzt werden können, aber je nach Plattform unterschiedlich aussehen. Die Dokumentation der Alternative zu NativeBase „react-native-elements“[7] richtet sich zu stark auf iOS, sodass dieses Paket nicht gewählt wurde. Durch die oben definierte Anforderung, soll nun auch auf dem Desktop eine identische Oberfläche zur Verfügung stehen. Dies wird den Bibliotheken „React“[8] und die Oberflächenbibliothek „MaterialUI“[9] erreicht, die ebenfalls Komponenten anbietet, die die Material Guidelines erfüllen. Eine Alternative zu MaterialUI stellt die Bibliothek „MUI“[10] dar, die aber schlechter gewartet wird und daher MaterialUI gewählt wird. React wird genutzt, um die Code-Struktur aller Plattformen identisch zu halten, denn React Native nutzt im Kern React, um Komponenten zu erzeugen, sodass der Code aller Plattformen gut harmoniert.
Als drittes Kriterium wurde ermittelt, dass Schnittstellen für plattformspezifischen Code erstellt werden sollen, um so den Anteil für Code-Sharing zu maximieren. Dies ist innerhalb der Thesis gelungen, indem ein plattformspezifischer Code in so genannten Modulen verkapselt wird. In Kapitel 4.2.9 werden die Module genauer betrachtet.
Die Anforderungen beziehen sich ebenfalls auf kryptografische Verfahren, die aktuellen Standards entsprechen sollen. Die Algorithmen der Schlüsselableitung sollen über einen Kostenfaktor verfügen und auf eine eigene Implementierung dieser Verfahren soll verzichtet werden. Zur Schlüsselableitung wird das bekannte und weit verbreitete PBKDFv2 genutzt, das über einen verstellbaren Kostenfaktor verfügt. Es existieren zwar noch zahlreiche andere Hashing-Verfahren, jedoch gilt PBKDFv2 immer noch als sicher und kann sowohl von der mobilen als auch der Desktop-Anwendung ohne weitere Abhängigkeiten genutzt werden. Zur Verschlüsselung der Daten wird OpenPGP (Garfinkel, 1994) genutzt, womit Schlüssel, ähnlich wie bei SSH (IETF RFC 4253[11]), erzeugt werden, die eine Ver- und Entschlüsselung von Daten ermöglichen. Einer der Gründe zur Nutzung von OpenPGP ist, dass bisher kaum Passwortmanager existieren, die OpenPGP zur Verschlüsselung nutzen. Des Weiteren wird, nach einigen Problemen mit der Umsetzung einer datenbankbasierten Lösung mit SQLCipher[12], wird das Paket „RxDB“[13] genutzt, das die Daten mittels AES256 verschlüsselt. Eine genauere Beschreibung der Probleme die zur Auswahl von RxDB führten, befindet

[6] GitHub: https://github.com/GeekyAnts/NativeBase, abgerufen am 02.12.2018
[7] GitHub: https://github.com/react-native-training/react-native-elements, abgerufen am 03.12.2018
[8] GitHub: https://github.com/facebook/react, abgerufen am 02.12.2018
[9] GitHub: https://github.com/mui-org/material-ui, abgerufen am 02.12.2018
[10] GitHub: https://github.com/muicss/mui, abgerufen am 03.12.2018
[11] IETF: https://tools.ietf.org/html/rfc4253, abgerufen am 02.12.2018
[12] GitHub: https://github.com/sqlcipher/sqlcipher, abgerufen am 01.12.2018
[13] GitHub: https://github.com/pubkey/rxdb, abgerufen am 19.11.2018

sich in Kapitel 4.3.5. Alle kryptografischen Verfahren werden über die zuvor erwähnten Module, die auf der mobilen Plattform durch native Module unterstützt werden, implementiert.
Wichtig bei jeder produktiv eingesetzten Anwendung, ist die Durchführung von Tests, wie oben ebenfalls definiert. Tests sollen unter Electron und React Native mittels mit „Mocha“[14] und für die Plattformen jeweiligen Erweiterungen durchgeführt werden. Somit ist es möglich, dass für den Core-Code auch der gemeinsame Test-Code verfasst werden kann.
Die letzten beiden Anforderungen richten sich an Funktionen, die zur Sicherheit beitragen. Passwörter werden bei der Eingabe unkenntlich gemacht, sodass das Ausspähen von Daten nicht möglich ist und der Zugriff auf die Daten gesperrt wird, wenn der Nutzer die Anwendung verlässt.
Der Code für den Passwortmanager und für die Masterthesis wird in zwei getrennten Repositories auf dem Gitlab-Server der Hochschule bereitgestellt. GitLab bietet zudem die Möglichkeit mittels „Continuous Integration (CI)“ Aufgaben nach einem Push auf den Master-Branch durchzuführen. CI könnte genutzt werden, um die Codequalität mittels eines Linters anhand vorher definierter Metriken zu bewerten. Da dies durch die Hochschule aber nicht aktiv unterstützt wird und dafür ein eigener Server genutzt werden muss, wird der Linter lokal genutzt, um die Qualität des Codes auf einem hohen Niveau zu halten.

4.2 Architektur

Im Folgenden werden die einzelnen Bestandteile, aus denen sich der eigene Passwortmanager zusammensetzt, erläutert.

4.2.1 Komponenten

Komponenten sind darstellbare Elemente, die wiederverwendet werden können, oder aus Routen ausgelagert wurden. Die Komponenten erhalten ihren Namen dadurch, dass diese von der React-Klasse `Components` abgeleitet werden. In Listing 4.1 ist eine Komponente aus dem Quellcode von Safeword zu sehen.

```
1 import React from 'react'

3 // Additional imports

5 const styles = StyleSheet.create({
6   // ...
7 })
```

[14] GitHub: https://github.com/mochajs/mocha, abgerufen am 27.11.2018

```
 9 class OpenPgp extends React.Component {
10   onOpenFilePicker = () => {
11     // Pass callback so we can react on selection of dir in this
          component
12     NavigationService.navigate('FilePicker', {
13       onSelect: this.onFilePickerClose
14     })
15   }

17   onFilePickerClose = filePath => this.props.handleAdditional('
       filePath', filePath)

19   render() {
20     return (
21       <View>
22         <Text style={styles.text}>Key files directory*</Text>

24         <Button block light onPress={this.onOpenFilePicker}>
25           <Text>Select directory</Text>
26         </Button>
27       </View>
28     )
29   }
30 }

32 export default connect()(OpenPgp)
```

Listing 4.1: Aufbau einer Komponente

Eine Komponente wird als JavaScript-Klasse implementiert, die von `React.Component` erbt (Zeile 9) und somit Lifecycle-Methoden, einen State und Properties erhält.
Eine der Lifecycle-Methoden ist die Methode `render()`, zu sehen in Zeile 19, die die Repräsentation der Komponente an den Aufrufer zurückgibt. Im Code kann der `render()`-Funktion entnommen werden, dass ein an HTML angelehnter Code zurückgegeben wird (Zeile 21-27). Diese Syntax wird JSX[15] genannt und ermöglicht innerhalb von JavaScript-Dateien zugehörigen UI-Code zu verfassen. JSX erhöht den Komfort für den Entwickler, sodass Komponentenlogik und die darzustellende Repräsentation gebündelt dargestellt werden. Des Weiteren zeigt die Methode `render()` in den Zeilen 21, 22 und 24, dass die Komponente `OpenPgp` durch eine Komponente `<View>` und deren Kinder repräsentiert wird. Mittels geschwungener Klammern können auch Variablen oder Objekte eingebettet werden, wie am Attribut `onPress` der Komponente `<Button>` in Zeile 24 zu sehen ist.
Ein State ist für die Lebenszeit der Komponente verfügbar und enthält die Informa-

[15] React: https://reactjs.org/docs/introducing-jsx.html, abgerufen am 19.11.2018

tionen, die für die Lebenszeit der Komponente zwischengespeichert werden müssen. Eine Änderung des States würde ein erneutes Rendern bewirken, weil sich beispielsweise der Text in einem Eingabefeld geändert hat. Dies zeigt auch die Stärke des State-Systems in React: Bei einer Änderung des States wird nur die betroffene Komponente einschließlich der Kinder neu gezeichnet, anstatt den ganzen DOM neu zu zeichnen, was sehr in effizient ist. Jede Instanz der Komponente erhält sowie verwaltet einen eigenen State, der nach jedem Rendern der Anwendung bestehen bleibt. Eine Komponente hat jedoch keinen Zugriff auf den State einer Kindkomponente. Sollte der Zugriff darauf gewünscht sein, müssen Funktionen als Property übergeben, oder eine zentrale State-Verwaltung genutzt werden. Die Komponente `OpenPgp` aus Listing 4.1 erweitert den standardmäßig leeren State nicht, da in diesem Beispiel keine Informationen in der Komponente zwischengespeichert werden müssen.
Properties, in der technischen Dokumentation von React „props“ genannt, enthalten Übergabeparameter an die referenzierte Komponente. Die Komponente `<Button>` erhält in Zeile 24 als Property `onPress` eine Funktion, auf die in der Button-Komponente mittels `this.props.onPress` zugegriffen werden kann. Wenn sich die Properties einer Komponente ändern, wird auch ein erneuter Render-Prozess gestartet.
Komponenten lassen sich ähnlich optisch anpassen, wie es aus der Web-Entwicklung bereits bekannt ist. Dadurch, dass es sich bei der Desktop-Anwendung um pure Web-Entwicklung handelt, sind auch alle gewohnten CSS-Regeln verfügbar, wobei sich die Syntax leicht verändert hat. Dies ist darauf zurückzuführen, dass Objekte nur bestimmte Schreibweisen akzeptieren. So ist das Minus in JavaScript ein mathematischer Operator und deshalb muss für die CSS-Regel `margin-right` das Schlüsselwort `marginRight` genutzt werden. Die optische Anpassung der mobilen Anwendung geschieht analog zur Desktop-Anwendung, jedoch sind einige Schlüsselworte nicht verfügbar und auch die Wertebereiche der Attribute unterscheiden sich. Styles werden in Zeile 5-7 des Listings 4.1 erstellt, aufgrund des Umfangs wird aber nur die Erstellung eines sogenannten „Stylesheets“ dargestellt.

4.2.2 Routen

Routen sind technisch gesehen identisch zu Komponenten. In dieser Thesis werden Routen aber von Komponenten zur besseren Strukturierung getrennt. Der Name entstammt ebenfalls der Web-Entwicklung und bezieht sich auf die verschiedenen möglichen Pfade in der URL einer Web-Anwendung. So erhält sowohl der Pfad `/settings` als auch `/settings/sync` jeweils eine eigene Route-Komponente. Listing 4.2 zeigt den Code der Desktop-Route `/settings/about`.

```
1 import React from 'react'
```

```
2  // Additional imports

4  const style = theme => ({
5    // ...
6  })

8  class About extends React.Component {
9    changeLocation = path => this.props.history.push(path)

11   render() {
12     const { classes } = this.props

14     return (
15       <Navigation name="About">
16         <List className={classes.list}>
17           <ListItem className={classes.listItem}>
18             <ListItemText
19               primary="Developer"
20               secondary="David Littig, 2018"
21             />
22           </ListItem>
23           {/* Additional list items */}
24         </List>
25       </Navigation>
26     )
27   }
28 }

30 export default withRouter(withStyles(style)(About))
```

Listing 4.2: Aufbau einer Route-Komponente

Listing 4.2 ist zu entnehmen, dass sich Routen ähnlich aufbauen lassen wie Komponenten, aber sich dadurch, dass diese so spezifisch sind, per Konvention nur einmal nutzen lassen. Die Komponente `<Navigation>` aus Zeile 15 bildet den Rahmen und enthält die Seitenleiste mit den Navigationsoptionen und die obere „AppBar“ mit dem Titel „About“. Die Route „About“ stellt eine Liste (Zeile 16) mit weiteren Navigationsoptionen dar, die in den Zeilen 17-23 mittels `ListItem`-Komponenten erzeugt werden. Eine dieser Komponenten erhält Text-Properties, die dem Nutzer Informationen über den Entwickler mitteilen, wie in Zeile 19-20 angedeutet.

4.2.3 Store

Der Store ist die zentrale, dynamische und nicht persistente Datenhaltung für den Betrieb der Anwendung. Der Store wird mit Hilfe von „Redux“ realisiert, einem State Container für JavaScript-Anwendungen. Der durch Redux implementierte Informationsfluss zwischen den Komponenten und dem Store wird in Kapitel 4.2.4 detailliert

dargestellt.
Der Store steht nach dem Start der Anwendung zur Verfügung und wird zuerst mit einem initialen State befüllt. Das Store-Objekt besteht aus einzelnen State-Objekten, die in Listing 4.3 dargestellt werden.

```
1  {
2    "auth": {
3      "isAuthenticated": false,
4      "token": null
5    },
6    "setup": {
7      "error": null
8    },
9    "config": {
10     "readOnly": [],
11     "store": Object
12   },
13   "persistence": {
14     "adapter": null,
15     "passwords": [],
16     "groups": []
17   },
18   "message": {
19     "info": null
20   },
21   "settings": {
22     "readOnly": [],
23     "store": Object
24   },
25   "sync": {
26     "adapter": null
27   }
28 }
```

Listing 4.3: Struktur des globalen Stores

Dem Quellcode kann unter anderem aus den Zeilen 2, 6 und 9 entnommen werden, dass im Store verschiedene States als Objekte hinterlegt sind. Dies ermöglicht eine Separation der Informationen nach Einsatzgebiet. Aus Performance-Sicht sollte dies auch zwingend so umgesetzt werden, denn Komponenten, die auf den gesamten Store zugreifen würden, rendern erneut, wenn sich ein Wert im Store ändert. Daher kann eine Komponente nur Teile des Stores „abonnieren“ und zeichnet auch nur bei Änderung der abonnierten Daten die Komponente neu.
Teil des Store sind sowohl die Konfiguration (`config` in Zeile 9) als auch die Einstellungen (`settings` in Zeile 21). Diese besitzen unter dem Schlüsselwort `store` jeweils eine JavaScript-Klasse, die den Zugriff auf die Datenbank mittels RxDB verkapseln. Mittels dieser Objekte können gespeicherte Daten geschrieben oder gelesen werden. Für den schnellen Zugriff auf die Daten bietet der Store das Feld `readOnly`,

in dem dann alle Daten der Konfiguration bzw. Einstellungen verwaltet werden. Weiterreichende Details und die Daten, die die Konfigurationsinstanz verwaltet, werden in Kapitel 4.2.7 erläutert. Auf Basis der Konfiguration im Store und der erstellten Ordner im Dateisystem, kann Safeword erkennen, ob der Manager bereits aufgesetzt wurde oder nicht und leitet zum Anmeldefenster bzw. zum Setup-Assistenten weiter. Ein wichtiger Schritt bei der Anmeldung beim Passwortmanager ist die sogenannte Hydrierung des Stores (engl. hydration). In diesem Schritt werden persistente Daten, wie Passwörter und Gruppen, entschlüsselt und in dem Store gespeichert. Der Store dient bei dem Betrieb der Anwendung als zentrale Stelle zum Bezug von Daten, was zwar eine erhöhte Komplexität bei der Synchronisierung von Datenbestand und Store bedeutet, aber an allen anderen Stellen die Programmierung erleichtert, da die Daten so zentral und ohne Wartezeit zur Entschlüsselung verfügbar sind. Der Abruf ist dadurch unabhängig vom Adapter, da der Adapter für die Befüllung des States mit den korrekten Passwort- oder Gruppendaten verantwortlich ist.

4.2.4 Datenfluss

Verantwortlich für die temporäre Datenspeicherung ist der globale Store, in dem übergreifend über alle Komponenten Daten abgelegt werden können. Dieser ist mit Hilfe der Bibliothek Redux implementiert. Redux besteht aus fünf Bestandteilen: Action Creators, React Komponenten (UI), Store, den darin enthaltenen States und Reducers. Alle diese Bestandteile sind eng miteinander verbunden, wie Abbildung 4.1 zeigt.

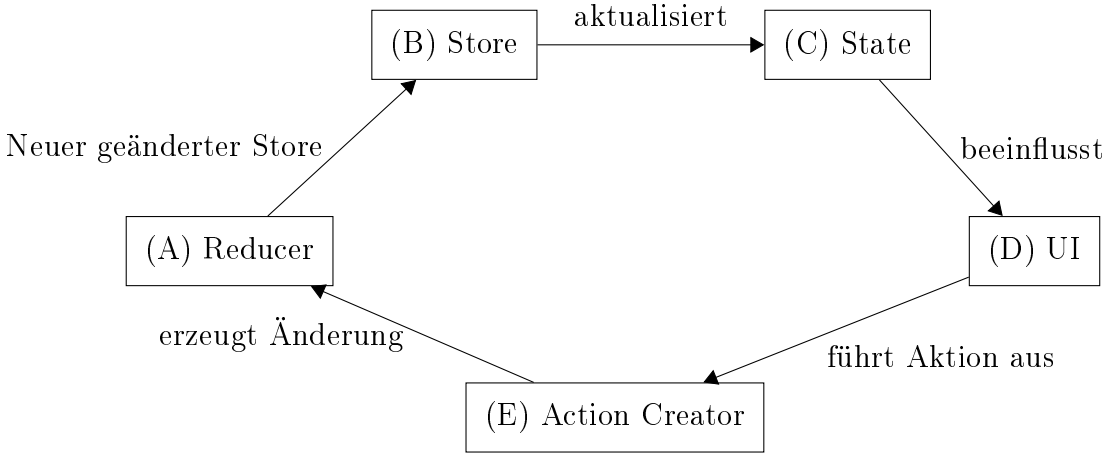


Abbildung 4.1: Datenfluss der Anwendung

Wird die Anwendung gestartet, wird zu Beginn der Store (B), bekannt aus Kapitel 4.2.3, aufgebaut. Dazu werden sogenannte Reducer (A) benötigt. Ein Reducer ist eine Funktion, die angepasste State-Objekte basierend auf einer Änderungsaktion

erzeugt. Was genau eine Änderungsaktion ist, wird im späteren Verlauf dieses Abschnitts erläutert, da zum Start der Anwendung diese noch nicht existieren. Liegt keine Änderungsaktion oder kein State vor, soll der Reducer einen initialen State definieren. Dieser initiale State wird von jedem registrierten Reducer abgefragt. Da ein Reducer nicht den gesamten Store inklusive aller States verändern und nur einen einzigen State verwalten kann, liegen bei Safeword mehrere Reducer vor, weil aus Gründen der Les- und Wartbarkeit aber auch der Geschwindigkeit mehrere States implementiert wurden. Der Store stellt nach der Initialisierung States zu Verfügung, die die Komponenten abonnieren können. Die Anbindung geschieht über die Funktion `connect()` und einer weiteren als Parameter übergebenen Funktion, wie in Listing 4.4 gezeigt wird.

```
1 // ... Component code
2 const mapStateToProps = ({persistence: {adapter}}) => ({adapter})

4 export default connect(mapStateToProps)(Group)
```

Listing 4.4: Anschließen einer Komponente an den Redux-Store

Die übergebene Funktion `mapStateToProps` in Zeile 2 erhält als ersten Parameter das Store-Objekt, das alle States enthält. Dort wird eine Funktion von JavaScript genutzt, die sich „Destructuring“[16] nennt. Damit ist es möglich mit einer kurzen Schreibweise auf die Attribute eines Objektes zuzugreifen und Attribute zur Verfügung zu stellen. Der Code `{persistence: {adapter}}` ist funktional identisch zu `const adapter = store.persistence.adapter`. Wird das Destructuring direkt in der Parameterliste einer Funktion genutzt, kann zudem auf die Definition einer Variable, die den Store enthält, verzichtet werden. In obigem Beispiel hat der Entwickler nur Zugriff auf die Variable `adapter` aus dem Store, aber keinen Zugriff auf das Store-Objekt mehr, das ursprünglich der Funktion übergeben wurde. Als Rückgabewert wird ein Objekt erzeugt, das den Adapter als Parameter enthält. Die Funktion dient der Überführung von Daten der States (C) in die Properties der Komponenten, wie auch schon der Name der Funktion suggeriert. Dass die Daten in den Properties einer Komponente verfügbar sind, bietet zwei Vorteile. Zum Einen kann auf wichtige Daten, die eine Komponente benötigt, wie gewohnt über die Properties zurückgegriffen werden, zum Anderen kann die Komponente automatisch auf Änderungen reagieren, wenn sich der State im Store ändert. Denn wird der Store geändert, wird der neue Store an abonnierende Komponenten weitergeleitet. Mit Hilfe der übergebenen Funktion werden die Properties aktualisiert. Haben sich die Properties verändert, wird die Komponente neu gerendert. So kann automatisch und effizient eine Komponente auf State-Änderungen reagieren, falls sich ein für die Komponente relevanter Wert ge-

[16] ExploringJS: http://exploringjs.com/es6/ch_destructuring.html, abgerufen am 24.11.2018

ändert hat. Die Funktion `connect` in Zeile 4 erzeugt eine modifizierte Komponente `Group`, die über die übergebene Funktion `mapStateToProps` Daten aus dem Store in Properties überführt.
Die React-Komponente (D) stellt auf Basis des States und der Properties eine visuelle Repräsentation der Daten bereit. Einige Komponenten können sogenannte „Actions“ ausführen, um Daten zu manipulieren, beispielsweise kann über einen Knopf ein neues Passwort erstellt werden. Um Daten im Store zu hinterlegen, also das Passwort in der Übersicht aller bisher erzeugter Passwörter anzuzeigen, muss die Action einen Action Creator (E) mit der Erstellung einer Änderungsaktion beauftragen. Exemplarisch folgt ein Auszug aus dem Quelltext des Action Creators für den `Persistence`-State in Listing 4.5.

```
1  export const PERSISTENCE_UPDATE_PW = 'PERSISTENCE_UPDATE_PW'
2  // other action types

4  // Action creator to update passwords
5  const updatePasswords = passwords => ({
6    type: PERSISTENCE_UPDATE_PW,
7    passwords
8  })

10 // ...

12 export const Persistence = { updatePasswords, ... }
```

Listing 4.5: Action Creator für den Perisistence-State

Die zuvor genannte Änderungsaktion wird in Zeile 5-8 erstellt. Diese wird durch ein Objekt repräsentiert, die über einen Typ sowie weitere optionale Parameter verfügt, die nach belieben gesetzt werden können. Da es sich in Listing 4.5 um die Action `updatePasswords` handelt, wird als Funktionsparameter das Array der Passwörter übergeben, das wiederum als Eigenschaft der Änderungsaktion genutzt wird. Änderungen an den States im Store werden nicht direkt ausgeführt, sondern die generierte Änderungsaktion des Action Creators wird über den Store dem Reducer mittels `store.dispatch(Persistence.updatePasswords([ ... ]))` übergeben. Der Reducer (A) erzeugt ein neues State-Objekt, das die Änderungen enthält und anschließend in den Store (B) einpflegt wird. Der aktualisierte State (C) wird an die Komponenten (D) weitergereicht und der Nutzer sieht dann, nachdem die Komponente erneut gerendert wurde, dass das neue Passwort in der Übersicht angezeigt wird.
An diesen Kreislauf halten sich auch alle anderen Komponenten, die mit dem Redux-Store verbunden sind. Alternativ kann auf den Redux-Store verzichtet werden, indem über mehrere Ebenen hinweg Event-Listener, die auf Änderungen von zum Beispiel

State-Information reagieren, und weitere Eigenschaften als Properties übergeben werden. Die Event-Listener werden von einer im Komponentenbaum tiefer gelegenen Komponente genutzt, um einen Wert wieder nach oben zu reichen. Dieser Listener muss aber von allen dazwischen liegenden Ebenen immer weitergegeben werden, sodass sich die Komplexität stark erhöht und die Wartbarkeit verschlechtert. Dieses Verfahren bietet sich also nur bei flachen Komponentenhierarchien an, wie beispielsweise im Setup-Prozess von Safeword.

4.2.5 Modelle

Modelle werden genutzt, um die Objekte, die diese Anwendung nutzt, strukturiert zu erfassen. Konkret handelt es sich dabei um die Modelle „Password“ und „Group“, die eng miteinander verknüpft sind, wie Abbildung 4.2 zeigt.

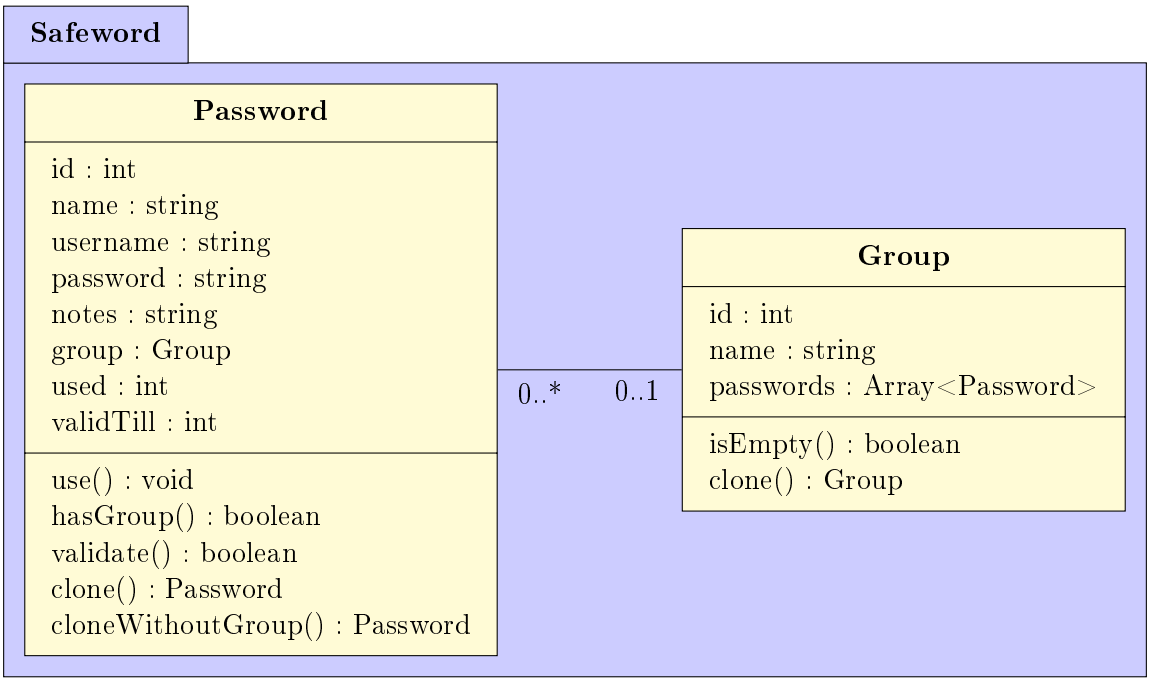


Abbildung 4.2: Modelle von Safeword

Das Model Password enthält Attribute wie ID, Label/Name, Nutzername, das eigentliche Passwort, Notizen, letztes Nutzungsdatum, Gültigkeitsdatum und natürlich eine Referenz auf eine Instanz einer Gruppe. Wird das Passwort keiner Gruppe zugeordnet, enthält das Passwort trotzdem eine Gruppen-Instanz, die dann aber nur über die standardmäßig vorinitialisierten Werte verfügt. Diese Design-Entscheidung hat den Grund, dass an allen Stellen des Passwortmanagers davon ausgegangen werden kann, dass eine Instanz vorliegt, was die Anzahl an `if/else`-Verzweigungen reduziert. Das Modell Group verfügt über eine ID, einen Namen und ein Array mit Passwörtern,

die dieser Gruppe angehören. Die Liste mit zugehörigen Passwörtern ist notwendig, damit in kurzer Zeit mit wenig Rechenaufwand alle Passwörter, die einer Gruppe angehören, auf einer Seite angezeigt werden. Wäre diese Referenz nicht implementiert, müsste bei jedem Aufruf über alle Passwörter iteriert werden, um alle Passwörter zu finden, die auf die aktuell betrachtete Gruppe verweisen.
Sowohl Password als auch Group verfügen über `clone()`-Funktionen, sodass alle Werte der Instanzvariabeln auch in einer neuen Instanz verfügbar sind. Das Password-Modell verfügt darüber hinaus über eine Funktion, die das Passwort ohne der zugehörigen Gruppe klont. Die Funktionen zum Klonen einer Instanz werden für das Persistieren von Änderungen genutzt, denn die Schnittstelle erfordert ein altes und ein neues Objekt, das die vom Nutzer gemachten Änderungen beinhaltet. In JavaScript sind Objekte und Arrays veränderbar (engl. *mutable*), sodass Änderungen an einem Objekt nicht automatisch ein neues Objekt erzeugen. Folglich muss, um Änderungen dokumentieren zu können, die Instanz kopiert und zwischengespeichert werden.

4.2.6 Adapter

Adapter sind das Herzstück des Passwortmanagers. Diese stellen eine modulare Architektur bereit, sodass die Anwendung um weitere Speicherarten, Synchronisierungsziele, Import-/Exportformate und Authentifizierungsarten erweitert werden kann. Jeder Adapter erbt von einer unspezifischen Adapter-Klasse, beispielsweise existieren zwei Adapter für Speicherarten, namentlich `OpenpgpAdapter` und `RxAdapter`, die von der Klasse `PersistenceAdapter` erben. Passender wäre hier ein Interface wie in Java gewesen, das leider in JavaScript nicht integriert ist. Die Klasse `PersistenceAdapter` enthält somit Methoden, die von den erbenden Klassen überschrieben werden können. Des Weiteren enthält diese Klasse auch Funktionen, die in allen Unterklassen verfügbar sind, wie beispielsweise Funktionen zum Aktualisieren des globalen States, wenn sich Passwort- oder Gruppen-Instanzen verändert haben.
Wie bereits in Kapitel 4.2.3 erwähnt, werden für den Betrieb der Anwendung alle Daten aus dem Store bezogen, sodass wie hier immer auf einen konfigurierten Adapter zurückgegriffen werden kann. Dadurch, dass alle Adapter durch die Oberklasse zumindest immer über alle Funktionen, wenn auch ohne Funktionalität, verfügen, muss bei der Entwicklung nicht beachtet werden, welcher Adapter-Typ angesprochen wird.

4.2.7 Basiskonfiguration

Die Basiskonfiguration verkapselt, wie bereits erwähnt, die Datenbankinstanz, in der die Konfiguration hinterlegt wird. Das Datenbankschema der Tabelle, in der die Kon-

figuration zudem verschlüsselt abgelegt wird, wird in Abbildung 4.3 dargestellt.

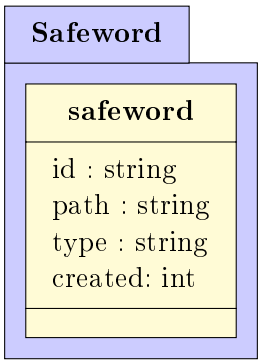


Abbildung 4.3: Datenbankschema der Konfigurationstabelle

Im Quellcode aus Listing 4.3 ist ersichtlich, dass sowohl bei den Einstellungen als auch bei der Konfiguration eine direkter Zugriff auf die Daten über das Feld `readOnly` möglich ist. Dadurch, dass die Konfiguration nur beim Starten der Anwendung und bei der Anmeldung am Passwortmanager relevant ist, wurde hier entschieden, dass das Feld im globalen Store für zukünftige Weiterentwicklungen zwar vorhanden ist, aber nicht genutzt wird, da die dadurch entstehende Komplexität der Synchronisierung nicht gerechtfertigt ist. Der Zugriff auf die Einstellungen geschieht daher über das im Feld `store` hinterlegte Konfigurationsobjekt.
Bei der Konfiguration ist das Feld `created` besonders, denn dieses zeichnet die Erstellungszeit auf, die auch in einer Datei auf dem Dateisystem abgelegt wird. Dies soll als weiterer Faktor dienen, sodass die gespeicherten Daten nur auf der erstellenden und synchronisierten Maschine nutzbar sind, aber nicht wenn bei einem Angriff, die Schlüsseldateien oder die Passworte entwendet werden. So wird es dem Angreifer effektiv erschwert mit einem kopierten Datensatz an einer anderen Maschine einen neuen Datenspeicher mit den vorhandenen Daten aufzusetzen.

4.2.8 Einstellungen

Die Einstellungen der Anwendung werden in der „Settings“-Instanz verwaltet, die, analog zur Basiskonfiguration, ebenfalls eine Datenbankinstanz verkapselt. Das zugrundeliegende Schema dieser Instanz wird in Abbildung 4.4 dargestellt.
Die Einstellungen werden durch Key-Value-Paare realisiert, sodass nahezu beliebige Informationen als String hinterlegt werden können. Listing 4.6 zeigt einen Auszug der verwalteten Schlüssel, sodass der Zugriff sicher über zentral abgelegte Konstanten erfolgen kann.

```
1 const NOTIFICATIONS_ENABLED = 'NOTIFICATIONS_ENABLED'
2 const LOGGING_ENABLED = 'LOGGING_ENABLED'
```

```
3 const PROFILE_FIRSTNAME = 'PROFILE_FN'
4 // ...
```

Listing 4.6: Konstanten zum Zugriff auf Einstellungen

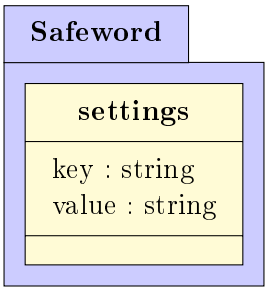


Abbildung 4.4: Datenbankschema der Einstellungstabelle

Wie bei der Basiskonfiguration, wird auch bei den Einstellungen im Redux-Store das Feld `readOnly` angeboten, das eine mit der Datenbank synchronisierte `Map`[17] zur Verfügung stellt. Mittels der `get`-Funktion kann auf die Inhalte mittels der aus Listing 4.6 gezeigten Konstanten zugegriffen werden. Aus Gründen der Performanz, wenn beispielsweise vier Felder direkt nacheinander aktualisiert werden, aber nur eine Aktualisierung benötigt wird, muss das Feld `readOnly` durch den Entwickler auf aktuellem Stand gehalten werden. Dies geschieht mittels der Funktion `initReadOnly` der Settings-Instanz, die im Redux-Store unter dem Schlüsselwort `store` abgelegt ist.

4.2.9 Module

Module sind Schnittstellen, die einen Zugriff auf plattformabhängige Funktionen normieren. Beispielsweise besitzt Node.js für den Desktop eine andere Schnittstelle, um auf das Dateisystem zuzugreifen, als das Paket `react-native-fs`[18], das für die mobile Anwendung genutzt wird. Es folgt eine Übersicht über alle erstellten Module und eine kurze Beschreibung dieser.

- **aes**: Stellt Verschlüsselungsverfahren AES über `crypto`-API von Node.js auf Desktop-Plattform bzw. über natives Modul bei mobiler Anwendung bereit. Genutzt wird AES256 im CBC-Modus mit PKCS5-Padding
- **clipboard**: Kopieren eines String in die Zwischenablage
- **connectivity**: Überprüfen, ob Gerät mit dem Internet verbunden ist und ob mobile Daten genutzt werden

[17] Mozilla Developer Network: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Map, abgerufen am 26.11.2018
[18] GitHub: https://github.com/itinance/react-native-fs, abgerufen am 26.11.2018

- **filesystem**: Zugriff auf das Dateisystem über `fs`-API von Node auf dem Desktop und `react-native-fs` bei der mobilen Anwendung
- **keystore**: Sicheres Ablegen des Schlüssels im Keyring (Linux), Keychain (MacOS), Credential Vault (Windows), Keychain (Android) und Keystore (iOS). Nutzt für den Desktop `node-keytar`[19] und für Mobile `react-native-keychain`[20]
- **locker**: Sperren des Bildschirms bei Verlassen der Anwendung
- **masterkey**: Erzeugen und Lesen des Schlüsselmaterials, bestehend aus Hash des Passworts und mit AES verschlüsseltem, zufälligem Schlüssel
- **notification**: Erzeugen von Benachrichtigungen auf Betriebssystemebene
- **openpgp**: Stellt `openpgpjs` für OpenPGP über das offizielle Paket und ein eigens dafür erstelltes Paket `react-native-openpgpjs`[21], das plattformspezifische Ergänzungen enthält, bereit
- **pbkdf2**: Stellt die Hashing-Funktion PBKDFv2 über `crypto`-API von Node.js auf Desktop-Plattform bzw. über natives Modul bei mobiler Anwendung bereit. Als HMAC wird SHA512 genutzt, der Salt ist 256 Bit lang
- **permissions**: Anfordern der Rechte, um lokalen Speicher zu lesen und zu beschreiben
- **protocol**: Protokolliert Authententifizierungsversuche im Dateisystem
- **random**: Erzeugt kryptografisch sichere Zufallszahlen
- **sha**: Stellt SHA256 über `crypto`-API von Node.js auf Desktop-Plattform bzw. über natives Modul bei mobiler Anwendung bereit
- **webdav**: Stellt Zugriff auf WebDAV-Quellen über das Paket `webdav-client`[22] bereit

4.2.10 Native Module für Android und iOS

Native Module sind Erweiterungen in React Native, die es ermöglichen, den Code nicht nur in JavaScript für die V8-Engine zu schreiben, sondern auch native Funktionen und deren Geschwindigkeitsvorteile zu nutzen. In Safeword wurden sowohl

[19] GitHub: https://github.com/atom/node-keytar, abgerufen am 26.11.2018
[20] GitHub: https://github.com/oblador/react-native-keychain, abgerufen am 26.11.2018
[21] GitHub: https://github.com/dlittig/react-native-openpgpjs, abgerufen am 26.11.2018
[22] GitHub: https://github.com/perry-mitchell/webdav-client, abgerufen am 26.11.2018

kryptografische Algorithmen, wie AES, PBKDFv2 und SHA256 aufgrund der höheren Geschwindigkeit, als auch Benachrichtigungen, da diese eine native Funktion sind, als native Module umgesetzt. Im Folgenden wird erläutert, warum diese nativen Module notwendig sind und wie diese implementiert wurden.

Kryptografische Algorithmen

Kryptografische Algorithmen sind sehr aufwendig und in React-Native als JavaScript-Code in der Ausführung langsamer als nativer Code, der beispielsweise in Java oder Swift geschrieben wurde. Deshalb sollten Pakete auf GitHub gefunden werden, die die kryptografischen Verfahren mittels nativer Module implementieren. Leider sind diese Pakete teilweise schlecht gewartet, unvollständig oder basieren nur auf JavaScript-Code, sodass diese für die Nutzung in Safeword gänzlich ungeeignet sind.
Deshalb wurde eine Implementierung mittels eines nativen Moduls erwogen, namentlich `Encryption`. Die kryptografischen Verfahren wurden aber nicht selbst implementiert, sondern dabei handelt es sich um die nativen Funktionen des Betriebssystems, wie Listing 4.7 zeigt.

```
1  // imports ...
2  public class EncryptionModule extends ReactContextBaseJavaModule
       {
3    private static final String PBKDF = "PBKDF2withHmacSHA512";

5    // Functions for aes encryption, decryption, sha512 hashing ...

7    /**
8    * Derives a key from a passphrase based on PBKDF2 with
         HmacSHA512. Resolves the promise with the hex encoded string
          representation.
9    * @param key Key as normal utf8 string
10   * @param salt Salt as normal utf8 string
11   * @param cost Cost factor
12   * @param bytes Length of the generated key in bytes
13   * @param promise Promise object provided by RN Bridge
14   */
15   @ReactMethod
16   public void pbkdf2(String key, String salt, Integer cost,
         Integer bytes, Promise promise) {
17     try {
18       PBEKeySpec spec = new PBEKeySpec(key.toCharArray(), salt.
             getBytes(StandardCharsets.UTF_8), cost, bytes*8);
19       SecretKeyFactory factory = SecretKeyFactory.getInstance(
             PBKDF);
20       byte[] hash = factory.generateSecret(spec).getEncoded();

22       promise.resolve(Hex.encodeHex(hash, false));
23     } catch(Exception e) {
24       promise.resolve(null);
```

```
25      }
26    }
27  }
```

Listing 4.7: Natives Modul Encryption

Ein Modul muss zwingend von der Klasse `ReactContextBaseJavaModule` erben, die von React Native zur Verfügung gestellt wird, ersichtlich in Zeile 2. Anschließend können Funktionen definiert werden, die dann über die React Native Bridge im JavaScript-Code genutzt werden können. Zudem können auch Konstanten definiert werden, wie bei einer Java-Klasse üblich und in Zeile 3 geschehen. In diesem Modul sind noch zahlreiche weitere Funktionen vorhanden, in diesem Beispiel wird aber nur die Funktion `pbkdf2()` näher betrachtet. Funktionen, die über die Bridge verfügbar sein sollen, werden mit der Annotation `@ReactMethod` (Zeile 15) gekennzeichnet. Somit können gezielt Funktionen freigegeben werden und Hilfsfunktionen bleiben verborgen. Anschließend wird der Funktionskopf implementiert (Zeile 16), die mit der JavaScript-Funktion übereinstimmt. Einzig der Parameter `Promise` existiert nur im Java-Code und ermöglicht Zugriff auf die Promise-API[23] aus JavaScript. Im Funktionskörper wird nach Initialisierung der Schlüsselspezifikation (Zeile 18) und notwendigen Factory (Zeile 20) der Hashwert errechnet. Anschließend wird das resultierende Byte-Array als Hex-String encodiert und als Ergebnis des Promises zurückgegeben. Sollte es zu einem Fehler bei dem Hashing-Verfahren kommen, wird dieser im `catch`-Block abgefangen und das Promise erhält als Ergebnis `null`.
Die Nutzung der nativen Module geschieht wie in Listing 4.8 am Beispiel PBKDFv2 gezeigt.

```
1  import { NativeModules } from 'react-native'
2  const Encryption = NativeModules.EncryptionModule

4  const hash = (password, salt) => new Promise(resolve => {
5    Encryption.pbkdf2(password, salt, 25000, 32) //256 Bit
6      .then(key => {
7        if(key !== null)
8          resolve(Buffer.from(key, 'hex'))
9        else reject(false)
10     })
11 })

13 export { hash }
```

Listing 4.8: Nutzung des nativen PBKDFv2-Moduls

[23] Mozilla Developer Network: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise, abgerufen am 26.11.2018

Alle nativen Module werden in Zeile 1 importiert, der Zugriff auf das `Encryption`-Modul geschieht in Zeile 2. Wie bereits erwähnt, kann die Funktion dann einfach genutzt werden und die Parameterliste deckt sich mit der, der Java-Funktion, ausgenommen des letzten Parameters `promise` (Zeile 5). Mittels `then()` kann auf das Ergebnis des Promises reagiert werden. Im Code wird beispielsweise der Hash, falls dieser erfolgreich ermittelt werden kann (Zeile 7), als Buffer weiterverarbeitet (Zeile 8). Andernfalls wird das Promise mit dem Wert `false` abgewiesen.

Benachrichtigungen

Benachrichtigungen lassen sich standardmäßig nicht mit React Native erzeugen, weshalb zuerst nach einem geeigneten Paket auf GitHub gesucht und schließlich `react-native-notifications`[24] gefunden wurde. Auf GitHub war bereits sichtbar, dass das Paket kaum gewartet wird und die Anzahl der offenen Issues, scheinbar aus Mangel an aktiven Entwicklern, stetig wächst. Benachrichtigungen konnten zwar erstellt werden, jedoch ohne das Launcher-Icon der Anwendung. Eine Option zur Übermittlung des zu nutzenden Icons gibt es nicht und viele Funktionen des Pakets, wie etwa Push-Benachrichtigungen, werden nicht genutzt, was schlussendlich zum Entfernen des Pakets und zur Entscheidung der eigenen Implementierung führte.
Das native Modul `Notification` erzeugt Benachrichtigungen und nutzt dabei das Icon der Anwendung. Zudem ist das Modul auch mit Android 8.0 Oreo und neuer kompatibel, da die neue Schnittstelle ebenfalls berücksichtigt wurde. Aus Gründen des Umfangs dieser Thesis, wird hier auf den nativen Code des nativen Moduls verzichtet. Der Aufruf des Moduls geschieht analog zum Beispiel, das in Listing 4.8 gezeigt wurde.

4.3 Implementierung

In folgendem Abschnitt wird auf Implementierungsdetails, von unter anderem der Authentifikation und den Adaptern, sowie die Probleme bei der Implementierung des Managers eingegangen.

4.3.1 Authentifikation

Die Authentifikation geschieht über das Modul `masterkey`. Der Masterkey stützt sich auf viele Module, die miteinander verbunden das Schlüsselmaterial erzeugen, wie Abbildung 4.5 zeigt.

[24] GitHub: https://github.com/wix/react-native-notifications, abgerufen am 26.11.2018

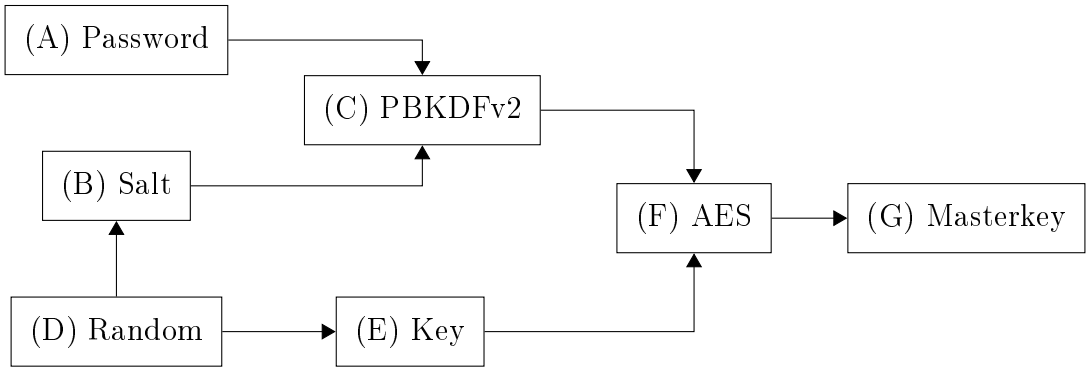


Abbildung 4.5: Erstellung eines Masterkeys

Bei dem Einrichtungsprozess vergibt der Nutzer ein Master-Passwort, das in Abbildung 4.5 mit dem Buchstaben „A“ gekennzeichnet wurde, das zur späteren Authentisierung genutzt wird. Dieses Passwort wird von der Hash-Funktion PBKDFv2 (C) verarbeitet, die als weiteren Parameter einen zufällig generierten Salt mit einer Länge von 256 Bit von dem Zufallsgenerator (D) erhält. Des Weiteren wird durch den Zufallsgenerator (D) ein zufälliger Schlüssel (E) mit einer Länge von 1024 Bit erzeugt. Der zufällig generierte Schlüssel (E) stellt hierbei den Schlüssel dar, der für die Verschlüsselung der Daten genutzt wird. Das vom Nutzer definierte Master-Passwort dient lediglich als Schlüssel für die Verschlüsselung mit AES (F). Der 1024 Bit starke Schlüssel wird verschlüsselt und der daraus resultierende Masterkey (G) wird in der Datei `master.swk` abgelegt.
Möchte sich der Nutzer zur Nutzung des Managers authentisieren, gibt dieser sein Master-Passwort (A) ein und bestätigt die Eingabe des Passworts. Anschließend wird der genutzte Salt (B) aus einer Datei gelesen. Beide Parameter werden genutzt, um den Hash des eingegebenen Passwortes mit PBKDFv2 (C) zu berechnen. Anschließend wird dieser Hash als Schlüssel genutzt, um den in der Datei `master.swk` abgelegten Ciphertext mittels AES (F) zu entschlüsseln. Ist das Master-Passwort korrekt, kann der Ciphertext entschlüsselt werden und die Anwendung erhält den Key (E), der zum Entschlüsseln der Daten benötigt wird. Dieses Verfahren erinnert an AES-CCMP (IEEE 802.11i) aus dem Funkstandard WPA2 (IEEE 802.11i).
Diese Design-Entscheidung wurde getroffen, damit der Entsperrprozess unabhängig von der verwendeten Speichermethode ist, sei es OpenPGP oder Rx. Ohne die vorgestellte Architektur wird das Master-Password als Passwort zur Entschlüsselung genutzt. Denn je nach Speichermethode kann die Entschlüsselung mit einem falschen Passwort einen hohen Ressourcenverbrauch bedeuten, ehe der Entschlüsselungsversuch fehlschlägt. Ein weiterer Vorteil ist, dass eine Änderung des Passworts möglich ist, ohne alle Daten neu verschlüsseln zu müssen. Es muss lediglich ein neuer Hash mit dem neuen Master-Passwort errechnet werden und der ursprüngliche zufällige

Schlüssel erneut mit AES verschlüsselt werden.

4.3.2 Adapter

Wie bereits in Kapitel 4.2.6 angedeutet, sind Adapter zentrales Element des Passwortmanagers. Mittels der um Adapter herum entwickelten Architektur ist es möglich, Daten je nach Adapter unterschiedlich zu verarbeiten und Safeword beliebig zu erweitern. Der Aufbau und die anschließende Nutzung des Adapters wird anhand der Klasse `OpenpgpAdapter` exemplarisch veranschaulicht.
Persistenz-Adapter stellen verschiedene Speicherarten bereit, wie OpenPGP und die integrierte Datenbank Rx in Safeword. Beide Adapter `OpenpgpAdapter` und `RxAdapter` müssen von der Super-Klasse `PersistenceAdapter` erben, die die wichtigen Grundfunktionen bereitstellt, was Zeile 3 von Listing 4.9 zeigt.

```
1  // imports ...

3  export default class OpenpgpAdapter extends PersistenceAdapter {
4    static description = 'OpenPGP'
5    describe = () => 'OpenPGP'

7    create = (props, password) => new Promise((resolve, reject) => {
8      const {filePath} = props

10     const options = {
11       userIds: [{ name:'Jon Smith', email:'jon@example.com' }],
12       curve: 'brainpoolP256r1',
13       passphrase: password
14     }

16     openpgp.generateKey(options).then(function(key) {
17       const privkey = key.privateKeyArmored
18       const pubkey = key.publicKeyArmored
19       const revocationSignature = key.revocationSignature

21       fs.createFile(`${filePath}/private.key`, privkey)
22       .then(() => {
23         fs.createFile(`${filePath}/public.key`, pubkey)
24         .then(() => resolve(true))
25         .catch(() => {
26           store.dispatch(SetupAction.error('Public file could not
                 be written.'))
27           reject(false)
28         })
29       })
30       .catch(() => {
31         store.dispatch(SetupAction.error('Private file could not
               be written.'))
32         reject(false)
33       })
```

```
34      }).catch(error => reject(false))
35    })

37    unlock = async (password, safeword) => {
38      const encryptedPrivateKey = await fs.readFile(`${
            safeword.path}/private.key`)
39      const privKeyObj = (await openpgp.key.readArmored(
            encryptedPrivateKey)).keys[0]
40      return await privKeyObj.decrypt(password)
41    }

43    // ...
44  }
```

Listing 4.9: Adapter am Beispiel OpenPGP

In der spezifischen Adapter-Klasse werden wichtige Funktionen, die für den Betrieb dieses Adapters wichtig sind, überschrieben. Das Listing 4.9 zeigt, dass unter anderem die Funktionen `create` (Zeile 7) und `unlock` (Zeile 37) überschrieben werden, sodass der Adapter initialisiert bzw. entsperrt werden kann. Um das Listing kurz zu halten, wurden JavaScript-Doc-Kommentare entfernt. Die Funktion `create` erhält zwei Parameter `props` und `password`. Dabei enthält `props` ein Objekt, in dem zusätzliche Informationen abgelegt werden können, die der Adapter zum Konfigurieren des Adapters benötigt, wie beispielsweise das Attribut in Zeile 8 `filePath`, das den Pfad zu den Schlüsseldateien enthält. In den Zeilen 10-14 werden in einem Objekt alle Daten gesammelt, die zum Erstellen von Schlüsselmaterial mit OpenPGP notwendig sind, wie zum Beispiel das Passwort oder der Typ der generierten Schlüssel. Schlüssel für OpenPGP können mit `openpgpjs` entweder über RSA (IETF RFC 8017) oder elliptische Kurven generiert werden. Elliptische Kurven erlauben mehr Sicherheit pro Bit eines Schlüssels, als beispielsweise RSA. Gewählt wurde `brainpoolP256r1` (IETF RFC 5639), das sowohl von der Desktop- als auch Androidanwendung unterstützt wird und durch angemessene Generierungszeiten nicht der User-Experience schadet. In Zeile 17 bis 19 werden dem Schlüsselobjekt `key` der private und öffentliche Schlüssel entnommen. Anschließend werden beide Schlüsseldateiem im Datesystem erzeugt (Zeile 21 und 23). Auftretende Fehler werden jeweils in einem `catch`-Block abgefangen und dem Nutzer wird anschließend Auskunft über den auftretenden Fehler gegeben (Zeilen 25-28 und 30-34).
Die Funktion `unlock` (Zeile 37-41) entschlüsselt das hinterlegte Schlüsselmaterial. Dazu werden zwei Parameter übergeben: Das Passwort, um den privaten Schlüssel zu entschlüsseln und die Basiskonfiguration, im Beispiel `safeword` genannt, die den Pfad zu den Schlüsseldateien enthält. Zeile 38 zeigt, dass der Dateiinhalt aus der Datei mit dem Namen `private.key` gelesen wird. Anschließend wird aus dem gelesenen Inhalt durch die Bibliothek ein Schlüsselobjekt erstellt, das das private verschlüssel-

te Schlüsselmaterial enthält. Anschließend wird das Schlüsselmaterial entschlüsselt. Sollte jedoch ein Fehler auftreten, wird dieser automatisch an die aufrufende Funktion weitergegeben.
Das Listing 4.10 zeigt, wie die Funktionen eines Adapters genutzt werden können, ohne einen spezifischen Adapter ansprechen zu müssen.

```
1  /**
2   * Copies a value to the clipboard
3   * @param {String} text Text to copy
4   * @param {Object} password The password of which the copied text
        comes from
5  */
6  const copy = (text, password) => () => {
7    const {dispatch} = store
8    Clipboard.copy(text, {
9      wipe: true,
10     callback: () => Notification.set('Sensitive information has
         been removed from clipboard.'),
11     duration: 20000
12   })
13   dispatch(Message.set('Copied to clipboard.'))

15   const oldPassword = {..password }
16   password.use()
17   store.getState().persistence.adapter.updatePassword(password,
       oldPassword)
18 }
```

Listing 4.10: Nutzung des OpenPGP-Adadpters

Bei dem gezeigten Code der Funktion `copy` handelt es sich um einen Listener, der bei Klick auf einen Knopf den Nutzernamen oder das Passwort in die Zwischenablage kopiert (Zeile 8-13). Safeword bietet die Möglichkeit zuletzt genutzte Passwörter anzuzeigen, weshalb beim Kopieren des Nutzernamen oder Passworts die Zeit der Nutzung protokolliert werden muss. Dies geschieht in Zeile 16 mittels der Funktion `use()` des Models. Der Adapter wird in der letzten Zeile angesprochen. Dazu muss der Adapter dem globalen Store entnommen werden, der immer über eine Instanz eines `PersistenceAdapter` verfügt. Das zuvor modifizierte Objekt des Passwortes muss nun abgespeichert werden. So werden der Funktion `updatePassword` das neue Passwort, dessen Attribute gespeichert werden müssen, und das alte Passwort übergeben, das zur Identifizierung des ursprünglichen Passworts dient.

4.3.3 Tests

Tests sind ein wichtiges Element in dem Entwicklungsprozess. Unter Node.js können Tests nicht nur Code der Business-Logik behandeln, die von der Nutzeroberfläche ab-

gekoppelt ist, sondern auch Tests der Benutzeroberfläche, wie es bei Electron möglich ist. Dazu ist das Paket „Spectron“[25] vonnöten, das Zugriff auf den Chromium-Treiber ermöglicht, sodass auch Komponenten einer Anwendung angesprochen werden können. Das nachfolgende Listing 4.11 zeigt, wie bei der Desktopanwendung getestet werden kann, ob die Anwendung erfolgreich gestartet ist und das Anwendungsfenster anzeigt wird.

```
1  describe('ui', function() {
2    it('opens a window', function() {
3      return app.client.waitUntilWindowLoaded()
4        .getWindowCount().should.eventually.have.at.least(1)
5        .browserWindow.isMinimized().should.eventually.be.false
6        .browserWindow.isVisible().should.eventually.be.true
7        .browserWindow.isFocused().should.eventually.be.true
8        .browserWindow.getBounds().should.eventually.have.property(
             'width')
9        .and.be.above(0)
10       .browserWindow.getBounds().should.eventually.have.property(
             'height')
11       .and.be.above(0)
12   })
13 })
```

Listing 4.11: Test des Programmstarts der Desktopanwendung

Die Funktionen `describe` und `it` in den Zeilen 1 bis 2 werden durch das Test-Framework Mocha bereitgestellt. Damit ist es möglich Tests nach Funktionen bzw. Eigenschaften zu gruppieren. So wird im Beispiel Code die Funktion „ui“ auf die Eigenschaft „opens a window“ geprüft. Durch Chai ist es möglich über das Schlüsselwort `should` eine Bedingungskette zu starten oder fortzuführen. In Zeile 3 und 4 wird auf die Anzahl der Fenster der Anwendung zugegriffen. Mittels `.should.eventually.have.at.least(1)` wird eine Assertion, also eine Behauptung, erstellt, dass mindestens ein Fenster geöffnet wird. Stellt sich diese Behauptung oder eine der folgenden Behauptungen der Kette als falsch heraus, gilt der Test der Eigenschaft als fehlgeschlagen und der Rest der Kette wird nicht mehr ausgeführt. Anschließend wird mit dem nächsten Eigenschaftstest oder gegebenenfalls Funktionstest fortgefahren. Damit der oben genannte Test erfolgreich abschließt, muss das Fenster zudem nicht minimiert, sichtbar und fokussiert sein sowie das Fenster eine Höhe und Breite größer als `0` Pixel aufweisen.
Es ist zudem auch möglich programmatisch mit Bedienelementen zu interagieren, Textfelder zu füllen und Businesslogik zu testen.

Bei den Tests für die mobile Plattform sind viele verschiedene Ansätze möglich.

[25] GitHub: https://github.com/electron/spectron, abgerufen am 27.11.2018

Ein aufstrebendes Test-Framework ist Cavy[26], das als React-Komponente im Komponentenbaum registriert werden muss und die Tests bei jedem Start der Anwendung im Entwicklungsmodus durchführt. Genau wie bei der Desktop-Anwendung mit Spectron, sind auch hier die Eingaben und Bedienungsschritte direkt nachverfolgbar, da die Anwendung im Emulator oder auf dem Testgerät wie gewohnt gestartet wird. Gründe, warum Cavy nicht gewählt wurde, sind unter anderem, dass bei Tests, die auf Komponenten zurückgreifen, Produktionscode und Testcode gleichermaßen in einer Komponente abgelegt werden müssen. Die nicht vorhandene Trennung des Codes führte schlussendlich dazu, dass auf etablierte Test-Pakete zurückgegriffen wurde, die aber leider keine Tests des User-Interface zulassen.
Die Planung sieht daher vor, Tests mittels Enzyme[27] durchzuführen. Enzyme ist ein Test-Renderer für React Native, mit dem es möglich ist, auf die gerenderten Komponenten zuzugreifen. Anders als bei Electron mit Spectron wird hier nicht die Anwendung gestartet, sondern der Code wird ohne Benutzeroberfläche in einer JavaScript-Engine ausgeführt. Ebenfalls wie bei den Tests der Desktopanwendung können auch hier Tests der Businesslogik erstellt werden. Die Tests der Businesslogik lassen sich aber auch durch die gemeinsame Nutzung von Mocha als Test-Framework auf beiden Plattformen testen, sodass auch bei Tests Code-Sharing vorliegt.
Aufgrund der zeitlichen Begrenzung konnte für die Desktop-Plattform nur der Test des erfolgreichen Starts implementiert werden. Auf der mobilen Plattformen wurden Tests vorbereitet und alle notwendigen Pakete bereits referenziert. Um den Test für die mobile Plattform erfolgreich ablaufen zu lassen, sind aber noch weitere Anpassungen des Tests nötig, die eventuell auch Änderungen an der Anwendung bedingen.

4.3.4 Synchronisierung

Die Synchronisierung bietet den Vorteil, dass Passwortdaten auf allen genutzten Geräten stets die aktuelle Version der Passwortdaten vorliegt. Zur Bereitstellung des Speichers eignen sich zahlreiche Webdienste wie Dropbox oder Google Drive, jedoch würden sich auch selbst betriebene „Nextcloud“[28]-Server als Datenendpunkt eignen. Durch den Anspruch eine Open-Source-Anwendung zu erstellen, sind quelloffene Lösungen zu bevorzugen. Aus diesem Grund wurde WebDAV (IETF RFC 4918) als Protokoll gewählt, das mit Nextcloud standardmäßig genutzt werden kann. Zur Nutzung des Protokolls wird die Bibliothek `webdav-client`[29] genutzt, die mit einigen Modifikation auch in der mobilen Anwendung funktioniert.
Analog zu den Persistenz-Adaptern, wurden auch die Synchronisierungs-Adapter im-

[26]GitHub: https://github.com/pixielabs/cavy, abgerufen am 28.11.2018
[27]GitHub https://github.com/airbnb/enzyme, abgerufen am 28.11.2018
[28]GitHub: https://github.com/nextcloud/server, abgerufen am 28.11.2018
[29]GitHub: https://github.com/perry-mitchell/webdav-client, abgerufen am 28.11.2018

plementiert. Der `WebdavAdapter` erbt von der Super-Klasse `SyncAdapter`, die alle benötigten Funktionen bereitstellt und die der spezifische Adapter bei Bedarf überschreibt. Des Weiteren interagiert der Synchronisierungs-Adapter mit dem Persistenz-Adapter, der beispielsweise zu synchronisierende Dateien bereitstellt oder von dem Web-Server bereitgestellte Dateien verarbeitet. Um dies zu gewährleisten überschreiben die Adapter `OpenpgpAdapter` und `RxAdapter` die folgenden vier Funktionen:

- `collectFilesForPushSync`: Sammelt Passwort-Dateien, die auf dem Server abgelegt werden sollen
- `collectAdditionalFilesForPushSync`: Sammelt zusätzliche Dateien, wie beispielsweise Schlüsselmaterial, das ebenfalls auf dem Server abgelegt werden soll
- `processFilesOfPullSync`: Verarbeitet Dateien, die vom Server zur Verfügung gestellt wurden
- `processAdditionalFilesOfPullSync`: Verarbeitet zusätzliche Dateien eines Adapters, die vom Server bezogen werden

Dadurch, dass die Daten des OpenPGP-Adapters auf dem Dateisystem abgelegt werden, können diese einfach auf dem Server hinterlegt und abgerufen werden. Zu beachten ist, dass es Kollisionen geben kann, wenn ein Datensatz von einem Gerät verändert wurde und diese Änderung nicht auf das andere Gerät übertragen wurde, die aber nur auftritt wenn an beiden Geräten gleichzeitig auf die Daten zugegriffen wird. Eine Synchronisierung wird immer, sofern erstmalig eingerichtet, bei der Anmeldung eines Nutzers durchgeführt. Der Nutzer hat darüber hinaus die Möglichkeit eine Synchronisierung manuell im Einstellungsmenü zu starten. Sollte es zu einem Konflikt kommen, wird der Nutzer über diesen Umstand in der Benutzeroberfläche benachrichtigt und die Änderungen müssen vorerst zurückgenommen werden. Anschließend müssen die Daten manuell synchronisiert werden, wonach schließlich die Änderung der Daten erneut durchgeführt werden kann.
Die Daten der Rx-Datenbank werden synchronisiert indem die Funktion `dump` einer Datenbank genutzt wird, die die gesamte Datenbankinstanz als JSON-Objekt zur Verfügung stellt. Diese wird in einer Datei auf dem Server abgelegt. Jedoch muss beachtet werden, dass jede Datenbankinstanz einen ID erhält, die ebenfalls in diesem exportierten Datensatz enthalten ist. Folglich kann eine Rx-Datenbank nicht synchronisiert werden, da bei einer Synchronisierung immer eine neue Datenbank-Instanz erzeugt werden muss und bei mehreren Geräten immer mehrere Datensätze mit unterschiedlichen IDs existieren. Lediglich Export und Import der Daten von der Datenbank müssen für den RxAdapter angepasst werden, sodass die Synchronisation im vollen Umfang genutzt werden kann.

Bei der Implementierung der notwendigen Funktionen des WebDAV-Clients wurde auf Docker[30] und WebDAV-Server-Images zurückgegriffen, um einen möglichst einfachen und wartungsarmen Betrieb des Servers zu ermöglichen. Leider sind diese Server-Images schlecht gewartet und fehlerhaft, sodass fünf verschiedene Images ausprobiert werden mussten. Die Fehler reichten von fehlender Möglichkeit, Daten auf den Server zu schreiben, über falsche Dateisystemrechte hin zu Images, bei denen der Server bei einem Aufruf sofort einen 500er Fehler erzeugt. Ein Image, das für die Entwicklung genutzt wurde, erwies sich nach einigen Tagen leider als untauglich, da der WebDAV-Server nur noch sehr langsam reagierte und die initiale Synchronisierung bis zu 15 Minuten in Anspruch nahm. Aus diesem Grund und der zeitlichen Begrenzung der Thesis wurde die Entwicklung der Synchronisierung an dieser Stelle beendet. Bei der Synchronisierung der Daten des OpenPGP-Adapters fehlt noch das Abgleichen der Schlüssel, die zum Ent- und Verschlüsseln genutzt werden. Zudem erwies sich die Ermittlung eines potentiellen Konfliktes auf dem WebDAV-Server als nicht vollständig korrekt.

4.3.5 Probleme

Die Entwicklung von Safeword wurde von vielen Problemen begleitet, die die Entwicklung nicht unwesentlich verzögert haben. Schon zu Beginn der Implementierung fiel auf, dass es eine Vielzahl von veralteten und fehlerhaften Paketen gibt. Somit muss neben der Paketsuche noch ein Funktionstest des Pakets durchgeführt werden, ob alle relevanten Schnittstellen des Pakets auch funktional sind. Beispielsweise konnte das Paket `react-native-secure-key-store`, das eine Schnittstelle zum Key Store des Betriebssystems bereitstellt, aufgrund des alten nativen Codes nicht mehr kompiliert werden, sodass der Passwortmanager seinen Dienst verweigerte. Für Verwunderung sorgte auch das Paket `react-native-aes`[31], das zwar eine Verschlüsselung, aber keine Entschlüsselung der Daten ermöglicht, da die Funktion nicht vollständig implementiert wurde und es somit zu Fehlern kam. Des Weiteren stellte sich nach Einsicht des Quellcodes heraus, dass die Schnittstellen der Anwendung nicht mit der Dokumentation übereinstimmen. Die Funktion zur Verschlüsselung nimmt entgegen der Dokumentation nicht Base64-Strings, sondern Hex-Strings, entgegen. Wird dies nicht beachtet, kommt es zu Fehlern bei der Verschlüsselung.
Ein Problem bei der Paketwahl zeichnete sich auch bei der Wahl des Paketes zur Speicherung in einer Datenbank ab. Zu Beginn wurde für diesen Zweck die Bibliothek `SQLCipher`[32] gewählt, für die es auch Portierungen in JavaScript gibt. Unter Node.js

[30] GitHub: https://www.docker.com/, abgerufen am 28.11.2018
[31] GitHub: https://github.com/tectiv3/react-native-aes, abgerufen am 28.11.2018
[32] GitHub: https://github.com/sqlcipher/android-database-sqlcipher, abgerufen am 28.11.2018

war es mittels `node-sqlite3` leider nicht möglich SQLCipher für Electron zu kompilieren, da der Prozess immer mit einem Fehler beendet wurde. Die Verschlüsselung kann zudem auch nur nach dem Kauf einer Lizenz durchgeführt werden. Als Reaktion darauf wurde, entgegen der ursprünglichen Planung, OpenPGP als primäre Verschlüsselungsmethode gewählt. Eine datenbankbasierte Speicherung sollte dennoch als Alternative zu OpenPGP implementiert werden. Dazu wurde bei der Recherche vor allem Wert auf Cross-Platform-Kompatibilität gelegt, sodass nicht erneut Pakete kompiliert werden müssen, was Fehlerpotential bieten würde. Als vielversprechender Kandidat stellte sich früh die Datenbank `Realm`[33] heraus. Bei der Inbetriebnahme stellte sich heraus, dass die Datenbank, entgegen den Behauptungen der Webseite, nicht mit aktuellen Version 10 von Node.js kompilieren lässt. Lediglich die Long-Term-Support-Version 8.X wird unter React Native unterstützt. Weshalb erneut eine andere Bibliothek gesucht werden musste. Aufgrund der Probleme mit Bibliotheken, für die Quellcode kompiliert werden muss, sollte eine Bibliothek gefunden werden die rein in JavaScript betrieben werden kann und keine nativen Abhängigkeiten aufweist. Schlussendlich wurde die Bibliothek `RxDB`[34] gewählt, die verschiedene Speichermöglichkeiten über Adapter unterstützt und daher auch in Electron und React-Native funktionsfähig ist.

Zuletzt bereitet React Native einige Probleme, bedingt durch den Umstand, dass Code-Sharing betrieben wird und auch gemeinsame Paketabhängigkeiten bestehen. Der Bundler, der die JavaScript-Dateien für React Native traversiert und zusammenfasst, durchsucht die betrachteten Dateien nach den Schlüsselworten `require` und `import`, um diesen Importbefehlen zu folgen. Mittels dieser Methode werden nur genutzte Abhängigkeiten in dem Bundle abgelegt, was als „Tree Shaking“ bekannt ist. Bei dem Tree Shaking werden `if/else`-Strukturen, die Pakete in Abhängigkeit von der Plattform inkludieren, ignoriert, sodass für die mobile Anwendungen auch Pakete importiert werden, die aber nicht existieren, wie beispielsweise `fs` oder `os`. Der Bundler bricht seine Arbeit an dieser Stelle ab, da die Abhängigkeiten nicht vollständig aufgelöst werden können. Nach langer Recherche konnte ermittelt werden, dass die Importe spezieller Pakete ignoriert werden können, indem diese in der Datei `package.json` deaktiviert werden.

Zudem tritt der Fehler auf, dass das Paket `openpgpjs` unter React Native nicht genutzt werden kann, wenn die Desktop- und Smartphoneanwendung die gleiche Version nutzen. Aus diesem Grund wurde das Paket `react-native-openpgpjs` erstellt, das immer eine Version minimal ältere Version von `openpgpjs` referenziert, als die Desktopanwendung nutzt. Im Verlauf der Entwicklung kam es bei einigen Versionen der Bibliothek zu Inkompatibilitäten mit der mobilen Plattform, die das erfolgreiche

[33] GitHub: https://github.com/realm/realm-js, abgerufen am 28.11.2018
[34] GitHub: https://github.com/pubkey/rxdb, abgerufen am 28.11.2018

Ausführen der Anwendung verhinderten. Die Bibliothek in den Versionen `4.{0,1}.X` behindern den Start jedoch nicht.

4.4 Betrieb

Dieses Kapitel behandelt wichtige technische Eigenschaften des Passwortmanagers, die bei dem Betrieb und der Weiterentwicklung beachtet werden müssen.

4.4.1 Verzeichnisse

In dem Quellcode-Ordner sind zwei große Frameworks vereint. Zum einen *Electron* für die Desktop-Plattformen und *React Native* für die mobilen Plattformen, hier im speziellen für Android. Jede Applikation, ob für Android oder Desktop, benötigt einen sogenannten „Entrypoint“. Dieser bildet den Startpunkt und initialisiert die Applikation. Dies sind die Dateien `index.desktop.js` sowie `index.native.js` für Desktop- respektive mobile Betriebssysteme. Diese binden dann Komponenten aus dem `src`-Ordner ein.

Der `src`-Ordner im Wurzelverzeichnis enthält zahlreiche Ordner, deren Zweck folgende Übersicht erläutert.

- `adapters`: Alle Adapter, die in Safeword genutzt werden
- `components`: Komponenten aus dem User-Interface, die wiederverwendet werden können oder aus Routen ausgelagert wurden
- `configuration`: Basiskonfiguration der Anwendung
- `globals`: Globale Variablen wie Plattform oder Speicherort des Masterkeys
- `model`: Modelle wie Passwort oder Gruppe
- `modules`: Module, die eine Abstraktionsschicht zu plattformspezifischem Code hinzufügen
- `routes`: Komponenten aus dem User-Interface, die einmalig verwendet werden und den Rahmen der Anwendung beinhalten
- `service`: Services, wie die Navigation
- `settings`: Einstellungen der Anwendung
- `store`: Redux-Store, der global Daten verwaltet
- `theme`: Enthält Farben, die im User-Interface genutzt werden

- `utils`: Hilfsfunktionen, die plattformunabhängig häufig durchgeführte Aufgaben übernehmen, wie das Suchen eines Objektes in einem Array

Sowohl bei dem Ordner `components`, `routes` als auch `test`, der sich im Hauptverzeichnis befindet, wird einer weitere Unterscheidung in `desktop` und `native` unternommen. Zusätzlich ist in diesen Ordnern, ausgenommen `routes`, das Verzeichnis `core` vorhanden. Demnach können plattformspezifische Komponenten in ihrem jeweiligen Ordner der Plattform abgelegt werden. Der gemeinsam nutzbare Code wird folglich im Ordner `core` abgelegt.
Zudem sind die Ordner `android` und `ios` relevant, denn diese beinhalten die nativen Module der Anwendung. Die Strukturierung dieser Ordner ist exakt wie bei einer nativen Anwendung angelegt.
Ebenfalls im Hauptverzeichnis befinden sich zahlreiche Konfigurationsdateien, die in Kapitel 4.4.3 näher erläutert werden.

4.4.2 Paketverwaltung

In der Anwendungsentwicklung haben sich Paketmanager etabliert, die alle Abhängigkeiten einer Applikation verwalten und auch Upgrades auf neue Versionen erleichtern. Im Node.js-Segment sind NPM (Node Package Manager) und Yarn die populärsten Paketverwalter. In dieser Thesis wird Yarn aufgrund der einfacheren Bedienung des Command Line Interfaces verwendet.
Zentrales Element ist bei diesen Verwaltern die Datei `package.json`. Diese enthält nicht nur Informationen über die Anwendung, wie Name, Autor, Versionsnummer oder Lizenz, sondern auch eine Liste mit Abhängigkeiten der Anwendung. Von Vorteil ist ein Paketverwalter, wenn verschiedene Pakete eine gemeinsame Abhängigkeit haben, aber die geforderte Versionen sich unterscheiden, da dieser dann automatisch die am besten passende Version auswählt oder gar mehrere installiert, falls unterschiedliche Version benötigt werden.

4.4.3 Konfiguration der Werkzeuge

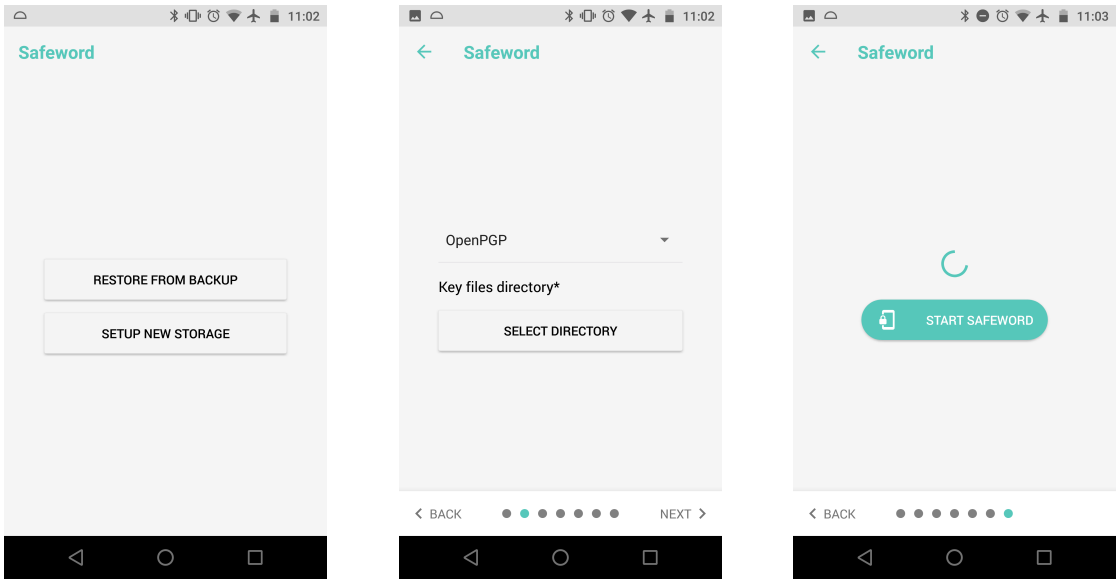
Electron und *React Native* werden standardmäßig mit zahlreichen Tools ausgeliefert, die die Entwicklung mit den Frameworks erleichtern. Zentrales Element ist dabei bei beiden Frameworks *Babel*. Wie bei jeder anderen Sprache, gibt es auch hier verschiedene Versionen der Sprache. Der Prozess der Implementierung der neuen Versionen in alle Browser ist aber langwierig, oder findet bei alten Browserversionen gar nicht erst statt. Babel transpiliert den JavaScript-Code von einer neueren Version in das ältere EcmaScript 6. Dadurch kann der Entwickler schon früher neue Features der Sprache nutzen und genießt einen größeren Browser-Support. Darüber hinaus kann

Babel alle JavaScript-Dateien zu einer Datei bündeln und anschließend minimieren. Das Verhalten und auch weitere Plugins können in der Datei `.babelrc` konfiguriert werden. In Safeword werden vor allem Plugins genutzt, die neue Sprachfunktionen in EcmaScript schon jetzt verfügbar machen.
Damit die Editoren der Entwickler sich immer gleich Verhalten, zum Beispiel bei der Einrückung oder Art des Umbruchs, wurde eine `.editorconfig` angelegt. EditorConfig definiert eine offene Schnittstelle, mit der sich verschiedene Stile und Einstellungen für den Editor festlegen lassen. Viele Editoren implementieren diese Schnittstelle und berücksichtigen die Datei `.editorconfig`, wenn diese im Hauptverzeichnis vorhanden ist. Dadurch ist es möglich, dass alle am Projekt teilnehmenden Entwickler den gleichen Zeichensatz, wie zum Beispiel UTF-8, verwenden und auch die gleiche Art von Umbrüchen, beispielsweise im UNIX-Stil, benutzen.
Ebenfalls enthalten ist ein Linter, namentlich *ESLint*. Ein Linter überprüft anhand vorhandener Metriken, ob Programmierstile eingehalten werden. Es werden dabei die Standard Regeln von ESLint und erweiterte React-Regeln genutzt. ESLint wird über die Datei `.eslint.yml` konfiguriert. In dieser Datei lässt sich dann beispielsweise einstellen, welche Art von Anführungszeichen für Strings genutzt werden soll, wie tief die Einrückung sein soll oder ob sich geschwungene Klammern von Blöcken in einer neuen Zeile oder am Ende der Zeile befinden sollen.

4.5 Bedienung

Dieses Kapitel gibt eine kleine Einführung in die Bedienung des Programms. Aus platztechnischen Gründen werden nur die wichtigsten Bedienungsschritte behandelt, zudem stammen alle Bildschirmfotos von der mobilen Plattform.

Die Abbildung 4.6 zeigt einige Schritte, die bei der Einrichtung von Safeword durchlaufen werden. Zu Beginn hat der Nutzer, wie in Abbildung 4.6(a) ersichtlich, die Möglichkeit einen neuen Datenspeicher anzulegen, oder aus einem existierendem Datensatz ein Backup herzustellen. Die Funktion der Wiederherstellung aus einem Backup ist für zukünftige Erweiterungen angedacht und bietet in dem entwickelten Prototyp keine Funktion. Die Auswahl des neuen Speichers führt zum ersten Schritt der Einrichtung, der über die folgenden Schritte informiert. Nach einem Klick auf `Next` wird der Nutzer zu Auswahl der Speicherart geführt, wie Abbildung 4.6(b) zeigt. Ausgewählt werden kann zum einen „OpenPGP“ oder „Rx“. Fällt die Auswahl auf `OpenPGP`, muss mittels des Knopfes noch der Ordner ausgewählt werden, in dem die zu erstellenden Schlüssel gespeichert werden sollen. Nach weiteren Einrichtungsschritten, wie dem Profil und der Vergabe des Master-Passworts, gelangt der Nutzer zum Abschlussbildschirm des Einrichtungsprozesses, wie in Abbildung 4.6(c)

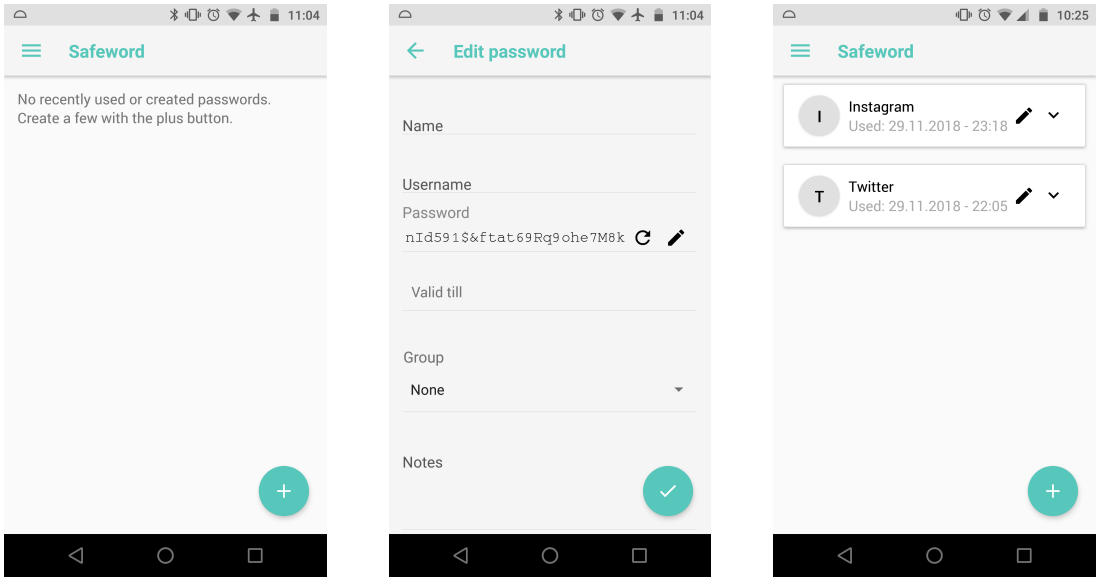


(a) Beginn der Einrichtung (b) Auswahl Speicherart (c) Ende der Einrichtung

Abbildung 4.6: Einrichtung der Anwendung

abgebildet.

Abbildung 4.7 zeigt die Ansichten, die in Safeword hauptsächlich für das Speichern und Lesen der Passwortdaten genutzt werden. Nach der Einrichtung sind keine Passwörter oder Gruppen hinterlegt, sodass der Nutzer von dem Hauptbildschirm mit einem Hinweistext begrüßt wird, wie in Abbildung 4.7(a) dargestellt. Über den „Floating Action Button“, der türkisfarbene Knopf am unteren rechten Rand, können sowohl neue Passwörter als auch Gruppen erstellt werden. Wünscht der Nutzer ein neues Passwort zu erstellen, wird dieser zu dem entsprechenden Bildschirm weitergeleitet und kann die Daten in die dafür vorgesehenen Felder eintragen, dargestellt in Abbildung 4.7(b). Über den Floating Action Button, der sich an gewohnter Stelle befindet, speichert der Nutzer das neue Passwort ab und gelangt zum vorherigen Bildschirm zurück. Abbildung 4.7(c) zeigt die Ansicht, in der der Nutzer nun alle bereits erstellten Passwörter vorfindet. Über den seitlichen App-Drawer, der in dem Hauptbildschirm zur Verfügung steht, können weitere Funktionen erreicht werden, wie die Filterung nach allen zuletzt genutzten Passwörtern sowie die Darstellung aller Passwörter und Gruppen. Des Weiteren besteht Zugriff auf den Passwort-Generator, die Gruppenverwaltung, die Einstellungen und die Lizenz.



(a) Hauptbildschirm

(b) Speichern eines Passworts

(c) Hauptbildschirm mit Passwörtern

Abbildung 4.7: Speichern und Abrufen eines Passworts

Kapitel 5

Evaluierung des Prototyps

Dieses Kapitel behandelt die kritische Betrachtung des entwickelten Passwortmanagers. Dazu werden die Ergebnisse des Code-Sharings zusammengetragen. Dies geschieht im Vergleich zu den Passwortmanagern, die in Kapitel 3.2 bereits verglichen wurden. Anschließend werden die möglichen Erweiterungen und die Architekturentscheidung erörtert.

5.1 Ergebnisse des Code-Sharings

In Kapitel 3.2 wurden Passwortmanager vorgestellt und unter anderem auch auf Code-Sharing untersucht. Dabei stach als einziger Passwortmanager Padlock heraus, da nur hier Code-Sharing konsequent für alle Plattformen umgesetzt wurde. Die Entwickler entschieden sich für ähnliche Tools, die auch bei Safeword genutzt werden, was einen Vergleich des Managers mit Safeword sehr interessant macht. Wichtig ist hier hervorzuheben, dass Safeword für die mobile Anwendung, aufgrund von Geschwindigkeitsvorteilen, React Native gewählt wurde, während Padlock für die mobile Anwendung Cordova nutzt. Im Folgenden wird die Umsetzung des Code-Sharing hinsichtlich mehrerer Kriterien, wie Kryptographie oder Wartung, analysiert und mit der Umsetzung von Padlock verglichen.

5.1.1 Kryptografie

Bei Code-Sharing kann die Kryptografie in zwei Gruppen eingeteilt werden. Dies ist zum einen der Code, der kryptografische Verfahren nutzt und zum anderen der Code, der die kryptografischen Verfahren implementiert.
In Safeword wird vor allem Code-Sharing zur Nutzung kryptografischer Verfahren betrieben. Ermöglicht wird dies durch Module, die Schnittstellen für alle Plattformen zur Verfügung stellen, wodurch im Core-Code keine Rücksicht auf plattformspezifi-

sche Unterschiede genommen werden muss.
Code-Sharing bei der Implementierung kryptografischer Verfahren zu betreiben, bedeutet, dass Pakete auf allen Plattformen ohne Anpassung genutzt werden können. Das ist bei der Kryptografie in Safeword größtenteils nicht der Fall, was aber technische Gründe hat. Alle Bibliotheken für kryptografische Verfahren basieren auf der Crypto-API, die ein Browser und auch Electron über Node.js für Verschlüsselungszwecke und der Generierung kryptografisch sicheren Zufalls bereitstellt, aber in React Native nicht zur Verfügung steht. Die Crypto-API ist, durch die darunterliegende Implementierung in C, schneller als Implementierungen in reinem JavaScript. Bei React Native wird auf die Implementierung der API in purem JavaScript aufgrund von Performance weitestgehend verzichtet und für höchstmögliche Performance als native Module implementiert, die sich den schnellen Schnittstellen der Betriebssysteme bedienen, wie für AES, SHA2 und PBKDFv2 geschehen. Um etwaige Geschwindigkeitsunterschiede im Vergleich zu der Desktop-Anwendung festzustellen, wird in Safeword OpenPGP durch eine reine JavaScript-Bibliothek realisiert, die für die mobile Plattform in das Paket `react-native-openpgpjs` verkapselt wurde, welches noch einige weitere Anpassungen mitbringt. Im Vergleich zur Desktop-Anwendung ist die Geschwindigkeit der mobilen Anwendung bei der Schlüsselerstellung erheblich geringer aber noch ausreichend, was auf die fehlende Beschleunigung durch native Schnittstellen zurückzuführen ist. Aus Gründen der Performance und damit einhergehenden User-Experience sollte aber auf ein natives Modul umgeschwenkt werden.
Padlock betreibt, analog zu Safeword, Code-Sharing bei dem Zugriff auf kryptografische Verfahren. Dies liegt darin begründet, dass, durch Cordova und die enthaltene WebView, einige Schnittstellen zur Verfügung stehen, die sich mit denen von Electron decken. Dadurch lassen sich sehr viele browserspezifische Pakete sowohl unter Electron als auch unter Cordova nutzen. So haben sich die Entwickler dazu entschieden, die Bibliothek `sjcl`[1] zu nutzen, die die Verschlüsselung in nativem JavaScript für alle Plattformen zur Verfügung stellt. Dadurch wird im Vergleich zu Safeword auch Code-Sharing bei der Implementierung der kryptografischen Verfahren realisiert, mit dem Nachteil einer geringeren Geschwindigkeit.
Hinsichtlich der Sicherheit entsteht bei Padlock ein wichtiger Vorteil, denn die kryptografischen Verfahren entstammen nur einem Paket. Sollte eine Schwachstelle bei der Implementierung einer dieser Verfahren publik werden, kann diese direkt für alle Plattformen behoben werden. Da in Safeword aber bei den kryptografischen Verfahren getrennte Bibliotheken eingesetzt werden, kann es jedoch sein, dass Schwachstellen unterschiedlich schnell behoben werden. Daher ist es besonders wichtig, nur gut gewartete Pakete für sicherheitskritische Anwendungen auszuwählen, um die Zeit, in

[1] GitHub: https://github.com/bitwiseshiftleft/sjcl, abgerufen am 03.12.2018

der die Software verwundbar ist, zu minimieren.
Somit kann zusammengefasst werden, dass Code-Sharing bei der Implementierung kryptografischer Verfahren großen Einfluss auf die Performance hat, sodass darauf lieber verzichtet werden sollte. Zwar ist der Anteil des Code-Sharings dann nicht so hoch, dennoch kann eine ähnlich saubere Struktur bei der Nutzung von kryptografischen Verfahren wie in Padlock auch in Safeword durch Module erreicht werden. Code-Sharing bei der Nutzung dieser Verfahren ist aber uneingeschränkt empfehlenswert und erleichtert die Wartung der Anwendung ungemein.

5.1.2 Bedienoberfläche

Dadurch, dass React Native keine Webseiten in einer WebView darstellt, sondern native Oberflächenkomponenten nutzt, die nicht kompatibel zu Electron sind, kann in Safeword kein Code-Sharing bei der Bedienoberfläche durchgeführt werden. Jedoch wird Code-Sharing für Quellcode verwendet, der von UI-Komponenten genutzt wird, die aber keinen Zugriff auf den State oder die View-Komponenten benötigen. In Safeword werden die Komponenten daher in jeweils dafür vorgesehene Ordner abgelegt. Bei der Gestaltung der Oberfläche muss beachtet werden, dass diese sowohl in Electron als auch React Native umsetzbar ist. Dieser Umstand erhöht die Komplexität und verzögert die Plaungs- und Implementierungphase. Mittels React Native können durch native Oberflächenkomponenten jedoch auch Anwendungen für Einsteigersmartphones erstellt werden, die trotz schwacher Hardware flüssige Animationen darstellen können.
Durch Cordova kann Padlock hier das volle Potential des Code-Sharings ausschöpfen. Sowohl Electron als auch Cordova stellen WebViews da, die die Entwickler von Padlock sowohl auf der Mobile- als auch Desktop-App verwenden. Dadurch werden die Komponenten nur an einer Stelle erstellt und können sich dank CSS und Media-Queries an unterschiedliche Bildschirmgrößen anpassen. Dies ist ein enormer Vorteil, denn die Anwendung sieht auf allen Plattform ohne weitere Anpassung immer gleich aus, sodass eine Umgewöhnung bei einem Gerätewechsel entfällt. Jedoch stellt die WebView einen erheblichen Nachteil dar, denn diese ist ressourcenintensiv und zeigt vor allem auf leistungsschwachen Geräten selten flüssige Animationen. Die zur Darstellung von Webseiten benötigten Bausteine wie Parser oder Interpreter sorgen für einen erheblichen größeren Fußabdruck, sodass dies je nach Gerät für Einbußen bei der User-Experience sorgen kann.
Wie bereits in vorherigen Kapiteln erwähnt, soll Code-Sharing nicht das primäre Ziel bei der Entwicklung darstellen, stattdessen gilt es einen Kompromiss für den Nutzer und den Entwickler zu finden. Zwar genießt der Entwickler von Padlock mit Cordova die Vorteile des Code-Sharings und damit die einfache Wartung sowie Entwicklung

der Oberfläche, jedoch lässt sich die Anwendung nicht so schnell wie eine native App bedienen, was schlussendlich für den Nutzer den größeren Ausschlag gibt. Deshalb wurde mit React Native ein guter Kompromiss gefunden, der zwar bei der Bedienoberfläche kein Code-Sharing bietet, aber bei allen anderen Anwendungsgebieten des Code-Sharings mit Cordova gleichziehen kann und darüber hinaus flüssige Animationen bietet.

5.1.3 Tests

Zwar wurden für Safeword aus Zeitgründen keine tiefgreifenden Tests implementiert, jedoch kann hier auf Wissen aus der Recherche und Konzeption zurückgegriffen werden.
Safeword bietet die Möglichkeit Tests für die Oberfläche und den Core-Code zu verfassen, zu dem auch die Adapter, Module und der Store zählen. Die Tests des Core-Codes lassen sich ebenfalls auf beiden Plattformen nutzen, was aber von den gewählten Test-Frameworks abhängt, die idealerweise für alle Plattform gleich sein sollen, wie in Safeword mit Mocha geschehen. Dies verringert effektiv den Aufwand Tests zu verfassen, da diese nur einmal implementiert werden müssen. Der plattformspezifische Code verlangt weiterhin spezifische Tests, da UI-Komponenten beispielsweise nur in ihrer eigenen Umgebung, in dem alle notwendigen Pakete verfügbar sind, getestet werden können.
Für Padlock haben die Entwickler Tests für den Core-Code verfasst, ließen aber Potential ungenutzt, da keinerlei Tests der Oberfläche oder der UI-Komponenten implementiert wurden.
Hier sticht Safeword positiv heraus, indem sowohl Core-Tests als auch UI-Tests möglich sind. Werden die Möglichkeiten des Code-Sharing von Test-Code betrachtet, so liegen beide Passwortmanager gleich auf, da nur Kernkomponenten, die nicht die Oberfläche betreffen, gemeinsam getestet werden können.

5.1.4 Implementierung und Wartung

Dieses Kapitel befasst sich mit den allgemeinen Ergebnissen zum Code-Sharing, die sich in der Implementierung und Wartung von Safeword ergeben haben. Diese Erkenntnisse lassen sich auch auf andere Anwendungen, die Code-Sharing betreiben, übertragen. Bereits in Kapitel 3.2 wurden in der Detailvorstellung der Passwortmanagers und dem anschließenden Vergleich Vorteile angedeutet. Beispielsweise müsse weniger Code geschrieben werden und sowohl die Implementierung als auch die anschließende Wartung würden sich vereinfachen.
Die Annahme, dass weniger Code geschrieben muss, kann zweifelsfrei bestätigt wer-

den, denn sowohl die Adapter-Strukturen als auch der Core-Code der Komponenten sind plattformunabhängig und haben den Aufwand stark verringert, der mit ungefähr 4000 Zeilen Code im Kern beziffert werden kann. Dadurch, dass der Code von verschiedenen Plattformen genutzt wird, und auch durch die gewählte Architektur modular gehalten werden muss, wird der Entwickler zur guten Strukturierung des Codes gezwungen. Die gewählte Architektur erleichtert die Entwicklung mit dem Core-Code, denn sollte es Plattformabhängigkeiten geben, werden diese durch Module verkapselt, die eine weitere Abstraktionsschicht und somit eine eigene Schnittstelle definieren.
Eine weitere Metrik von Code-Qualität, die in der statischen Code-Analyse genutzt wird, ist auch die Komplexität, die dem Quellcode inne wohnt. Diese beinhaltet beispielsweise Verzweigungen mittels `if/else`, die sich über einen Großteil des Codes erstrecken, beispielsweise um Code je nach genutzter Plattform auszuführen. Durch Code-Sharing und die in Safeword eingeführten Module können diese Verzweigungen ausgelagert werden und treten nur noch einmal zur Plattformidentifizierung im Modul auf. Als weitere Metrik der Qualität gilt auch die Länge des Funktionskörpers, bei der Code-Sharing unterstützt, indem Code aus Funktionen ausgelagert wird.
Bei der Entwicklung von Safeword fiel zudem auf, dass weitere Funktionen zuerst für eine Plattform implementiert wurden und anschließend die Funktion auf der verbleibenden Plattform komplettiert wurde. Dies erwies sich als sehr komfortabel, denn anschließend mussten nur noch die fehlenden View-Komponenten implementiert und an den vorhandenen Core-Code angeschlossen werden. Die Effektivität des Code-Sharings kann noch verbessert werden, indem, statt auf Module zurückzugreifen, Pakete genutzt werden, die ohne Anpassungen auf allen Plattformen genutzt werden können. Somit wird eine Abstraktionsschicht vermieden und die Komplexität verringert.
Im Zuge der Beseitigung von Fehlern fiel Code-Sharing vor allem durch den „Single point of failure“ auf, denn dadurch konnten Fehler, die beide Plattformen betreffen, einfach behoben werden. Wenn beispielsweise ein Passwort editiert wurde, wurde der Store nicht aktualisiert, obwohl der persistente Speicher die Änderungen enthielt. Der Fehler war schnell gefunden, denn dadurch, dass der Fehler auf beiden Plattformen auftritt, ist die Wahrscheinlichkeit sehr hoch, den Fehler im Core-Code zu finden. Bei beiden Persistenz-Adaptern konnte das Problem schnell behoben werden und die Verbesserung konnte direkt an alle Plattformen verteilt werden.
Nachteilig wirkt sich der erhöhte Aufwand auf die initiale Implementierung aus, denn es müssen alle Plattformen bedacht und eine adäquate Schnittstelle definiert werden. Als Folge der höheren Komplexität und der Abhängigkeiten der Plattformen voneinander, kann es dazu kommen, dass sich Schnittstellen mehrfach ändern. Diese Schnittstellenänderungen müssen dann folglich auch im plattformspezifischen Code

eingepflegt werden, was an dieser Stelle größeren Wartungsaufwand bedeutet. Das Code-Sharing verleitet dazu, für die Entwicklung Pakete auszuwählen, die direkt mit Electron und React Native genutzt werden können, da so effektiv plattformspezifischer Code für die Erstellung eines Moduls entfällt. Dies ist prinzipiell eine gute Idee, schränkt aber den Kreis der wählbaren Pakete stark ein und kann auch zu Problemen auf einer Plattform führen, wenn nicht alle Funktionen unterstützt werden, oder gar Fehler auftreten. Daher sollten plattformspezifische Pakete und auch selbst verfasste native Module ebenfalls evaluiert werden, um anschließend über ein Modul eine gemeinsame Schnittstelle zu schaffen. Es gilt zu beachten, dass Code-Sharing keine Code-Qualität garantiert. Die Metriken der Code-Qualität können auch von Quelltexten erfüllt werden, die nicht auf Code-Sharing setzen. Umgekehrt können ebenso Quelltexte, die auf Code-Sharing basieren, die Metriken nicht erfüllen. Jedoch kann Code-Sharing dazu beitragen, den Code besser zu strukturieren, sodass die Metriken mit weniger Aufwand erreicht werden können.

5.2 Erweiterungen

Safeword bietet schon jetzt viele Funktionen, jedoch sind auch einige Erweiterungen denkbar, sowohl in funktioneller als auch technischer Hinsicht. Wünschenswert wären weitere Einstellmöglichkeiten für Gruppen, wie beispielsweise eine Farbe oder ein Logo. Damit könnten Passwörter, die einer Gruppe angehören noch schneller identifiziert werden. Passwörter können zudem nur nach Nutzungsdatum oder Gruppe gefiltert werden. Sinnvoll wäre eine Suchfunktion, die zum Suchbegriff passende Passwörter anzeigt. Die Architektur von Safeword ist bereits darauf ausgelegt, dass mehrere andere Faktoren zur Authentifikation außer dem Passwort genutzt werden können, wie NFC-Tags oder Fingerabdrucksensoren. Erfreulich wäre eine Implementierung dieser Adapter inklusive der entsprechenden Integration in die Benutzeroberfläche. Um die Transparenz für den Nutzer im Synchronisierungsprozess zu erhöhen, wäre ein Protokoll denkbar, das klar verständlich auflistet, wann welche Datei synchronisiert wurde und ob es dabei zu einem Fehler gekommen ist. Eine logische Erweiterung dessen ist eine Synchronisierungsschlange, in der alle zu synchronisierenden Dateien eingefügt werden können, die dann so bald wie möglich abgearbeitet werden.
Die Erweiterungen technischer Art könnten zum Beispiel weitere Speicherarten sein, wie in Text-Dateien gespeicherte JSON-Objekte, deren sensitive Daten mittels AES verschlüsselt sind. Es wurde bereits erwähnt, dass Argon2 ein neuer empfohlener Algorithmus zur Schlüsselableitung ist, der in Safeword aber aufgrund von Mangel an nutzbaren Paketen nicht ausgewählt wurde. Inzwischen sind schon Bibliotheken für Node.js und Java entwickelt worden, die Argon2 nicht selber implementieren, son-

dern ein Binding zur Bibliothek der Referenzimplementierung herstellen, die in C verfasst wurde. Beispiele für gut gewartete Bibliotheken jeweils einer Plattform sind `node-argon2`[2] bzw. `argon2-jvm`[3].
Die Konformität von Safeword mit aktuellen Standards für Passwortmanager ist jedoch wichtiger als das Nachpflegen von Funktionen. Das BSI definiert im Grundschutzkatalog (Bundesamt für Sicherheit in der Informationstechnik, 2016) in den Artikeln M2.11, M2.22 und M4.306 wichtige Kriterien, die ein Passwortmanager zwingend erfüllen muss, sodass sensitive Daten in diesem sicher abgelegt werden können. Zur Überprüfung des Managers anhand dieser und eventuell weiterer Metriken z.B. von NIST wird dringend geraten, da so potentielle Schwachstellen der Implementierung offen gelegt werden und diese vor dem Produktivbetrieb behoben werden können.

5.3 Architektur-Entscheidung

Während der Recherche wurde ein wissenschaftliches Paper entdeckt, das eine neue Architektur namens „Kamouflage“ vorstellt, die den Diebstahl von Passwörtern aus einem Passwortmanager durch Offline-Angriffe erschweren soll (Bojinov u. a., 2010). Offline-Angriffe sind Angriffe, die auf einen lokal verfügbaren Datensatz oder eine lokal verfügbare Anwendung ausgeführt werden, um an verschlüsselte Daten zu gelangen oder diese zu sabotieren. Eine Variante des Offline-Angriffs wäre der Wörterbuchangriff. Um diese Art von Angriffen zu erschweren, sollen mehrere Datenbankinstanzen gepflegt werden, die jeweils die gleiche Anzahl an Passwörtern beinhalten, bei denen aber nur ein einziger Datensatz der Richtige ist. Sollte das Gerät mit dem Passwortmanager entwendet und ein falsches Master-Passwort eingegeben werden, wird einer der falschen Datensätze geöffnet und der Angreifer sieht Passwörter, die den echten Passwörtern zum Verwechseln ähneln, aber falsch sind, die von den Autoren Köder genannt werden. Webseiten oder Dienste könnten dann den Zugang verwehren und das Konto sperren, sollte der Angreifer sich mit einem Köder-Passwort authentisieren. Dazu müssten Webseitenbetreiber aber ihre Dienste anpassen, um dies Prüfung durchführen zu können.
Die Autoren des Papers stellen verschiedene Konzepte vor, um beispielsweise authentische Köder-Passwörter zu generieren, da die Annahme getroffen wird, dass Nutzer vorwiegend unsichere merkbare Passwörter im Passwortmanager abspeichern. Mit dieser Annahme wird aber der Sinn und Zweck eines Passwortmanagers verfehlt, der eher dazu dienen soll, dass schwache Passworte durch starke zufällige Passwörter er-

[2]GitHub: https://github.com/ranisalt/node-argon2, abgerufen am 30.11.2018
[3]GitHub: https://github.com/phxql/argon2-jvm, abgerufen am 30.11.2018

setzt werden, ohne dass sich der Nutzer diese merken muss. Da das Hauptziel der Erschwerung von Offline-Angriffen auch mit adäquaten kryptografischen Verfahren erreicht werden kann und die Implementierung der vorgestellten Architektur einen erheblichen Grad an Komplexität erzeugt, wurde die Kamouflage-Architektur bei der Konzeption der Architektur nicht beachtet.
Stattdessen wurde eine modulare Architektur geschaffen, die sich erweitern lässt und die Sicherheit der Passwörter nicht durch Täuschung des Angreifers erreicht, sondern durch die Wahl adäquater kryptografischer Verfahren, die sich in Zukunft jederzeit um neue sichere Verfahren erweitern lassen. Zudem wird dem Nutzer bei der Bedienung von Safeword nahe gelegt, alle schwachen Passwörter durch starke zufällige Passwörter zu ersetzen.

Kapitel 6

Fazit

Die am Anfang der Masterthesis gesetzten theoretischen Ziele zeigen Erkenntnisse hinsichtlich der Open-Source-Lizenzen und bereits bestehender Passwortmanager auf. Die vorgestellten Lizenzen stellen verschiedene Bedingungen an die Nutzung des Codes und formulieren Pflichten, die der Entwickler einhalten muss. Die für diese Thesis ausgewählte GPLv3-Lizenz gewährt dem Entwickler viele Freiheiten der Nutzung und stellt durch die auferlegten Pflichten einen Beitrag zur Open-Source-Welt sicher. Aus dem Vergleich der Passwortmanager können wichtige Erkenntnisse hinsichtlich Code-Sharing und -Qualität sowie der kryptografischen Verfahren entnommen werden. Der Vergleich der Manager zeigt eindrucksvoll, dass die Sicherheit von proprietären Passwortmanagern nur schwer überprüft werden kann, da der Quellcode nicht einsehbar ist. Durch die Einsicht in den Quellcode kann bei KeePass festgestellt werden, dass sicherheitskritische Algorithmen selbst implementiert wurden. Auch hier wird das Kerckhoffsche Prinzip bestätigt, denn nur bei quelloffenen Passwortmanagern können Schwachstellen frühzeitig erkannt und geschlossen werden. Aus diesem Vergleich konnten nützliche Erkenntnisse gewonnen werden, die direkt in die Implementierung des Passwortmanagers „Safeword“ einfließen und die Qualität nachhaltig verbessern. Der Passwortmanager Safeword wurde in der Entwicklung sukzessive um Funktionalitäten und Verfahren erweitert, die den definierten Zielen entsprechen. So verfügt dieser über eine Datenhaltung mit OpenPGP und unterstützt zudem die Speicherung in einer Datenbank. Es ist ein Passwort-Generator vorhanden, der über einen guten Zufall verfügt und sich frei konfigurieren lässt. Zudem wurde eine simple und moderne, aber auch funktionale Oberfläche geschaffen, die nicht nur schnellen Zugriff auf alle Funktionen erlaubt, sondern sich auch an die Regeln der Material Guidelines hält. Mit Safeword wird ein im Bereich der Passwortmanager neuer Weg eingeschlagen, nämlich das Betreiben von Code-Sharing über alle Plattformen hinweg. Das Code-Sharing zwingt den Entwickler auch dazu, seinen Code so zu strukturieren, sodass dieser auf allen Plattformen gleichermaßen genutzt werden kann. Die Design-

Entscheidung, wichtige Funktionen mittels Adaptern und Modulen zu verkapseln, erhöht den Anteil des Chode-Sharings und vereinfacht die Wartung des Core-Codes. Die Kombination aus Electron und React Native bietet eine gute Balance zwischen einer guten User-Experience und einem hohen Grad an Code-Sharing. Eine wichtige Erkenntnis aus der Entwicklung von Safeword ist jedoch, dass das Ziel, einen möglichst hohen Anteil an geteiltem Code zu erhalten, der nicht die Wahl der Pakete beeinflussen darf. Dank der modularen Architektur ist die Einbettung einer plattformspezifischen Bibliothek in Safeword sehr einfach, sodass bei Bedarf ein besser gewartetes plattformgebundenes Paket gewählt werden sollte.
Einige Ziele der Thesis konnten jedoch nicht vollumfänglich erreicht werden, wie das Testen der Anwendung oder die Implementierung einer Synchronisation über einen Cloud-Dienst. Diese konnten durch Probleme bei der Implementierung und der dadurch verknappten Zeit leider nicht fertiggestellt werden.
Im vorherigen Kapitel 5.2 wurden bereits einige mögliche Erweiterungen genannt, die aber alle keine hohe Dringlichkeit aufweisen. In sicherheitskritischen Anwendungen ist es wichtig, dass sensible Informationen nicht lange als Klartext im Arbeitsspeicher abgelegt sein dürfen. Aus diesem Grund sollte der nächste Schritt sein, die Passwörter verschlüsselt im Store abzulegen. Dazu kann das Modul `Aes` genutzt werden, das schon für den `Masterkey` genutzt wird. Die Änderung erfordert jedoch mehrere Anpassungen, denn diese liegen nicht nur im Core-Code, sondern auch in den View-Elementen, die die Passwörter anzeigen. Dort muss dann, wenn der Nutzer sich das Passwort im Hauptbildschirm anzeigen lassen möchte, oder das Passwort kopieren möchte, erst das Passwort entschlüsselt werden. Durch dieses Verfahren wird während der Nutzung nicht das Passwort unverschlüsselt im Speicher hinterlegt. Des Weiteren sollte über eine Implementierung von OpenPGP für React Native mittels eines nativen Moduls und BouncyCastle oder SpongyCastle anstatt `react-native-openpgpjs` nachgedacht werden, da die in Safeword genutzte Version der Bibliothek schon bald nicht mehr aktuell sein dürfte und erneut Inkompatibilitäten mit React Native auftreten könnten.
Sinnvoll ist die Weiterentwicklung im Rahmen eines Hochschulprojekts, sodass diese Ressourcen für Verbesserungen und die Fertigstellung der teilweise erreichten Ziele genutzt werden können. Somit wird sichergestellt, dass Safeword schlussendlich bereit für den Produktiveinsatz ist.

Anhang A

Recherchebericht

Hochschule Düsseldorf
University of Applied Sciences
HSD

Hochschulbibliothek
University Library
BIB

Thema: Konzeption und Entwicklung eines erweiterbaren Passwortmanagers als Cross-Platform Open-Source-Software mit gemeinsamer Code-Basis

Name: Masterthesis

Fachbereich: Medien

Aufgabe in einem Kurs: Nein

Beschreibung:

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Suchbegriffstabelle

Begriff	Synonym
Vulnerabilities	Schwachstelle
Verwundbarkeit	
Attack	Attack ISO Attack NIST
Threat	Threat ISO Threat NIST
CIA-Triad	CIA-Dreieck
Password	
Entropy	Password quality Password strength
Dictionary Attack	Wörterbuchangriffe
Raindow tables	
Authentication	
Zwei-Faktor-Authentifizierung	Two Factor Authentication Mehr-Faktor-Authentifizierung
Authentication factors comparison	Authentication biometry vs knowledge
Authentication biometry	
PGP	GPG
bcrypt	
PBKDFv2	
SHA2 Familie	SHA256 SHA512
Argon2	argon2 specification
GPL	GPLv3 license
Open source definition	
Kerckhoffsches Prinzip	Kerckhoffs principle
Random number generator	PRNG Pseudorandom number generator
Cross platform development	
Password manager	
Password manager attacks	Password manager security
Password manager comparison	
Code Sharing	

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Übersicht: Verwendete Datenbanken mit Suchanfragen

Google Scholar

1 Suchanfrage

- Open source definition

Researchgate

1 Suchanfrage

- Password quality

Springer Link

1 Suchanfrage

- Pseudorandom number generator

IEEE Computer Society

1 Suchanfrage

- Authentication biometry vs knowledge

USENIX

1 Suchanfrage

- Password manager attacks

3/9

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Verwendete Datenbank: Google Scholar

Suchanfrage:
Open source definition -> Trefferanzahl: 1

Relevante Treffer mit Verfügbarkeitshinweis:

The Open Source Definition 2008	Von Bruce Perens
Online verfügbar: http://www.academia.edu/download/31165688/2_semester_projects_glossary_0708_glossary_2sem_0708_aia4_srokamichal_osdtext.pdf	

4/9

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Verwendete Datenbank: Researchgate

Suchanfrage:
Password quality -> Trefferanzahl: 1

Relevante Treffer mit Verfügbarkeitshinweis:

Password Entropy and Password Quality
Online verfügbar: https://www.researchgate.net/ publication/221204731_Password_Entropy_and_Password_Quality

5/9

Hochschule Düsseldorf
University of Applied Sciences
HSD

Hochschulbibliothek
University Library
BIB

Verwendete Datenbank: Springer Link

Suchanfrage:
Pseudorandom number generator -> Trefferanzahl: 1

Relevante Treffer mit Verfügbarkeitshinweis:

Cryptanalytic attacks on pseudorandom number generators 1998
Online verfügbar: https://link.springer.com/chapter/10.1007/3-540-69710-1_12

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Verwendete Datenbank: IEEE Computer Society

Suchanfrage:
Authentication biometry vs knowledge -> Trefferanzahl: 1

Relevante Treffer mit Verfügbarkeitshinweis:

Comparing passwords, tokens, and biometrics for user authentication 2003
Online verfügbar: https://dx.doi.org/10.1109/JPROC.2003.819611

7/9

Hochschule Düsseldorf
University of Applied Sciences
HSD
Hochschulbibliothek
University Library
BIB

Verwendete Datenbank: USENIX

Suchanfrage:
Password manager attacks -> Trefferanzahl: 1

Relevante Treffer mit Verfügbarkeitshinweis:

Password Managers: Attacks and Defenses 2014
Online verfügbar: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver

8/9

Literatur

Adams, C. (2011). Dictionary attack. In H. C. A. van Tilborg & S. Jajodia (Hrsg.), *Encyclopedia of Cryptography and Security* (S. 332–332). Boston, MA: Springer US. doi:10.1007/978-1-4419-5906-5_74. (Siehe S. 3)

Aloul, F., Zahidi, S., & El-Hajj, W. (2009 Mai). Two factor authentication using mobile phones. In *2009 IEEE/ACS International Conference on Computer Systems and Applications* (S. 641–644). doi:10.1109/AICCSA.2009.5069395. (Siehe S. 19, 20)

Bojinov, H., Bursztein, E., Boyen, X., & Boneh, D. (2010). Kamouflage: Loss-Resistant Password Management. In D. Gritzalis, B. Preneel, & M. Theoharidou (Hrsg.), *Computer security – ESORICS 2010* (S. 286–302). Berlin, Heidelberg: Springer Berlin Heidelberg. (Siehe S. 6, 98).

Bundesamt für Sicherheit in der Informationstechnik. (2016). BSI-Grundschutz Katalog. Zugriff unter http://www.bsi.de/gshb/deutsch/index.htm. (Siehe S. 5, 16, 98)

Burnett, M. (2006). *Perfect Password: Selection, Protection, Authentication*. Elsevier Science. (Siehe S. 3).

E. Smith, R. (2012 November). A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles. *IEEE Security & Privacy*, *10*, 20–25. doi:10.1109/MSP.2012.85. (Siehe S. 40–42)

Eckert, C. (2014). *IT-Sicherheit: Konzepte - Verfahren - Protokolle* (9. Aufl.). De Gruyter Studium. De Gruyter Oldenbourg. (Siehe S. 1, 3, 17, 19).

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587–598. doi:10.2307/25148740. (Siehe S. 9)

FOSSA Europäische Komission. (2016). Code Review Results Report – KeePass Password Safe. Zugriff unter https://joinup.ec.europa.eu/sites/default/files/inline-files/DLV%20WP6%20-01-%20KeePass%20Code%20Review%20Results%20Report_published.pdf. (Siehe S. 34)

Garfinkel, S. (1994). *PGP: Pretty Good Privacy*. O'Reilly Media. (Siehe S. 61).

Gasti, P. & Rasmussen, K. B. (2012). On the Security of Password Manager Database Formats. In M. Foresti Saraand Yung & F. Martinelli (Hrsg.), *Computer*

Security – ESORICS 2012 (S. 770–787). Berlin, Heidelberg: Springer Berlin Heidelberg. (Siehe S. 6).

Grassi, P., Newton, E., Perlner, R., Regenscheid, A., Fenton, J., Burr, W., ... Theofanos, M. (2017). *SP 800-63b: Digital Identity Guidelines: Authentication and Lifecycle Management*. NIST: National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-63b. (Siehe S. 14)

Heiderich, D.-I. M., Aranguren, D.-I. A., Inführ, A., & Fässler, F. (2016). *Pentest-Report Padlock.io 04.2016*. Cure53. Zugriff unter https://padlock.io/docs/padlock-pentest-1604.pdf. (Siehe S. 36, 55)

Huth, A., Orlando, M., & Pesante, L. (2012). Password security, protection, and management. *US CERT*. (Siehe S. 4, 5, 15).

Institut für Rechtsfragen der Freien und Open Source Software (München). (2005). *Die GPL kommentiert und erklärt*. O'Reilly. (Siehe S. 9).

Karole, A., Saxena, N., & Christin, N. (2011). A Comparative Usability Evaluation of Traditional Password Managers. In K.-H. Rhee & D. Nyang (Hrsg.), *Information Security and Cryptology - ICISC 2010* (S. 233–251). Berlin, Heidelberg: Springer Berlin Heidelberg. (Siehe S. 38).

Kelsey, J., Schneier, B., Wagner, D., & Hall, C. (1998). Cryptanalytic attacks on pseudorandom number generators. In S. Vaudenay (Hrsg.), *Fast Software Encryption* (S. 168–188). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-69710-1_12. (Siehe S. 15)

Kerckhoff, A. (1883 Januar). La cryptographie militaire. *Journal des sciences militaires*, *9*, 5–83. (Siehe S. 8).

Kojima, R. e. a. (2016 Januar). *Free and open source software (foss) and other alternative license models in japan*. Springer International Publishing. (Siehe S. 9).

Li, Z., He, W., Akhawe, D., & Song, D. (2014). The emperor's new password manager: security analysis of web-based password managers. In *23rd USENIX security symposium (USENIX security 14)* (S. 465–479). San Diego, CA: USENIX Association. Zugriff unter https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-li-zhiwei.pdf. (Siehe S. 6)

Lucas, M. W. (2006). *PGP & GPG: Email for the Practical Paranoid* (1. Aufl.). No Starch Press. (Siehe S. 53).

Ma, W., Campbell, J., Tran, D., & Kleeman, D. (2010). Password Entropy and Password Quality. In *Fourth International Conference on Network and System Security* (S. 583–587). doi:10.1109/NSS.2010.18. (Siehe S. 15)

Metzger, A. (2015). *Free and Open Source Software (FOSS) and other Alternative License Models: A Comparative Analysis*. Ius Comparatum - Global Studies in Comparative Law. Springer International Publishing. (Siehe S. 29).

Müller, S., Krummeck, G., & Romsy, M. (2016). *Dokumentation und Analyse des Linux-Pseudozufallszahlengenerators: eine Studie im Auftrag des Bundesamtes für Sicherheit in der Informationstechnik (BSI)*. Bundesamt für Sicherheit in der Informationstechnik. Zugriff unter https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/LinuxRNG/LinuxRNG.pdf. (Siehe S. 15)

Newton, E., Perlner, R., Burr, W., Dodson, D., Polk, T., Gupta, S., & Nabbus, E. (2013). *SP 800-63-2: Electronic Authentication Guideline*. NIST: National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-63-2. (Siehe S. 2, 15)

Oechslin, P. (2003). Making a Faster Cryptanalytic Time-Memory Trade-Off. In D. Boneh (Hrsg.), *Advances in Cryptology - CRYPTO 2003* (S. 617–630). Berlin, Heidelberg: Springer Berlin Heidelberg. (Siehe S. 3).

O'Gorman, L. (2003 Dezember). Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, *91*(12), 2021–2040. doi:10.1109/JPROC.2003.819611. (Siehe S. 18, 19)

ISO/IEC 27000: information technology – security techniques. (2018). *ISO/IEC 27000*. (Siehe S. 1, 12–14).

Peltier, T. R. (2013). *Information Security Fundamentals, Second Edition*. CRC Press. (Siehe S. 13, 14).

Perens, B. (1999). The Open Source Definition. In *Open Source: Voices from the Open Source Revolution* (S. 171–188). O'Reilly. (Siehe S. 9, 16).

PGP Corporation. (2003). *An Introduction to Cryptography* (1. Aufl.). PGP Corporation. (Siehe S. 46).

Picolet, J. & LLC, N. (2016). *Hash Crack: Password Cracking Manual*. CreateSpace Independent Publishing Platform. (Siehe S. 2).

Poimann, A. (2018). Eins für alle - Fünfzehn Passwortmanager im Test. *c't - Magazin für Computer und Technik*. (Siehe S. 30).

Schilling, A. (2018). Schutzmaßnahmen zur sicheren Identifizierung und Authentifizierung für Cloud-basierte Systeme. In G. Borges & B. Werners (Hrsg.), *Identitätsmanagement im Cloud Computing: Evaluation ökonomischer und rechtlicher Rahmenbedingungen* (S. 33–51). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-662-55584-2_3. (Siehe S. 18)

Shannon, C. E. (1949 Oktober). Communication theory of secrecy systems. *The Bell System Technical Journal*, *28*(4), 656–715. doi:10.1002/j.1538-7305.1949.tb00928.x. (Siehe S. 41)

Silver, D., Jana, S., Boneh, D., Chen, E., & Jackson, C. (2014). Password Managers: Attacks and Defenses. In *23rd USENIX Security Symposium (USENIX Security*

14) (S. 449–464). San Diego, CA: USENIX Association. Zugriff unter https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-silver.pdf. (Siehe S. 6)

Stajano, F., Mjølsnes, S., Jenkinson, G., & Thorsheim, P. (2016). *Technology and Practice of Passwords: 9th International Conference, PASSWORDS 2015, Cambridge, UK, December 7-9, 2015, Proceedings.* Lecture Notes in Computer Science. Springer International Publishing. (Siehe S. 3).

Stobert, E. & Biddle, R. (2014). The Password Life Cycle: User Behaviour in Managing Passwords. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)* (S. 243–255). Menlo Park, CA: USENIX Association. Zugriff unter https://www.usenix.org/system/files/conference/soups2014/soups14-paper-stobert.pdf. (Siehe S. 14)

Stoneburner, G., Goguen, A. Y., & Feringa, A. (2012). *SP 800-30. Risk Management Guide for Information Technology Systems.* U.S. Department of Commerce. Gaithersburg, MD, United States: National Institute of Standards & Technology. doi:10.6028/NIST.SP.800-30r1. (Siehe S. 12–14)

Swoboda, J., Pramateftakis, M., & Spitz, S. (2008). *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen.* Studium : IT-Sicherheit und Datenschutz. Vieweg+Teubner Verlag. (Siehe S. 1, 4, 12, 17).

Taha, M., Ahmed, T., E. Moktar, A., H. Salim, A., & M. Abdullah, S. (2013 August). On password strength measurements: Password entropy and password quality. In *International Conference On Computing, Electrical And Electronic Engineering (ICCEEE)* (S. 497–501). doi:10.1109/ICCEEE.2013.6633989. (Siehe S. 15)

Tietz, C., Pelchen, C., Meinel, C., & Schnjakin, M. (2017). *Management Digitaler Identitäten.* Aktueller Status und zukünftige Trends. Universitätsverlag Potsdam. (Siehe S. 17).

Trojahn, M. (2016 Januar). *Sichere Multi-Faktor-Authentifizierung an Smartphones mithilfe des Tippverhaltens.* Springer Berlin Heidelberg. doi:10.1007/978-3-658-14049-6. (Siehe S. 18, 19)

Turan, M. S., Barker, E. B., Burr, W. E., & Chen, L. (2010). *SP 800-132. Recommendation for Password-Based Key Derivation: Part 1: Storage Applications.* U.S. Department of Commerce. Gaithersburg, MD, United States: National Institute of Standards & Technology. doi:10.6028/NIST.SP.800-132. (Siehe S. 3)

Wiemer, F. & Zimmermann, R. (2014 Dezember). High-speed implementation of bcrypt password search using special-purpose hardware. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)* (S. 1–6). doi:10.1109/ReConFig.2014.7032529. (Siehe S. 4)

Zhao, R. & Yue, C. (2014). Toward a secure and usable cloud-based password manager for web browsers. *Computers & Security*, *46*, 32–47. doi:10.1016/j.cose.2014.07.003. (Siehe S. 6)

Zhou, B., Neamtiu, I., & Gupta, R. (2015). A Cross-platform Analysis of Bugs and Bug-fixing in Open Source Projects: Desktop vs. Android vs. iOS. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering* (7:1–7:10). EASE '15. Nanjing, China: ACM. doi:10.1145/2745802.2745808. (Siehe S. 8)