

**Fakultät
Elektronik und Informatik**

57640 IS-Projekt

Projektarbeit
über das Thema

**Symmetrische Software-Kryptographie zur sicheren
Nutzung von Storage-Services (IaaS)**

Konzept und Prototyp

Autoren: Michael Holzwarth
35429
Felix Friedrich
34648

Betreuender Prof.: Prof. Dr. Christian Koot

Abgabedatum: 07.10.2014

I Ehrenwörtliche Erklärung

Aalen, den 07.10.2014

Wir versichern hiermit ehrenwörtlich durch unsere Unterschriften, dass wir die vorliegende Projektarbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften oder dem Internet entnommen worden sind, sind als solche kenntlich gemacht. Keine weiteren Personen waren an der geistigen Herstellung der vorliegenden Arbeit beteiligt. Die Arbeit hat noch nicht in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung dieser oder einer anderen Prüfungsinstanz vorgelegen.

Michael Holzwarth

Felix Friedrich

II Kurzfassung

Die folgende Arbeit befasst sich mit der Frage ob und wie es, grade auf dem Hintergrund jüngsten Enthüllungen über Geheimdienstaktivitäten, möglich ist Cloudanbieter privat oder geschäftlich für die Speicherung geheimer und/oder wertvoller Daten zu ermöglichen. Im Fokus steht hierbei, die Vorteile die das Cloudcomputing mit sich bringt nutzbar zu machen, ohne die Nachteile der Tatsache, dass Daten fremdem Zugriff ausgeliefert werden, in Kauf nehmen zu müssen oder die Nachteile soweit wie möglich zu eliminieren. In diesem Zusammenhang wird ein Prototyp entwickelt und vorgestellt, der in der Lage ist Daten sicher verschlüsselt auf dem Cloudspeicher von Google abzulegen. Anschließend wird die Bedeutung dieser Entwicklung eingeschätzt und ein Ausblick auf zukünftig mögliche Weiterentwicklung und Verbesserungen gegeben.

II Inhaltsverzeichnis

I	Ehrenwörtliche Erklärung	I
II	Kurzfassung	II
II	Inhaltsverzeichnis	III
IV	Abbildungsverzeichnis	VI
V	Tabellenverzeichnis	VI
1	Einleitung	1
1.1	Motivation	1
1.2	Problemstellung und Problemabgrenzung	1
1.2.1	Annahmen	1
1.2.2	Problemstellung	1
1.2.3	Problemabgrenzung	2
1.3	Ziel der Arbeit	2
1.4	Gang der Arbeit	3
2	Grundlagen	4
2.1	Cloud Computing	4
2.1.1	Definition	4
2.1.2	Daseinsberechtigung	4
2.1.3	Charakteristik	7
2.1.4	Servicemodelle	8
2.1.5	Bereitstellungsmodelle	9
2.1.6	Allgemeine Sicherheitskriterien für IaaS	9
2.2	Java	11
2.2.1	Java Development Kit	11
2.2.2	Programmiersprache Java	11
2.2.3	Java Runtime Environment	12
2.3	Kryptographie	13
2.3.1	Verschlüsselung	14
2.3.2	Integrität	17
3	Technische Analyse/Umsetzung	20
3.1	Cloud-Anbieter	20
3.1.1	Magisches Quadrat	20
3.1.2	Entscheidungsgrundlage Cloud-Anbieter	22
3.2	Google Cloud Storage	23
3.3	Schnittstelle Google Cloud Storage: gsutil und boto	23
3.4	Eclipse IDE for Java Developers	25

3.5	Versionsverwaltung: git	25
3.6	Kryptographie	27
3.6.1	Notwendige Verfahren	27
3.6.2	Entscheidungsgrundlage: BSI	27
4	Ergebnisse	28
4.1	Architektur	28
4.1.1	Package: control	28
4.1.2	Subpackage: util	30
4.1.3	Package: model	30
4.1.4	Subpackage: cc	31
4.1.5	Package: view	31
4.2	UML-Diagramme	33
4.3	Anwendungsfälle	34
4.3.1	Begriffslexikon Anwendungsfälle:	34
4.3.2	AF-001 Benutzer erstellen:	35
4.3.3	AF-002 Authentifizierung:	36
4.3.4	AF-003 Dateiupload:	37
4.3.5	AF-004 Dateidownload:	38
4.3.6	AF-005 Datei löschen:	39
4.4	Kryptosystem	40
4.4.1	Account anlegen	40
4.4.2	Login	40
4.4.3	Dateien ablegen	41
4.4.4	Programm schließen	41
4.4.5	Programmstart mit vorhandener state.cfg	42
4.4.6	Datei herunterladen	42
4.5	GUI und Funktionalitäten	43
4.5.1	Login	43
4.5.2	Account erstellen	43
4.5.3	Hauptfenster	44
4.5.4	File Menü	44
4.5.5	Dateiauswahl	45
4.5.6	Help Menü	45
4.5.7	Help Fenster	46
4.5.8	About Fenster	46
4.5.9	Kontextmenü	47
4.5.10	Delete Fenster	47
4.5.11	Synchronisierungsstatus	48
5	Diskussion	49
5.1	Bedeutung	49
5.2	Erweiterung der Funktionalitäten	50

5.2.1	Erhöhung der Bedienerfreundlichkeit	50
5.2.2	Serverseitiges Management	53
5.2.3	Erweiterte Sicherheit	55
6	Literatur	60
	Anhang	I
A	Installationsanleitung	I
B	Digitaler Anhang	I

IV Abbildungsverzeichnis

Abb. 1	Speicherbedarf	5
Abb. 2	Wirtschaftlichkeit [MWRS11, S. 39]	7
Abb. 3	IaaS Modelle [HSG10, S. 77]	8
Abb. 4	Magisches Quadrat [LTG ⁺ 14]	20
Abb. 5	OAuth 2.0 [Inc14c]	24
Abb. 6	Eclipse	26
Abb. 7	Atlassian SourceTree	26
Abb. 8	Anwendungsfalldiagramm	34
Abb. 9	Login	43
Abb. 10	Account erstellen	43
Abb. 11	Hauptfenster	44
Abb. 12	File Menü	44
Abb. 13	Dateiauswahl	45
Abb. 14	Help Menü	45
Abb. 15	Help Fenster	46
Abb. 16	About Fenster	46
Abb. 17	Kontextmenü	47
Abb. 18	Delete Fenster	47
Abb. 19	Synchronisierungsstatus	48

V Tabellenverzeichnis

Tab. 1	AF-001 Benutzer erstellen	35
Tab. 2	AF-002 Authentifizierung	36
Tab. 3	AF-003 Dateiupload	37
Tab. 4	AF-004 Dateidownload	38
Tab. 5	AF-005 Datei löschen	39

1 Einleitung

1.1 Motivation

Unternehmensdaten müssen aus Gründen der Einhaltung gesetzlicher Vorschriften oder des Unternehmensgeheimnisses vertraulich behandelt und geschützt werden. Die gleiche Sicherheit wie bei der internen Datenhaltung muss auch in einem ausgelagerten System, zum Beispiel der Cloud, mit derselben Sorgfalt gewährleistet sein. Eine Auslagerung der Daten hebt nicht die sicherheitsspezifischen Anforderungen an Vertraulichkeit und Datenschutz auf. Eine große Herausforderung stellt hierbei der Verlust über die absolute Kontrolle der Daten und damit einhergehend die Komplexität des Schutzes dar.

Cloud-Anbieter verfügen über verschiedene Sicherheitsmechanismen zum Schutz vor unbefugtem Datenzugriff von außerhalb. Es existiert jedoch kein garantierter Schutz vor unbefugtem Zugriff innerhalb des Anbieters. Administratoren und weiteres, über die Infrastruktur berechtigtes Personal besitzen die Möglichkeit auf gespeicherte Daten zuzugreifen.

Dies macht eine verschlüsselte Ablage der Daten notwendig um sicherzustellen, dass die Daten auch innerhalb der Cloud in vollem Umfang den Sicherheitsstandards entsprechen.

1.2 Problemstellung und Problemabgrenzung

1.2.1 Annahmen

Da die Notwendigkeit an Sicherheit daraus resultiert, dass fremde Systeme grundsätzliche ersteinmal als unsicher gelten müssen, gehen wir bei der Entwicklung des Prototypen davon aus, dass der Client auf dem die Applikation läuft als sicher gilt. Szenarien in denen auch dieser als korrumpiert gilt, lassen sich grundsätzlich nicht mehr absichern.

1.2.2 Problemstellung

Eine symmetrische Software-Kryptographie soll möglichst Daten, die in eine Cloud ausgelagert werden, vor Fremdzugriff schützen. Die Konzeption und der Entwurf eines Prototyps sollen evaluieren, wie diese Herausforderung effizient und sicher gelöst

werden kann. Dabei steht der betriebssystemunabhängige Zugriff auf die entsprechende Applikation und das eingesetzte Verschlüsselungsverfahren im Vordergrund. Erarbeitet werden soll die Implementierung für einen einzelnen Cloud-Anbieter.

Zu den Grundfunktionalitäten des geforderten Prototyps gehören der Cloud-Zugriff, die Verschlüsselung, die Speicherung in der Cloud, eine Dateiübersicht und das lokale Herunterladen der Daten.

Die Verwendung soll durch eine graphische Oberfläche vereinfacht werden.

1.2.3 Problemabgrenzung

Eine Gegenüberstellung weiterer kommerzieller Produkte von Cloud-Anbietern im Bezug auf Implementierung und Performance wird in dieser Arbeit nicht behandelt. Ebenso ist eine Optimierung vorhandener APIs bzw. Schnittstellen und Kryptosystemen nicht Gegenstand dieser Arbeit. Die Hardware des Anwenders wird als sicher betrachtet, weswegen es keines sicheren Schlüsselmanagements bedarf. Dateien, welche manuell bzw. ohne die Applikation in die Cloud geladen werden, werden nicht berücksichtigt.

1.3 Ziel der Arbeit

Ziel der Arbeit ist es herauszuarbeiten, ob und wie Cloudspeicher zur sicheren Ablage von Daten im Sinne des deutschen Datenschutzverständnisses genutzt werden können. Hierfür soll eine entsprechende Applikation entwickelt werden. Außerdem ist zu klären, ob der Einsatz einer Cloudplattform sinnvoll und möglich ist, auf Hintergrund der jüngsten Enthüllungen Edward Snowdens und anderer. Diese haben gezeigt, dass eine solche Infrastruktur auf US-amerikanischem Boden im Besonderen, aber auch im Ausland im Allgemeinen als nicht vertrauenswürdig gelten muss und dort abgelegte unternehmenswichtige Daten gefährdet sind. Ist es möglich die Vorteile solcher Plattformen dennoch zu nutzen? Wie gefährdet sind die Daten dann noch?

Hinzu kommt eine Abwägung, ob und wie eine solche Applikation marktfähig entwickelt und vertrieben werden kann.

1.4 Gang der Arbeit

Zunächst werden im Kapitel **2 Grundlagen** alle Grundlagen erläutert, die zum Verständnis der in **3 Technische Analyse/Umsetzung** beschriebenen Konzeptionierung und Implementierung dienen. Darin werden die verwendeten Sicherheitsalgorithmen, die Notwendigkeit von Cloudcomputing, dessen wirtschaftliche Aspekte, die Abgrenzung der einzelnen Anbieter, die Programmiersprache Java, das Kryptosystem AES sowie die zur Entwicklung der Applikation verwendeten Werkzeuge und Technologien beschrieben. Nachfolgend wird in **4 Ergebnisse** die Applikation selbst in Architektur und Aufbau des Programmcodes und das entworfene Sicherheitskonzept erläutert. Außerdem findet sich hier auch eine detaillierte Beschreibung des graphischen Interfaces und eine Bedienungsanleitung. In **5 Diskussion** werden die Ergebnisse (die Applikation) im Sinne der Motivation und der Zielsetzung bewertet sowie Möglichkeiten zur Optimierung der Applikation und eine Aussicht auf zukünftige Weiterentwicklungen gegeben.

In der Ausarbeitung werden teilweise Anglizismen und englische Begriffe in deutschsprachigen Fließtexten verwendet, zum Teil in Eindeutschungen. Dies geschieht, da das Zielpublikum eine an der Informatik interessierte Fachöffentlichkeit ist. Wir gehen davon aus, dass viele Begriffe der Informatik auch im deutschen Sprachgebrauch auf englisch verwendet werden und so in diesem Zusammenhang eindeutiger sind als ihre deutsche Übersetzung. So ist zum Beispiel ein `Bytearray` ein stehender Begriff, während die Übersetzung `Bytefeld` einer Erläuterung bedürfen würde.

Die Dokumentation des Programmcodes sowie die UML Diagramme sind auf Englisch gehalten, da auch im Programmcode selbst ausschließlich englische Methoden-, Variablen- und Klassennamen verwendet wurden. Dies geschah, da in unseren Augen Englisch die Fachsprache der Informatik ist und außerdem ansonsten eine unleserliche Vermischung von deutschen Variablen und Methodennamen mit der grundsätzlich englischen Notation der Programmiersprachen entstehen würde.

2 Grundlagen

2.1 Cloud Computing

Da der Begriff Cloud Computing nicht eindeutig definiert und in seinem strukturellen Aufbau unterschiedlich ist, wird nachfolgend Cloud Computing beschrieben und in diesem Zusammenhang näher auf die Merkmale von Infrastructure as a Service (IaaS) eingegangen.

2.1.1 Definition

Definition von Cloud Computing nach dem National Institute of Standards and Technology (NIST) und der European Network and Information Security Agency (ENISA) [ST11]:

”Cloud Computing ist ein Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.”

Definition von Cloud Computing nach dem Bundesamt für Sicherheit in der Informationstechnik (BSI) [Inf14a]:

”Cloud Computing bezeichnet das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz. Angebot und Nutzung dieser Dienstleistungen erfolgen dabei ausschließlich über definierte technische Schnittstellen und Protokolle. Die Spannbreite der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software.”

2.1.2 Daseinsberechtigung

Cloud Computing bietet Unternehmen die Möglichkeit hohe Investitionskosten für die IT-Infrastruktur in variable laufende Kosten umzuwandeln. Dies ist vor allem für kleinere und mittlere Unternehmen sowie Start-Ups interessant, da nicht nur der enorme Kapitaleaufwand bei der Beschaffung von IT-Hardware sondern auch anfallende

Kosten für die Administration umgewandelt und minimiert werden können [MSLL14, S. 78]. Die Möglichkeit einer Bereitstellung zusätzlicher Ressourcen bietet zudem eine flexible Alternative zu einer stetigen anforderungsorientierten Erweiterung der IT-Landschaft. Dies ermöglicht eine risikolose, bedarfsgerechte Steuerung der Kapazitäten und Konzentration auf die Kernkompetenz des Unternehmens. Im Allgemeinen betreffen diese Vorzüge alle Bereiche der IT-Infrastruktur (Rechenleistung etc.) [MWRS11, S. 35]. Im Projekthintergrund wird nachfolgend im Detail der wirtschaftliche Aspekt der Speichernutzung bei IaaS betrachtet [MWRS11, S. 35ff.]:

Elastische Skalierbarkeit

Gängige Ansätze fordern eine kostenintensive und zeitlich aufwändige Analyse bestehender Systeme und möglicher Erweiterungen. Schwankungen des Bedarfs erschweren eine exakte Kalkulation notwendiger Betriebsmittel, sodass eine Fehl- oder wohlwollend angesetzte Kalkulation zu ungenutzten Kapazitäten und einer unnötigen Belastung der Umwelt führt [MSLL14, S. 91ff.]. Durch Cloud Computing können innerhalb kürzester Zeit Ressourcen bereitgestellt oder entzogen werden.

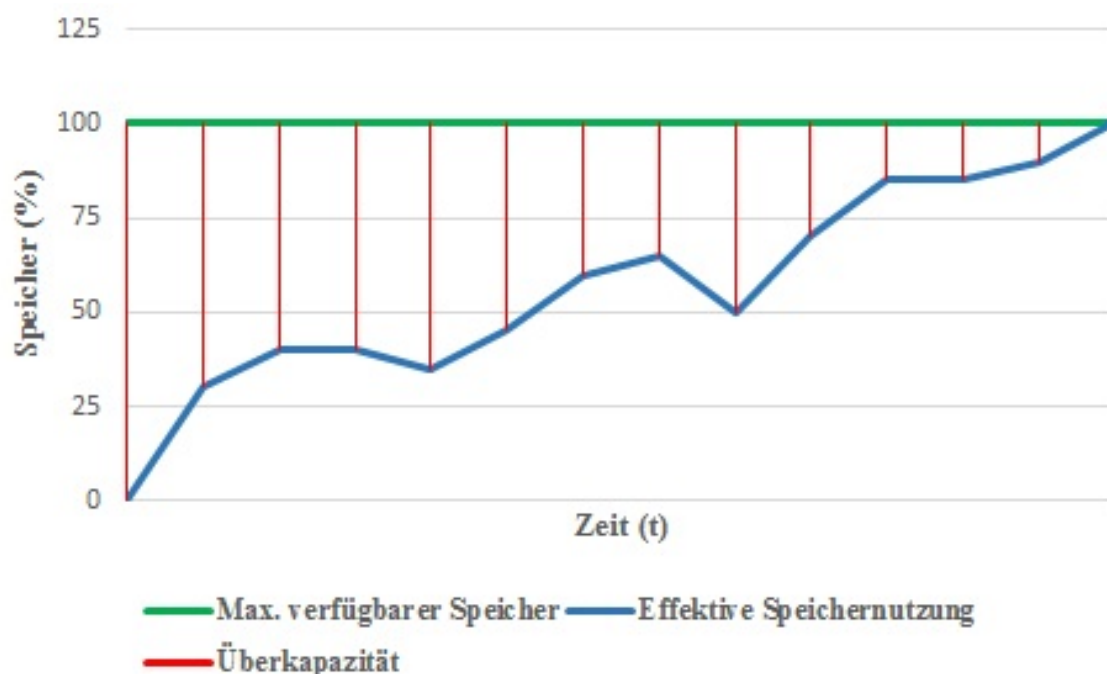


Abbildung 1: Speicherbedarf

Abbildung 1 stellt schemenhaft den Speicherbedarf eines Unternehmens dar. Eine Überkapazität tritt dann ein, wenn der zum Zeitpunkt x verwendete Speicher unter dem tatsächlich verfügbaren Speicher liegt - eine Unterkapazität, wenn mehr Speicher benötigt wird, als vorhanden ist. Eine Kalkulation benötigter Ressourcen ist zu Beginn schwierig und eine Überkapazität sogar, um einen ständigen Integrationsaufwand zu vermeiden, gewünscht. Der Anstieg an Daten führt unweigerlich zur Erreichung der Kapazitätsgrenzen. Im Extremfall wird die Migration besserer Systeme notwendig, was teilweise Geschäftsprozesse beeinflussen kann. In jedem Fall führt jedoch eine Beschaffung und Integration neuer Hardware zu finanziellem und administrativem Aufwand. Sinkt der Speicherbedarf wieder, bspw. durch Löschen redundanter Daten, wäre eine solche irreversible Investition, kurzfristig betrachtet, als unnötig anzusehen. Bei Cloud Computing wird diese Herausforderung auf den Anbieter übertragen - der benötigte Speicher kann bedarfsgerecht gesteuert und abgerechnet werden.

Materielle Wertverluste

Der stetige technische Fortschritt führt zu einem rapiden Wertverlust der vorhandenen Hardware. Auch die Lebensdauer von stark beanspruchter Hardware, wie beim Einsatz in Servern, macht einen Austausch notwendig. Deshalb ist es nicht sinnvoll für ein Unternehmen einen großen, nicht verwendeten "Vorratsspeicher" anzulegen. Dieser müsste eventuell noch vor Erreichung seiner maximaler Auslastung ersetzt werden. Dies bedeutet, dass zum einen die Instandhaltung der IT-Infrastruktur Kosten verursacht, zum anderen aber auch innerhalb einer kurzen Zeitspanne die angefallenen Kosten zum Investitionszeitpunkt nicht mehr dem eigentlichen Wert entsprechen. Ein solcher Wertverlust kann durch Cloud Computing vermieden werden. Die Verwaltung und Aktualität der Infrastruktur obliegt einzig dem Anbieter, welcher aufgrund großer Margen eher in der Lage ist, neuste Technik zu einem moderaten Preis einzukaufen und diese Einsparungen an den Kunden weiterzugeben.

Höhere Wirtschaftlichkeit

$$\text{Nutzung}_{\text{Cloud}} \times (\text{Einnahmen} - \text{Kosten}_{\text{Cloud}}) \leq \frac{\text{Nutzung}_{\text{EZ}} \times (\text{Einnahmen} - \text{Kosten}_{\text{EZ}})}{\text{Auslastung}_{\text{EZ}}}$$

Abbildung 2: Wirtschaftlichkeit [MWRS11, S. 39]

Anhand der Formel in Abbildung 2 kann die Wirtschaftlichkeit des Einsatzes von Cloud Computing kalkuliert werden. Diese gilt prinzipiell für die Berechnung der Rentabilität einer Umstellung der gesamten IT-Infrastruktur auf Cloud Computing, lässt sich jedoch in selber Weise rein auf den ökonomischen Aspekt der Speichernutzung anwenden. Die linke Seite bildet das Ergebnis der Kosten aus Multiplikation der Nettoeinnahmen mit der tatsächlichen Beanspruchung des gebuchten Speichers beim Einsatz von Cloud Computing. Die rechte Seite betrachtet selbiges bei eigener IT-Verwaltung, unter Berücksichtigung der tatsächlichen Auslastung. Der höhere Wert daraus lässt auf eine höhere Kostenersparnis schließen.

[WPG⁺10] [Mil09, S. 71ff.]

2.1.3 Charakteristik

Nach NIST [ST11, S. 2f.] können Cloud Services in fünf Eigenschaften unterschieden werden:

- On-demand Self Service: Keine Interaktion mit dem Cloud Service Provider (CSP)
- Broad Network Access: Ein Zugriff ist nicht an zusätzliche Software gebunden.
- Resource Pooling: Ressourcen des CSP sind gebündelt und allen gleichermaßen zugänglich, eine vertragliche Bindung für einen Speicherort ist jedoch möglich
- Rapid Elasticity: Flexible Erweiterungen der Services
- Measured Services: Ressourcennutzung wird gemessen und überwacht

Zusätzlich werden diese Eigenschaften durch die Cloud Security Alliance (CSA) [All11, S. 14ff.] um nachfolgende Punkte erweitert:

- Mandantenfähigkeit: Ressourcen werden geteilt, Unterscheidung der Mandanten notwendig
- Pay per Use: Nur tatsächlich verwendete Ressourcen müssen bezahlt werden
- Service orientierte Architektur: Grundvoraussetzung für Cloud Computing

2.1.4 Servicemodelle

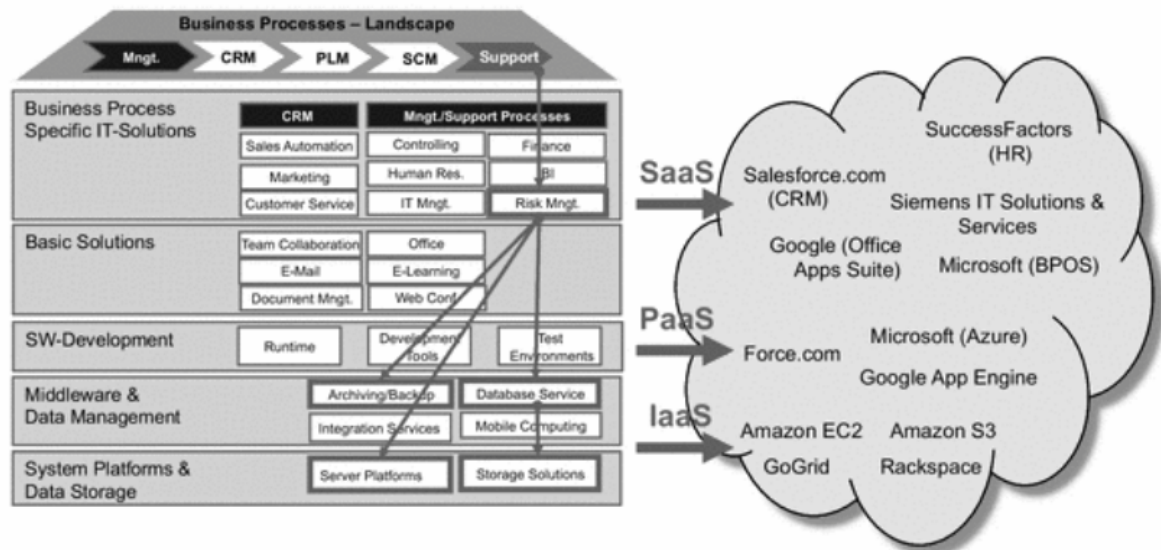


Abbildung 3: IaaS Modelle [HSG10, S. 77]

Cloud Computing kann in seiner Art nach dem sogenannten technischen Cloud-Stack unterschieden werden. Dies ist ein Schichtenmodell, in dem die oberen Schichten auf den unteren aufbauen kann. Jede Schicht kann anhand ihres Abstraktionsgrades klassifiziert werden:

- **Schicht 3 - Software as a Service (SaaS):**
Software, die über eine Cloud-Infrastruktur angeboten wird. Die Bereitstellung, Betreuung und der Betrieb erfolgt gänzlich durch den Anbieter. Eine Abrechnung erfolgt je Softwareaufruf, was den Erwerb der üblichen Software-Lizenz ersetzt. Beispiele hierfür sind *Google Apps* oder *Microsoft Office 365*.
- **Schicht 2 - Plattform as a Service (PaaS):**
Der Anbieter stellt eine verwaltete Cloud-Infrastruktur zur Verfügung, worauf eigene Software bereitgestellt werden kann. Der Kunde erhält zudem eine integrierte Entwicklungs- und Laufzeitumgebung. Eine Abrechnung erfolgt nutzungsabhängig. Zu den Vertretern von PaaS gehören beispielsweise *Google App Engine* oder *Microsoft Azure*.
- **Schicht 1 - Infrastructure as a Service (IaaS):**
Reine, bedarfsgerechte Nutzung der Rechnerinfrastruktur des Anbieters. Der

Anwender muss alles außer der virtuellen Infrastruktur selbst verwalten. Dies bedeutet zwar einen Mehraufwand, bietet aber gleichzeitig eine größtmögliche Skalierbarkeit. Bekannte Anbieter sind *Google Cloud Storage* oder Amazon Webservices (AWS) mit *Amazon S3* oder *Amazon EC2*.

[Inf14a] [EMP13, S. 65f.] [ST11, S. 2f.]

Im Rahmen dieser Projektarbeit bezieht sich das Konzept auf Infrastructure as a Service (IaaS), worauf nachfolgend im Näheren eingegangen wird.

2.1.5 Bereitstellungsmodelle

- **Public IaaS Cloud:**
Cloud Computing Services für die breite Öffentlichkeit - je Server mehrere unabhängige Nutzer.
- **Private IaaS Cloud:**
Cloud Computing Services für Unternehmen - je Server ein Unternehmen.
- **Hybrid IaaS Cloud:**
Cloud Computing Services für Unternehmen - physikalische Server je nach Anwendungsgebiet entweder Public oder Private.
- **Community IaaS Cloud:**
Cloud Computing Services für Unternehmen - Zusammenschluss von Interessensgemeinschaften/Institutionen zur gemeinsamen Nutzung.
- **Personal IaaS Cloud:**
Cloud Computing Service durch den Anwender selbst.

[Inf14a] [ST11, S. 3]

2.1.6 Allgemeine Sicherheitskriterien für IaaS

- **Sicherheitsrichtlinien:**
Jede sicherheitsbewusste Organisation betrachtet und prüft sorgfältig die Einhaltung von Sicherheitsrichtlinien. Die Qualität der Sicherheitspolitik eines IaaS-Anbieters ist ein Indikator dafür, wie dieser für die Verantwortung der Sicherheit einsteht.
- **Unabhängige Sicherheitsbeauftragte:**
Sicherheitsbeauftragte sollten unabhängig berichten, aber in enger Zusammen-

arbeit mit technischem Personal des Cloud-Anbieters stehen. Die Verantwortung des Sicherheitspersonals besteht darin, die ständige Sicherheit des Dienstes zu gewährleisten.

- Upgrades und Patches:

Upgrades und Patches sollten in einer zeitgemäßen und sicheren Form umgesetzt werden um Lücken vor der Enthüllung zu beheben und das Sicherheitsniveau stets aufrecht zu erhalten.

- Untersuchungen:

Der Cloud-Anbieter muss regelmäßig Schwachstellenanalysen durchführen und die Infrastruktur nach Auffälligkeiten untersuchen. Alle Funde müssen auf die möglichen Auswirkungen hin bewertet und zeitnah behoben werden.

- Forensik:

Sicherheitsrelevante Protokolle müssen lang genug beibehalten werden, um die Verfügbarkeit für forensische und gesetzliche Anforderungen zu erfüllen. Solche Protokolle tragen zur Ermittlung bei, wie ein Zwischenfall aufgetreten ist und welche Auswirkungen er hat.

- Handhabung von Zwischenfällen:

Das Management eines Zwischenfalles und dessen Gegenmaßnahmen sollte darauf ausgelegt sein, den Umfang zu dokumentieren und für den Kunden transparent zu halten. Für den Kunden eines Cloud-Anbieters sind hierbei die Rahmenwerte der Reaktion (Erkennung, Offenlegung, Behebung und Prüfung) für eine weitere Kooperation notwendig.

- Geschäftskontinuität:

RPO (engl. Recovery Point Objective) definiert die Höchstmenge an Datenverlust, welche nach einem entsprechenden Vorfall akzeptabel ist. Ausgedrückt wird dies zusätzlich in Zeit, der Zeit zwischen entsprechendem Verlust und Rückführung zur letzten zuverlässigen Sicherung. RTO (engl. Recovery Time Objective) definiert die Höchstzeitdauer, die für die Wiederherstellung und vollständigen Zugriff auf die Daten akzeptabel ist.

[Win11, S. 93ff., 224f.]

2.2 Java

Die Applikation zur sicheren Speicherung in der Cloud wurde komplett in Java in der Version 1.7 implementiert. Der Ursprung von Java liegt in den frühen 90er Jahren, als ein Team um James Gosling (genannt das Green Team) bei Sun Microsystems damit begann, eine der ersten plattformunabhängigen Programmiersprachen zu entwickeln. Sie sollte die Computerwelt revolutionieren, indem sie die Grenzen zwischen den Systemen, die damals noch deutlicher waren als heute, verwischt. Heute gehört die gesamte Java Technologie der Oracle-Gruppe, die Sun Microsystems im Januar 2010 kaufte.

[Ora14][mic14][Ull12, S. 52]

Die Java Technologie, welche die Plattformunabhängigkeit zur Verfügung stellt, besteht im Kern aus drei Teilen:

- JDK: Java Development Kit
- Java: Die Programmiersprache
- JRE: Java Runtime Environment

2.2.1 Java Development Kit

Das **Java Development Kit** ist das Entwicklungswerkzeug mit welchem Java Applikationen erstellt werden können. Seit 2006 wird es von Sun Microsystems unter der Gnu Public License mit einer sog. Linking Exception (Ausnahmegenehmigung für das Linken einer Programmbibliothek) veröffentlicht und mittlerweile wird es von einer OpenSource Community in enger Zusammenarbeit mit Firmen wie IBM und Apple als OpenJDK in einer freien Version weiterentwickelt. Es besteht hauptsächlich aus Bibliotheken, sowie dem Java Compiler **Javac** mit dem aus Java Code der sog. **Java-Bytecode** erzeugt werden kann.

[Ull12, S. 63f]

2.2.2 Programmiersprache Java

Die Programmiersprache **Java** ist eng mit C++ verwandt und strikt objektorientiert. Das Besondere an Java ist außerdem eine Vielzahl an Bibliotheken, die vom JDK direkt mitgeliefert und (als es noch ausschließlich Sun gehörte) von Sun Microsystems gewartet werden. Hierdurch können viele Standardaufgaben, die bei der Entwicklung

anfallen, direkt mit einer Java Bibliothek gelöst werden, ohne dass diese Funktionalitäten erst entwickelt oder weite externe Bibliotheken eingebunden werden müssten, die dann weitere Lizenzen erfordern oder zweifelhaften Ursprungs sind.

[Ull12, S. 55, 60]

2.2.3 Java Runtime Environment

Das **Java Runtime Environment** ist die Besonderheit der Java Technologie, welche den plattformunabhängigen Betrieb der Java Applikationen erst ermöglicht. Das JRE nimmt keinen üblichen Maschinencode entgegen, der prozessorgebunden wäre und damit nur auf bestimmten Architekturen laufen könnte, sondern **Java-Bytecode**. Dieser Bytecode besteht auch aus Prozessoranweisungen, allerdings für den Prozessor der virtuellen Maschine, welche das JRE zur Verfügung stellt. Darin besteht der Kern der Plattformunabhängigkeit: Wurde einmal ein JRE für die betreffende Architektur und das Betriebssystem entwickelt, sind Java Applikationen auf diesem lauffähig, solange sie sich strikt an Java und seine Bibliotheken halten.

Diese Besonderheit ist auch der Grund, aus dem die Wahl der Programmiersprache zu Beginn des Projektes auf Java fiel. Zum Einen kann die Software ohne Portierungsaufwand auf allen gängigen Betriebssystemen verwendet werden, zum Anderen entfällt ein großer Teil des Entwicklungsaufwandes für Funktionalitäten die nicht dem Kern der Applikation dienen, wie ein graphisches Interface oder der Umgang mit Dateien und Ordnern. Dadurch lässt sich mehr Zeit auf die eigentlichen Kernfunktionen wie Verschlüsselung und Cloudanbindung, sowie das zugrunde liegende Sicherheitskonzept verwenden.

[Ull12, S. 63f]

2.3 Kryptographie

Unter dem Begriff Kryptographie wird meist das Ver- und Entschlüsseln von Botschaften verstanden, mit dem Anspruch, dass nur berechtigte Personen in der Lage sind, diese zu lesen. Dies ist zwar nur bedingt falsch, vernachlässigt aber wichtige Aspekte, die von essentieller Bedeutung für die moderne Kryptographie sind: Kryptographie umfasst heute die vier Kernziele

- Vertraulichkeit
- Integrität
- Authentizität
- und Verbindlichkeit.

[SSP08, S. 14-17] [Sch06, S. 2]

Der Aspekt der **Vertraulichkeit** ist der im Einleitungssatz erwähnte. Vertraulichkeit bedeutet, dass eine Botschaft derart geschützt ist, dass sie ausschließlich von autorisierten Personen gelesen werden kann. Dies wird im Computerzeitalter mittels Verschlüsselungsalgorithmen erreicht. Allerdings schützt Vertraulichkeit nicht davor, dass eine Botschaft manipuliert wird. Um dies sicherzustellen muss bei einem Kryptosystem die **Integrität** der Botschaft sichergestellt werden. Das bedeutet nicht, dass eine Botschaft nicht manipuliert werden könnte, es bedeutet nur, dass eine Manipulation sofort entdeckt werden und die Botschaft entsprechend als nicht integer entlarvt werden kann. Nun sind wir in der Lage eine Botschaft zu erzeugen, die weder von Fremden gelesen, noch unbemerkt manipuliert werden könnte. Allerdings kann nicht sichergestellt werden, dass die Botschaft von dem in ihr erwähnten Absender kommt. Deshalb gibt es noch die Aspekte **Authentizität und Verbindlichkeit**. Authentizität garantiert, dass eine Botschaft tatsächlich von der Person kommt, die im Absender erwähnt ist. Meist geht dies Hand in Hand mit Verbindlichkeit: Ist es möglich festzustellen, dass eine Botschaft nur von einer betreffenden Person gesendet werden konnte, ist es dieser Person nicht möglich, den Versandt abzustreiten.

[Sch06, S. 2] [SSP08, S. 14-17]

Im Folgenden sollen vor allem die Aspekte der Vertraulichkeit und Integrität näher erläutert werden, da sie von Bedeutung für den entwickelten Prototypen sind und daher für das Verständnis des Konzeptes wichtig sind.

2.3.1 Verschlüsselung

Wie schon erwähnt, stellt die Verschlüsselung von Nachrichten eine Möglichkeit dar, Vertraulichkeit zwischen zwei Kommunikationspartner herzustellen. Zu diesem Zweck werden schon seit Jahrhunderten Verschlüsselungsverfahren entworfen, um dem Gegner immer einen Schritt voraus zu sein. Die meisten davon sind mittlerweile hinfällig. Selbst wenn sie keine essentielle Schwachstelle haben, gehen sie vor den enormen Rechenkapazitäten von Computern in die Knie. Daher werden in der modernen Kryptographie Verschlüsselungsverfahren mit gigantischen Schlüsselräumen, das Bundesamt für Sicherheit in der Informationstechnik spricht von min. 100 Bit, also 2^{100} möglichen Schlüsseln [Inf14b, S. 15], verwendet. Außerdem arbeitet die moderne Kryptographie nicht mehr auf gewöhnlichen Buchstabenalphabeten, sondern auf Bitebene. Dadurch werden die Algorithmen unabhängig von den zu verschlüsselnden Daten. Diese Verschlüsselungsalgorithmen teilen sich in zwei Hauptgruppen: Symmetrische und asymmetrische Chiffren.

[Sch06, S. 1] [SSP08, S. 15]

Symmetrische Chiffren

Symmetrische Chiffren sind Verschlüsselungsverfahren die zum ver- und entschlüsseln dasselbe Geheimnis nutzen, also symmetrisch sind. Manche dieser Verfahren arbeiten auch mit zwei unterschiedlichen Geheimnissen, wobei aus dem einen mit sehr geringem Aufwand das andere erzeugt werden kann. In diesem Sinne beschreibt symmetrisch die Tatsache, dass der Besitzer eines Geheimnisses sowohl verschlüsselte Nachrichten erzeugen, als auch entschlüsseln kann. Diese Chiffren wiederum teilen sich in zwei Gruppen: Strom- und Blockchiffren. Die häufigste Form der symmetrischen Algorithmen sind Blockchiffren. Eine Blockchiffre erhält als Eingabe einen geheimen Schlüssel sowie eine Nachricht in Blocklänge (je nach Chiffre unterschiedlich) und gibt ein Chifftrat von Blocklänge aus. Hierdurch entstehen einige Probleme für Ihre Verwendung. Nachrichten, die größer oder kleiner der Blocklänge sind, müssen entweder aufgefüllt oder zerteilt werden. Das Auffüllen sorgt dafür, dass ein Angreifer Analysen über den verschlüsselten Klartext anstellen kann, da er vermuten kann, mit was und wieviel der Nachricht aufgefüllt wurden. Daher wurden für dieses Auffüllen Komplexe Mechanismen entworfen, das sogenannte Padding. Stromchiffren dagegen arbeiten auf Bitebene und sind zeitabhängig. Das bedeutet, dass ein Bit unterschiedlich behandelt wird, je nachdem an welcher Stelle es im Chifftrat vorkommt. Am

Besten kann man sich eine Stromchiffre als Pseudozufallszahlengenerator vorstellen (manche sind auch daraus entworfen): Ein solcher Generator erzeugt nach Eingabe des Keys (hier Seed genannt) einen Bitstrom der Objektiv betrachtet zufällig ist, also etwa gleichverteiltes Vorkommen von 0en und 1en hat und keine Regelmäßigkeiten aufweist. Dieser erzeugte Bitstrom wird nun per XOR-Operation mit dem Klartext verknüpft. Jemand der den Bitstrom entschlüsseln möchte benötigt den selben Generator sowie den Seed und ist damit in der Lage mit der selben XOR-Operation den Klartext wiederherzustellen.

[Sch06, S. 4, 223, 249] [SSP08, S. 22-25] [Inf14b, S. 22-24]

Während Bruce Schneier in seinem Standardwerk Angewandte Kryptographie noch beschreibt [Sch06, S. 249], dass Stromchiffren beliebter seien, aber bald ein Umbruch erfolgen könnte, sind Blockchiffren heute deutlich verbreiteter. Dies liegt vermutlich auch an Standards wie AES. AES ist ein Verschlüsselungsstandard des US-amerikanischen National Institute of Standardisation (NIST) und soll im Folgenden, als der häufigste Vertreter der symmetrischen Chiffren, näher erläutert werden:

AES

AES (Advanced Encryption Standard) ist ein kryptographischer Standard, welcher im Zuge einer 1997 bekannt gegebenen Ausschreibung des US-Amerikanischen Handelsministeriums festgelegt wurde. Es hat die mittlerweile als unsicher geltenden DES bzw. 3DES Standards abgelöst, gilt heute als Standard der symmetrischen Verschlüsselung und als für viele Jahre sicher. Es gibt ihn in den Varianten AES-128, AES-192 und AES-256, wobei die Zahlen sich auf die jeweils verwendete Schlüssellänge in Bit beziehen. Der verwendete Algorithmus ist der Rijndael von Vincent Rijmen und Joan Daemen. Unter anderem wird er von der NASA und der US-amerikanischen Regierung verwendet und ist Teil der Standards WPA2, SSH, IPSec, SSL und vieler anderer. Außerdem ist er in jedem größeren Betriebssystem an der einen oder anderen Stelle implementiert. Die sehr hohe Verbreitung ist vor allem seiner Einfachheit, der Möglichkeit ihn in Hardware zu implementieren, der mathematischen Eleganz, dem geringen Rechenaufwand und nicht zuletzt dem offenen Ausschreibungsverfahren geschuldet. Gerade die Besonderheit, dass dieser Standard der US-Amerikaner offen liegt, von jedem selbst implementiert werden kann, nicht übernommen werden

muss und in einem äußerst transparenten Verfahren ausgeschrieben und ausgewählt wurde, wird bis heute gelobt und ist Hauptgrund für das in AES gesetzte Vertrauen. Grundsätzlich ist AES eine symmetrische Blockchiffre. Es arbeitet also bei der Ver- und Entschlüsselung mit denselben Schlüsseln. Außerdem werden die Nutzdaten in 16 Byte Blöcke geteilt, welche getrennt verschlüsselt werden. Die Blockgröße ist im Gegensatz zur Schlüssellänge nicht variierbar. Der Algorithmus sieht die Möglichkeit zwar vor, sie wurde jedoch nicht im AES standardisiert.

[Wik14a] [ST01a, S. 7ff]

Betriebsmodi

Für Blockchiffren (teilweise auch für Stromchiffren) wurden im Laufe der Jahre verschiedene sog. Betriebsmodi entwickelt. Ein Betriebsmodus stellt die Art und Weise dar, in der eine Blockchiffre verwendet wird. Sie wurden eingeführt, da die Verwendung einer Blockchiffre in ihrer reinen Form problematisch werden kann. Zum einen gehört dazu die Tatsache, dass die Eingabe der Blocklänge entsprechen muss und ein Padding Gefahren für die Sicherheit mitbringen kann. Hinzu kommt, dass eine Chiffre bei gleichem Schlüssel und gleicher Eingabe immer das selbe Chifftrat erzeugt, was kryptographische Analysen erleichtert. Der einfachste denkbare Betriebsmodus nennt sich Electronic Codebook Mode. Hier wird die Nachricht in Blöcke von gleicher Blocklänge zerteilt und jeweils einzeln verschlüsselt. Dieser löst keines der oben genannten Probleme, ermöglicht es aber die Nachricht parallel zu verschlüsseln, also einen Geschwindigkeitsvorteil auf rechenstarken Systemen zu erzeugen. Weitere Betriebsmodi erzeugen aus einer zugrundeliegenden Blockchiffre eine Stromchiffre oder machen Chifftratblöcke von vorangegangenen abhängig um dafür zu sorgen, dass die Eindeutigkeit von Chifftratblöcken aufgehoben wird. Im Folgenden soll einer dieser Betriebsmodi, der Counter Mode, näher erläutert werden.

[Sch06, S. 223ff.] [Inf14b, S. 23f.] [SSP08, S. 88-94]

Counter Mode

Im Counter Mode wird nicht der Klartext selbst mittels Algorithmus und Schlüssels chiffriert, sondern mittels eines (im besten Fall zufällig) gewählten Initialisierungsvektor (Nonce, number only used once) und eines Zählers (Counter) eine Bytefolge in Blocklänge gebildet. Diese wird dann mittels des Schlüssels chiffriert und per XOR Operation mit einem der Blocklänge entsprechen Teil des Klartextes verknüpft.

Danach wird der Zähler um eins erhöht, auf dieselbe Weise wird mit dem Rest der Nachricht fortgefahren. Wichtig zu verstehen ist hierbei, dass der Initialisierungsvektor nicht Teil des Geheimnisses ist (welches ausschließlich der Schlüssel darstellt, siehe Kerckhoffs' Prinzip [SSP08, S. 38]), sondern offen übertragen werden kann. Er dient ausschließlich der Diversifikation gleicher Klartexte und dem Auffüllen einer Blocklänge. Durch den Counter Mode ist es möglich, Blockchiffren als Stromchiffren zu betreiben, da Klartexte beliebiger Länge mit den erzeugten Chiffrebytes bitweise verknüpft werden können. Vorteil bei diesem Modus ist auch, dass Chiffre im Voraus errechnet werden können, wodurch sich unter Umständen Zeitvorteile ergeben. Fehlerhaft übertragene Bits werden eins zu eins fehlerhaft entschlüsselt, weitere Teile der Nachricht sind nicht betroffen.

[ST01b] [LRWXX] [Sch06, S. 243f] [SSP08, S. 93f]

Asymmetrische Chiffren

Zur Vollständigkeit werden hier die asymmetrischen Chiffren genannt. Sie verwenden, im Gegensatz zu den symmetrischen Chiffren, zwei unterschiedliche Geheimnisse (Schlüssel) zum ver- und entschlüsseln von Botschaften. Dies löst das Problem des Schlüsselaustausches, der sonst immer über einen getrennten sicheren Kanal erfolgen müsste. Das Geheimnis zum Verschlüsseln kann offen verteilt werden (der sog. public key), während das Geheimnis zum entschlüsseln (private key) niemals herausgegeben wird. Sie haben allerdings den Nachteil mit deutlich größeren Schlüsseln zu arbeiten und sehr viel langsamer zu sein.

[Sch06, S. 525] [SSP08, S. 109]

2.3.2 Integrität

Um die Integrität von Botschaften festzustellen werden üblicherweise Hashingverfahren verwendet. Hashing wird zum Beispiel auch verwendet um sicher zu stellen, dass Botschaften korrekt übertragen wurden, also keine Übertragungsfehler entstanden sind. Im Falle der Integritätssicherstellung gegen gezielte Angriffe werden allerdings besondere kryptographische Hashfunktionen verwendet.

[SSP08, S. 15] [Sch06, S. 2]

Kryptographisches Hashing

Ein sicherer Hash Algorithmus ist in erster Linie eine Hashfunktion wie jede andere, also ein Algorithmus, der Prüfsummen auf Eingabewerte beliebiger Länge berechnet. Prüfsummen fanden schon vor der Kryptographie Einsatz in der Informatik, vor allem um die korrekte Übertragung von Daten zu gewährleisten. Ein gutes Beispiel hierfür bietet der CRC (Cyclic Redundancy Check). Wenn aus einem Bitstrom mittels CRC eine Prüfsumme berechnet wurde, kann anhand dieser nach der Übertragung der Daten relativ sicher festgestellt werden, ob alle Daten korrekt übertragen wurden, da der CRC Wert sich durch kleinste Veränderungen im Bitstrom stark verändert. Im Falle der kryptographischen Hashfunktion ist diese Eigenschaft ebenso wichtig, allerdings kommen in diesem Umfeld weitere Anforderungen hinzu: Eine kryptographische Hashfunktion sollte eine Einwegfunktion sein. Das bedeutet, dass es nicht möglich sein darf, aus einem berechneten Hashwert die ursprünglichen Daten zu rekonstruieren. Die meisten Hashfunktionen stellen dies sicher, indem der Hashwert kürzer als die Eingabedaten ist. Durch diese Kompression ist es nicht mehr möglich die ursprünglichen Daten zu ermitteln. Eine weitere Anforderung ist, dass eine kryptographische Hashfunktion kollisionsresistent sein muss. Bei einer kollisionsresistenten Hashfunktion ist es nicht möglich in effektiver Zeit zwei unterschiedliche Datenströme zu berechnen die den selben Hashwert erzeugen [EGT14, S. 4]. Aus dieser Anforderung ergibt sich auch, dass es nicht möglich ist einen Datenstrom derart zu bearbeiten, dass er einen gewünschten Hash produziert. Diese Anforderungen erfüllt CRC nicht, dafür aber kryptographische Hashfunktionen wie die in SHA standardisierten. Dagegen sind diese meist deutlich komplexer (rechenintensiver) und die erzeugten Hashes länger. Der eigentliche Sinn von kryptographischen Hashfunktionen liegt vor allem in zwei Anwendungsfällen:

Sicherstellung von Integrität

Um die Integrität eines übertragenen Bitstroms zu garantieren, kann der Hashwert zusätzlich sicher übertragen werden (üblicherweise asymmetrisch verschlüsselt). Nach dem Erhalt der Nachricht, kann der Empfänger anhand des Hashes überprüfen, ob sie unverändert übertragen wurde. Hier liegt auch der Grund der Kollisionssicherheit: Es ist effektiv nicht möglich eine neue sinnvolle Nachricht mit einem manipulierten Inhalt zu erzeugen, die den selben Hash aufweist.

Beweis: Geheimnisbesitz

Will eine Person einer anderen beweisen im Besitz eines Geheimnisses zu sein, vertraut aber der Übertragung nicht und möchte es daher nicht preisgeben, kann sie einen sog. Salt (Eng. Salz, von versalzen) erzeugen und diesen mit dem Geheimnis zusammen hashen. Dieser Hash inklusive des Salt wird verschickt und der Empfänger kann seinerseits das Geheimnis und den Salt hashen. Ist der anderen Person das Geheimnis bekannt, müssen die Hashes sich gleichen. Der Salt soll in diesem Fall verhindern, dass ein eventueller Angreifer den Hash des reinen Passwortes nicht aufzeichnen kann und sich mit dem Besitz des Hashes als Kenner des Geheimnisses ausgibt. Außerdem verhindert der Salt, dass Passwortlisten (sog. Rainbowtables) von gehashten Geheimnissen (in diesem Fall : Passwörtern) vorrausberechnet und mit gestohlenen Hashes abgeglichen werden können. Dieses Verfahren wird zum Beispiel verwendet, wenn Server Passwörter für den Login speichern. So müssen sie die Passwörter nicht im Klartext speichern und der Schaden bei einem Einbruch wird minimiert.

[SSP08, S. 31ff] [Sch06, S. 35f]

SHA2-256

SHA (Secure Hash Algorithm) ist eben so wie der AES eine Standardisierung. In diesem Falle allerdings die Standardisierung eines sicheren Hash Algorithmus. SHA2 ist der Nachfolger von SHA1 und ermöglicht die Erzeugung von sicheren Hashes von 224, 256, 384 oder 512 Bit Länge. Der Algorithmus dieser verschiedenen Versionen ist grundsätzlich immer derselbe, allerdings unterscheiden sich die Anzahl der berechneten Runden und die Länge der Initialwerte. Zum Teil (256 zu 224) wird auch einfach der letzte Teil der Ausgabe ignoriert und damit eine geringere Länge des Hashwertes erreicht.

[Inf14b, S. 5f][ST12, S. 3, 7f, 10f]

3 Technische Analyse/Umsetzung

3.1 Cloud-Anbieter

Der Markt für Cloud Computing in Form von IaaS befindet sich in einem stetigen Wandel und einer raschen Weiterentwicklung. Aus diesem Grund muss die Wahl des Anbieters sehr sorgfältig und unter Betrachtung aller Gesichtspunkte getroffen werden. Die aktuelle Studie "Magic Quadrant for Cloud Infrastructure as a Service" von Gartner [LTG⁺ 14] vergleicht die Vorzüge und Nachteile der derzeit angebotenen Produkte unter Betrachtung aller gängigen Anwendungsfälle von IaaS. Hierzu gehören beispielsweise:

- Entwicklung und Testverfahren
- Produktionsumfeld (inkl. geschäftskritischer Aufgaben interner und kundenorientierter Anwendungen)
- Automatisierung
- Wiederherstellung im Sonderfall
- Einzelanwendungsfälle und virtuelle Rechenzentren
- Entwurfsmuster für native Cloud Anwendungen und Unternehmensanwendungen

3.1.1 Magisches Quadrat



Abbildung 4: Magisches Quadrat [LTG⁺ 14]

- Marktführer (Leaders):

Der Marktführer-Quadrant stellt Anbieter dar, die über ein für den strategische Einsatz geeignetes Angebot verfügen und ehrgeizige Ziele verfolgen. Sie sind zwangsläufig nicht die besten Anbieter für spezielle Bedürfnisse und bedienen möglicherweise nicht alle Anwendungsfälle, decken aber ein breites Spektrum davon ab. Marktführer besitzen eine Erfolgsbilanz über Auslieferung, bedeutende Marktanteile und viele referenzierbare Kunden.

- Herausforderer (Challengers):

In dieser Studie befindet sich keiner der Anbieter im Herausforderer-Quadrant. Im Allgemeinen sind Herausforderer in der Lage einige Marktanforderungen zu befriedigen. Sie liefern einen guten Service, welcher auf eine bestimmte Menge von Anwendungsfällen ausgerichtet ist und verfügen über eine Erfolgsbilanz der Auslieferung. Herausforderer können sich jedoch nicht schnell genug an die Nachfrage auf dem Markt anpassen.

- Visionäre (Visionairs):

Visionäre haben eine ehrgeizige Vision von der Zukunft und tätigen erhebliche Investitionen zur Entwicklung von einzigartigen Technologien. Zu ihnen zählen neue Marktteilnehmer oder bestehende Unternehmen, die in einen neuen Markt vorstoßen möchten. Ihre Dienste befinden sich noch im Entstehungsprozess, sie verfügen jedoch über eine Vielzahl an Möglichkeiten zur Entwicklung. Obwohl sie eventuell viele Kunden besitzen, reicht das aktuelle Produkt nicht aus um ein breites Spektrum von Anwendungsfällen zu bedienen.

- Nischenspieler (Niche Players):

Nischenspieler sind womöglich ausgezeichnete Anbieter für Anwendungsfälle auf die sie sich spezialisiert haben, können aber bei weitem nicht alle Anwendungsfälle abdecken. Sie können in diesen Markt neu eingetreten sein oder noch nicht über ausreichend signifikante Marktanteile verfügen. Nischenspieler können Marktführer in einem Gebiet abseits von IaaS sein, befinden sich aber zum aktuellen Zeitpunkt für Cloud-IaaS in relativ frühen Stadien. Je spezifischer die Bedürfnisse, desto wahrscheinlicher ist es das passende Produkt in diesem Quadranten zu finden.

[LTG⁺14]

3.1.2 Entscheidungsgrundlage Cloud-Anbieter

In Abbildung 4 **Magisches Quadrat [LTG⁺14]** ist ersichtlich, dass die Top-Unternehmen Amazon (Amazon Web Services) und Microsoft (Microsoft Infrastructure Services) bei den aktuellen Marktführern einzuordnen sind, während sich Google mit seinem IaaS-Angebot Google Cloud Storage im Quadrant der Visionäre befindet.

Die Strategie der Google Cloud Plattform basiert auf dem Konzept anderen Unternehmen eine Leistungsfähigkeit anzubieten, wie Google sie selbst verwendet. Dies wird durch Nutzung der innovativen internen technischen Möglichkeiten erreicht. Obwohl Google erst im Dezember 2013 in den IaaS-Markt vorgedrungen ist (seit 2008 Google App Engine als PaaS), befindet sich die angebotene Lösung bereits unter den Visionären. Das liegt unter anderem daran, dass sich Google auf seine Fähigkeiten beschränkt, anstatt diese von Grund auf neu zu strukturieren. Daher wird es auf lange Sicht in der Lage sein das Angebot schneller als seine Konkurrenten voranzubringen. Google verfügt über eine umfassende Vision und ausreichend Erfahrung darüber, wie native Cloud-Anwendungen über einen Lebenszyklus hinweg entwickelt und verwaltet werden. Die Vorstellung von fließenden Grenzen zwischen IaaS und PaaS wird Anwendern in absehbarer Zeit einen Kompromiss zwischen Kontrolle und automatisierter Verwaltung bieten. Die gesamte Unternehmung Google verfügt über eine Vielzahl von Rechenzentren, enorme infrastrukturelle Kapazitäten und ein eigenes globales Hochleistungsnetzwerk. Da der IaaS-Zweig Google Compute Engine (GCE) lediglich geringe Mehrkosten für Google darstellt, kann es den Preis trotz hoher Leistungsfähigkeit aggressiv gestalten. Dennoch wird sich Google nicht durch den Preis, sondern durch seine Plattform und zusätzliche Verwaltungsfunktionen von den anderen IaaS-Anbietern unterscheiden. Aktuell ist Google Cloud Storage jedoch ein relativ neues Produkt, sodass noch keine operative Erfolgsbilanz gezogen werden kann. Hinzu kommen kleinere Störungen bis zur Freigabe der allgemeinen Verfügbarkeit von GCE. Eine der größten Herausforderungen besteht darin, das Vertrauen der Unternehmen zu gewinnen und den Support auszubauen. Allerdings wird Google bereits heute als künftiger Marktführer für IaaS wahrgenommen, nicht zuletzt wegen der gewohnten Unterstützung durch Software-Unternehmen und Entwickler. Diesen Vorteil konnten wir uns im Rahmen der Projektarbeit zunutze machen und auf Entwicklungsmöglichkeiten von Google, bspw. durch die Verwendung der Schnittstelle gsutil, zurückgreifen. Da das Ziel dieser Arbeit ebenfalls in einem visionären Qua-

dranten einzuordnen wäre, haben wir uns für Google Cloud Storage und gegen Amazon Web Services und Microsoft Infrastructure Services entschieden.

[LTG⁺14]

3.2 Google Cloud Storage

Google Cloud Storage ermöglicht es Daten auf der Google-Infrastruktur mit hoher Zuverlässigkeit, Verfügbarkeit und Leistung abzulegen. Der Aufbau kann in drei Ebenen [Inc14d] eingeteilt werden:

- Objects:

Objects sind einzelne Daten, die in Google Cloud Storage gespeichert werden. Sie bestehen aus zwei Komponenten, Objektdaten und Objekt-Metadaten. Objektdaten sind in der Regel Dateien, die gespeichert werden sollen - Objekt-Metadaten eine Sammlung von Objekteigenschaften. Jedes Object gehört zu genau einem Bucket und kann nicht zwischen den Buckets geteilt werden.

- Buckets:

Buckets sind grundlegende Container für Objects. Diese befinden sich innerhalb eines Buckets des zugehörigen Projects. Darin können Objects organisiert und deren Zugriff verwaltet werden. Im Gegensatz zu einer Struktur aus Verzeichnissen und Ordnern ist keine Verschachtelung möglich. Ein Bucket gehört fest zu einem Project und kann nicht von anderen verwendet werden.

- Projects:

Jeder Bucket ist genau einem Project zugeordnet. Ein Project besteht aus einer Gruppe von Benutzern, mehreren APIs, Authentifizierung, Überwachungsoptionen und wird mit einem sog. "Billing Account" einzeln abgerechnet. Ein Project kann eine beliebige Anzahl an Buckets enthalten.

3.3 Schnittstelle Google Cloud Storage: gsutil und boto

gsutil [Inc14a] ist eine Python-Anwendung, welche einen konsolenbasierten Zugriff der gängigen Betriebssysteme Linux/Unix, Mac OS oder Windows auf Google Cloud Storage ermöglicht. Voraussetzung ist ein installiertes Python in der Version 2.6.x oder 2.7.x, Python 3.x ist dazu derzeit inkompatibel. Dabei nutzt gsutil die Notation der Standardbefehle von Linux, welche direkt in einer Konsole oder durch eine

Anwendung aufgerufen werden können. Damit gsutil als Schnittstelle zum eigenen Google Cloud Storage verwendet werden kann, muss dieses initial durch einen Authentifizierungscode mit dem zugehörigen Account verknüpft werden. Nachfolgend sind die für das Projekt relevanten Befehle aufgelistet (ein Ausführen von gsutil mit einem Python-Interpreter und eine korrekte Authentifizierung ist vorausgesetzt):

- Auflisten: **ls**
gsutil ls [Optionen] gs://<Bucket>
- Kopieren: **cp**
Upload: gsutil cp [Optionen] <Quellpfad lokaler Datei> gs://<Bucket>
Download: gsutil cp [Optionen] gs://<Bucket>/<Dateiname> <Zielpfad lokaler Datei>
- Entfernen: **rm**
gsutil rm [Optionen] gs://<Bucket>/<Dateiname>

Desweiteren verarbeitet gsutil Befehle für das Erstellen und Löschen von Buckets, Verschieben, Umbenennen und Editieren von Objekten sowie zur Rechteverwaltung.

boto [Inc14b] ist eine OpenSource Python-Bibliothek, die als Schnittstelle zu Google Cloud Storage dient. Das zugehörige GCS-Oauth2-boto-Plugin ist ein Plugin zur Authentifizierung mit OAuth 2.0 Anmeldeinformationen. Neben Google verwendet es auch Amazon Web Services als Schnittstelle.

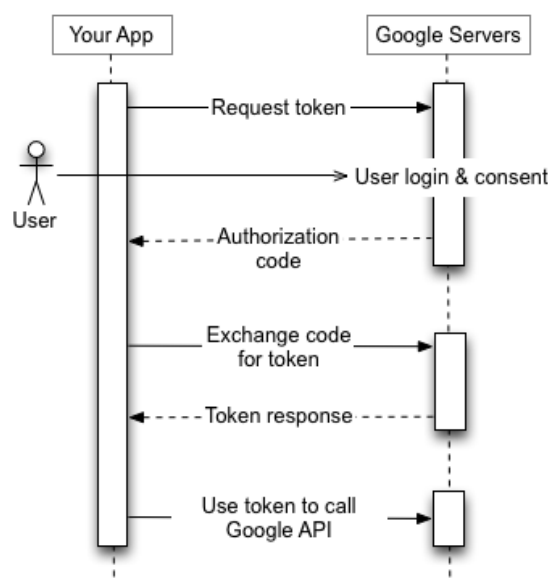


Abbildung 5: OAuth 2.0 [Inc14c]

In Abbildung 5 wird das Authentifizierungsverfahren OAuth 2.0 an den Google-Servern dargestellt. Ein Anwender meldet sich im System an, erhält daraufhin einen Authentifizierungscode, bestätigt diesen und kann anschließend mit jeglicher, auf die Google APIs zugreifender, Anwendung Google Cloud Storage verwenden. Die Informationen werden in einer *.boto Datei gespeichert und bleiben somit automatisierend für weitere Zugriffe erhalten. [Inc14c]

Neben gsutil verfügt Google Cloud Storage über eine **XML API** für HTTP-Anfragen von Web-Services, eine **JSON API** in Version v1 und weitere **Cloud Storage Tools**.

3.4 Eclipse IDE for Java Developers

Eclipse ist eine OpenSource Entwicklungsumgebung zur gestützten Entwicklung von Software. Die IDE for Java Developers beinhaltet zusätzlich Pakete und Tools, die eine Programmierung in der Programmierhochsprache Java erleichtert. Interessant hierbei ist, das Eclipse selbst für Java und auch in Java entwickelt wurde. Es ist also genau so vielseitig nutzbar wie die Anwendungen die damit entwickelt werden. Durch Plugins kann die Funktionalität von Enclipse erweitert werden, etwa um UML Diagramme aus Code oder umgekehrt aus UML Diagrammen Code zu erzeugen oder um die Entwicklung in anderen Sprachen zu ermöglichen. Im Zuge dieses Projekts wurde Eclipse in der Version **Kepler Service Release 2, Build id: 20140224-0627**, verwendet - siehe Abbildung 6.

3.5 Versionsverwaltung: git

git ist eine Software unter der OpenSource-Lizenz GNU GPLv2, welche zur verteilten Versionsverwaltung von Dateien eingesetzt wird. Seinen Ursprung fand git in der Verwaltung von Quellcode zur Entwicklung des Linux-Kernels. In diesem Projekt fand Version 2.1.2, unter Einsatz von *Atlassian SourceTree* als Interface, Verwendung - siehe Abbildung 7. Dies machte eine gleichzeitige, standortunabhängige Entwicklung und Synchronisierung von Änderungen des Prototyps möglich.

[Cha09]

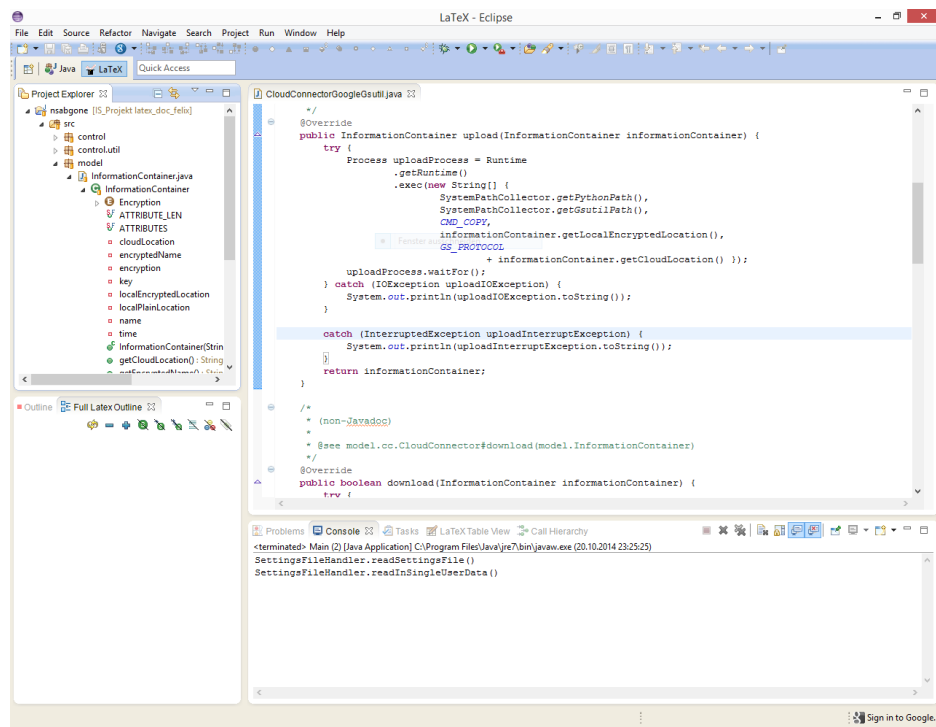


Abbildung 6: Eclipse

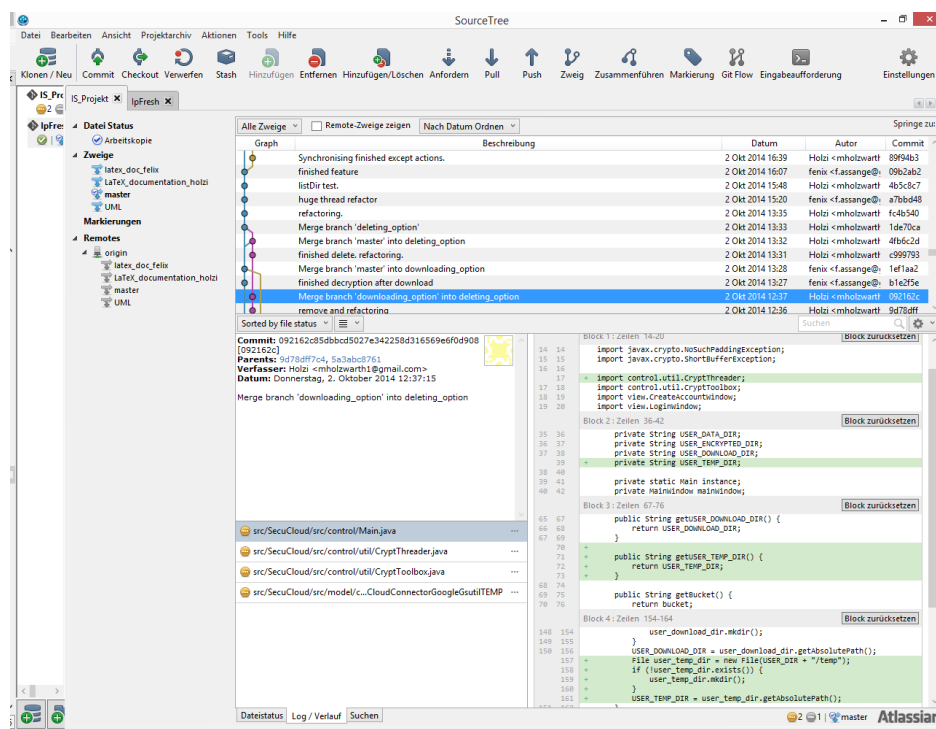


Abbildung 7: Atlassian SourceTree

3.6 Kryptographie

3.6.1 Notwendige Verfahren

Zur Umsetzung des Prototypen werden zwei kryptographische Verfahren benötigt: Verschlüsselung und Hashing. Die Verschlüsselung dient der eigentlichen Umsetzung der Sicherheit, nämlich die Daten Verschlüsselt abzulegen. Das Hashing wird benötigt um auf der einen Seite die Accounts auf dem Zielsystem abzusichern, also verschiedene Benutzerkonten zu ermöglichen. Auf der anderen Seite sichert Hashing ab, dass die Daten von dritten auf dem Cloudspeicher manipuliert werden. Diese Möglichkeit lässt sich durch Verschlüsselung und andere kryptographische Verfahren zwar auf einem fremden Datenspeicher nicht verhindern, aber durch Hashing sicher erkennen.

3.6.2 Entscheidungsgrundlage: BSI

Das Bundesamt für Sicherheit in der Informationstechnik ist auf deutscher Seite eine Bundesbehörde, welche sich mit der IT-Sicherheit beschäftigt. Jährlich werden vom BSI unter anderem Kataloge herausgegeben, wie Unternehmen und Ämter in Deutschland Daten schützen sollten, inklusive Prognosen auf welche Zeitspanne diese Mechanismen als sicher zu betrachten sind. Hierunter fallen zwar auch Absicherung, Methoden und die Beratung von Unternehmen und Ämtern, der Augenmerk in diesem Projekt liegt allerdings zentral auf den Vorgaben des BSI bzgl. kryptographischer Algorithmen.

Für Blockchiffren empfiehlt das BSI zurzeit die Standards AES128, AES192, AES256. Mit dem AES128 liegt das Projekt also zurzeit an der unteren Grenze, allerdings ist hier zu berücksichtigen, dass die untere Grenze des BSI als auf Jahre äußerst sicher anzusehen ist. Außerdem hätte eine höhere Schlüssellänge eine deutlich erhöhte Rechenzeit zur Folge, was vor allem bei großen Datenmengen und schwachen Rechnern in unserem Fall inakzeptabel wäre. Die empfohlenen Betriebsmodi für Blockchiffren sind laut BSI Dokumenten der Galois-Counter-Mode, das Cipher-Block-Chaining sowie der Counter Mode. Auch hier erfüllt die Applikation also alle Anforderungen des BSI. Bei den Hashfunktionen lautet die Empfehlung des BSI für den Zeitraum nach 2015 SHA256, SHA384, SHA512 oder SHA512/256. Das bedeutet für dieses Projekt, dass auch das Hashing vom BSI zugelassen und sogar für die Zukunft empfohlen ist. [EGT14][Inf14b]

4 Ergebnisse

Im Folgenden Kapitel wird der entwickelte Prototyp näher erläutert, zunächst durch eine Beschreibung der Klassendefinitionen und der Architektur. Nachfolgend wird das zugrunde liegende kryptographische System erläutert und das graphische Interface sowie die Funktionalität aus Benutzersicht beschrieben.

4.1 Architektur

Der Applikation zugrunde liegt eine Architektur, die an das Model-View-Control Entwurfsmuster angelehnt ist. Sie unterteilt sich in drei packages (Abgrenzung von Programmteilen in Java) sowie zwei subpackages (packages, die in anderen enthalten sind). Im Folgenden werden diese sowie die enthaltenen Klassen näher skizziert. Eine genaue Auflistung der Methoden und Klassenvariablen kann den UML Diagrammen bzw. der Dokumentation entnommen werden.

4.1.1 Package: control

Das Package control enthält die Programmlogik, hierzu gehören Ver- und Entschlüsselung, Benutzerverwaltung und Koordination der Datei-Informationen.

Klasse: Main

Die Klasse Main stellt das Herz der Applikation dar: In ihr liegen applikationsweit verwendete Definitionen (public static final Variablen) und Methoden, die von den Views aus aufgerufen werden um auf Benutzerinteraktionen zu reagieren. Die meisten anderen Klassen werden hier instanziiert, die temporären Benutzerinformationen hinterlegt und mit get() und set() Methoden für andere Module nutzbar gemacht. Sie stellt also den Knotenpunkt dar, der die Module der Applikation verbindet. Main ist an der Programmlogik selbst nur in Form des Programmstartes und der Delegation anderer Aufrufe beteiligt. Kalkulationen oder Dateioperationen finden hier nicht statt. Sie ist mit dem Singleton Entwurfsmuster implementiert. So ist sichergestellt, dass alle Programmteile auf dieselbe Instanz von Main zugreifen, ohne dass eine entsprechende Referenz an die anderen Klassen übergeben werden müsste.

Klasse: FileListHandler

Auch der FileListHandler ist ein Singleton, kommt also applikationsweit nur in einer Instanz vor und wird von verschiedenen Programmteilen direkt aufgerufen. Im Kern besteht er aus einem Vector aus InformationContainer Instanzen (siehe Package model - InformationContainer), er ist also das temporäre (während des Programmblaufes) Speichermodul für Informationen über die in der Cloud liegenden Dateien. Zusätzlich besitzt er Methoden für Operationen auf diesem Vector, etwa neue Dateien hinzuzufügen oder aber Dateien anhand bestimmter Schlüssel zurück zu geben.

Klasse: SettingsFileHandler

Der SettingsFileHandler verwaltet die Datei settings.cfg in der Informationen über in der Applikation registrierte Benutzer gespeichert sind. Er erzeugt sie bei Bedarf, liest eine bestehende ein und macht die Benutzerinformationen anderen Programmteilen zugänglich oder schreibt neue Benutzer hinein.

Klasse: SystemInformationCollector

Der SystemInformationCollector sammelt, wie der Name suggeriert, Informationen über das System auf dem die Applikation gestartet wird. Zurzeit ist dies ausschließlich die Information über das zugrunde liegende Betriebssystem, es sind jedoch für die Zukunft weitere Informationen vorgesehen, etwa die Anzahl der Prozessorkerne um mittels Verteilung der Rechenlast schneller zu ver- und entschlüsseln.

Klasse: SystemPathCollector

Der SystemPathCollector macht anderen Modulen Informationen darüber zugänglich, wo sich weitere, für die Applikation notwendige Programme befinden. Zurzeit sind dies Python und Gsutil.

Klasse: ThreadInstanceCreator

Der ThreadInstanceCreator ist für das Threading zuständig. Im Laufe der Entwicklung stellte sich heraus, dass vor allem ver- und entschlüsselt, sowie der Up- und Download großer Dateien den Programmablauf erheblich behindern. Daher war es notwendig diese Operationen in einen weiteren Prozess auszulagern. Das übernimmt der ThreadInstanceCreator. Für jede neue Verschlüsselungs-, Up- bzw. Download- und Entschlüsselungsoperation wird eine neue Instanz dieser Klasse als neuer Thread

erzeugt. Möglich wird das über das von Java zur Verfügung gestellte Interface Runnable, welches die Klasse implementiert (implements). Eine neue Instanz bekommt im Konstruktor einen InformationContainer sowie einen Befehl (enum), wie dieser zu verarbeiten ist, übergeben und arbeitet Ver- bzw Entschlüsselung und Up- und Download selbstständig ab.

4.1.2 Subpackage: util

Das Subpackage util gehört zum Controller und besitzt Methoden die softwareweit unabhängig sind, also keine initialisierten Instanzen benötigen, etwa weil keine Klassenvariablen zur Speicherung von Informationen notwendig sind. Alle Methoden dieser Klassen sind statisch und können zu jeder Zeit von jedem Programmteil verwendet werden. Unter anderem liegen hier die Verschlüsselungsfunktionen sowie Hilfsfunktionen zur Umwandlung von Werten.

Klasse: CryptToolbox

Die CryptToolbox bietet Ver-/Entschlüsselung von Dateien oder Bytearrays, Hashing sowie die Generierung von kryptographisch sicheren Zufallszahlen. Auch werden hier die ThreadInstanceCreator Instanzen erzeugt, die dann Funktionen der CryptToolbox verwenden.

Klasse: SupportFunctions

Eine Hilfsklasse um Integer in Bytes zu konvertieren sowie, zur Erleichterung des Debuggings, Bytearrays auf der Konsole auszugeben.

4.1.3 Package: model

Das Package model ist für die Daten selbst zuständig. Es definiert in welcher Weise Informationen gespeichert und, nach Beendigung des Programmes, lokal im Speicher des Computers abgelegt werden.

Klasse: InformationContainer

Ein InformationContainer stellt in der Programmlogik eine in der Cloud liegende Datei dar. Er besitzt Klassenvariablen in denen Name, Schlüssel, Dateipfad und weitere Informationen gespeichert sind sowie get() und set() Funktionen, um auf und mit die-

sen Informationen zu arbeiten.

Klasse: InformationContainerStorer

Der InformationContainerStorer verarbeitet die Speicherung bei Beendigung der Applikation sowie das Laden der InformationContainer nach Programmstart. Er verwaltet die entsprechende Datei (state.cfg) und stößt an, dass die InformationContainer vor dem Speichern in Bytearrays umgewandelt und mittels des Benutzerpasswortes verschlüsselt werden.

4.1.4 Subpackage: cc

Das Subpackage cc gehört zum model und verwaltet die Speicherung der Daten in der Cloud. Darin wird das Interface für die Schnittstelle zu Cloudspeichern definiert. In diesem Prototyp ist dies gsutil zur Anbindung von Google Cloud Storage.

Interface: CloudConnector

Das Interface CloudConnector definiert, wie eine Schnittstelle zu einem Cloudanbieter aufgebaut sein muss, also welche Methoden sie besitzen muss und welche Übergabeparameter bzw. Rückgabewerte diese haben.

Klasse: CloudConnectorGoogleGsutil

Der CloudConnectorGoogleGsutil ist eine Implementierung des Interfaces CloudConnector zur Anbindung des Google Cloud Storage mittels der Python Schnittstelle gsutil und stellt die in CloudConnector definierten Methoden für Operation auf dem Cloudspeicher bereit.

4.1.5 Package: view

Das Package view enthält alle Teile des graphischen Interfaces. In diesem werden alle Fenster definiert. Die views verfügen über lediglich zwei Schnittstellen zur restlichen Applikation: Die Main Klasse, welche Aktionen (klicks) auf den Views entgegen nimmt und daraufhin andere Programmteile steuert, sowie die Klasse FileListHandler, welche dem MainWindow die in der Tabelle aufgelisteten Informationen (in der Cloud befindliche Dateien, InformationContainer) liefert. Eine genaue Beschreibung, wie das GUI aufgebaut ist, befindet sich in **4.5 GUI und Funktionalitäten**

Klasse: MainWindow

Das MainWindow ist die Hauptoberfläche der Applikation. Hier wird eine Liste der in der Cloud befindlichen Dateien angezeigt, es können weitere zum Upload ausgewählt oder aus der Cloud gedownloaded bzw. gelöscht werden.

Klasse: CreateAccountWindow

Das CreateAccountWindow erscheint, wenn der Benutzer entweder noch keinen Account angelegt hat oder aber wenn die Schaltfläche Create im LoginWindow ausgewählt wird. Hier muss vom Benutzer sein gewünschter Benutzername, sein Passwort sowie eine Wiederholung, zur Vorbeugung eines Tippfehlers, eingegeben werden.

Klasse: DeleteWindow

Das DeleteWindow erscheint, wenn die Applikation beim Start feststellt, dass im lokalen Speicher InformationContainer hinterlegt sind, die kein Pendant mehr in der Cloud besitzen. Dies kann geschehen, wenn sie manuell per Webinterface gelöscht wurden oder der Upload fehl schlug. Der Benutzer sieht hier, um welche Datei es sich handelt (Klarname, ID in der Cloud) und kann selbst entscheiden ob der InformationContainer zur Sicherheit behalten werden sollte, oder entfernt werden kann.

Klasse: LoginWindow

Das LoginWindow fordert den Benutzer auf, seine Benutzerdaten einzugeben um sich in der Applikation einzuloggen. Dies sind sein Benutzername, das dazugehörige Passwort sowie der Bucket auf dem der Benutzer arbeiten möchte. Dies ist zurzeit sehr Google Storage spezifisch, allerdings ist die Applikation aus dem Grund modularisiert aufgebaut. Hier können später weitere Cloud-Anbieter mit alternativen LoginWindows angebunden werden.

Klasse: NonEditableJTable

Die NonEditableJTable ist eine Hilfsklasse. Sie erbt vom DefaultTableModel (Java-Klasse die die Methoden von JTables definiert) und überschreibt die Funktion isCellEditable(int row, int column), sodass sie immer false zurück gibt. Im MainWindow erhält die angezeigte Tabelle dieses Model, damit der Benutzer die Tabelle nicht manuell bearbeiten kann, was zurzeit in der Applikation keinen Sinn machen würde und Fehler im Programmablauf erzeugen könnte.

Klasse: NotificationWindow

Das NotificationWindow wird angezeigt, wenn der Benutzer die Menüpunkte Help oder About in der Menüleiste im MainWindow betätigt. Entsprechend zeigt das NotificationWindow mit der Help Option eine Erklärung zur Bedienung der Applikation. Wenn About ausgewählt wurde, stehen dort Informationen über Entwickler und Version.

4.2 UML-Diagramme

Die UML Diagramme befinden sich im digitalen Anhang, Ordner UML. In der Datei ClassRelations.jpg findet sich eine Übersicht aller Klassen und deren Beziehungen zu einander. Attribute und Methoden wurden hier weg gelassen, da sie der Übersichtlichkeit abträglich wären. Des Weiteren gibt es für jedes Package bzw. Subpackage ein weiteres UML mit den jeweiligen Klassen, deren Beziehungen sowie Attributen und Methoden. Genauere Beschreibungen der einzelnen Methoden, Klassen und Attribute finden sich in der Dokumentation im Ordner JavaDoc im digitalen Anhang.

4.3 Anwendungsfälle

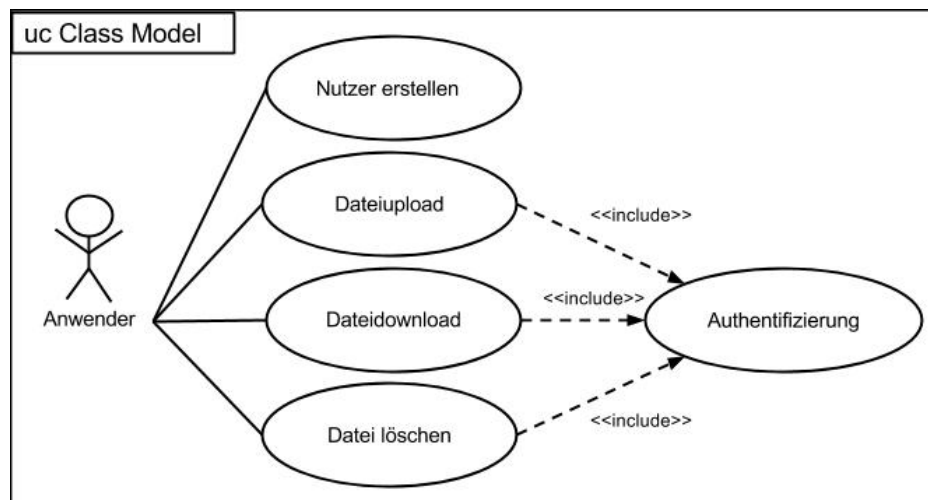


Abbildung 8: Anwendungsfalldiagramm

4.3.1 Begriffslexikon Anwendungsfälle:

- **Benutzer:**
Die Rechte und Möglichkeiten des Benutzers, sofern diese vorher beim Anbieter vergeben wurden, begrenzen sich auf das Ablegen lokaler und das Herunterladen bzw. Löschen von in der Cloud gespeicherten Dateien.
- **Authentifizierung:**
Der Login erfordert den Benutzernamen sowie das zugehörige Passwort.
- **Datei:**
Die hochzuladende Datei kann in jedem beliebigen Dateiformat vorliegen.
- **Dateiübersicht:**
Die Dateiübersicht gibt tabellarisch die in der Cloud abgelegten Dateien wieder. Dies beinhaltet den originalen Dateinamen, den kryptischen Dateinamen sowie den Zeitpunkt des Ablegens.
- **GUI:**
GUI (engl. Graphical User Interface) ist eine graphische Benutzeroberfläche, die dem Benutzer die Interaktion mit dem System vereinfacht.

4.3.2 AF-001 Benutzer erstellen:

Nummer	AF-001
Name	Benutzer erstellen
Akteure	Benutzer
Auslöser	Benutzer möchte einen Benutzer anlegen
Vorbedingung	Programm zeigte den Login-Dialog, Benutzer hat Create ausgewählt
Nachbedingung/Ziel	Ein neuer Benutzer wurde angelegt
Nachbedingung im Sonderfall	Es wurde kein Benutzer erstellt Erneute Aufforderung zur Dateneingabe
Normalablauf	<ol style="list-style-type: none"> 1. Benutzer startet Programm 2. Benutzer wählt Create im Login-Fenster 3. Benutzer gibt Benutzername ein 4. Benutzer gibt Passwort ein 5. Benutzer bestätigt das Passwort durch zweite Eingabe 6. Benutzer wurde angelegt 7. Login-Fenster wird angezeigt
Sonderfälle	<ol style="list-style-type: none"> 3a. Benutzer gibt nicht erlaubten Benutzernamen ein 4a. Benutzer gibt nicht erlaubtes Passwort ein 5a. Benutzer gibt nicht das identische Passwort ein 6a. Benutzer konnte nicht angelegt werden 7a. Create-Fenster wird erneut angezeigt

Tabelle 1: AF-001 Benutzer erstellen

4.3.3 AF-002 Authentifizierung:

Nummer	AF-002
Name	Authentifizierung
Akteure	Benutzer
Auslöser	Benutzer möchte sich anmelden
Vorbedingung	Programm zeigt den Login-Dialog, Benutzer ist im Cloud-System registriert und zum Zugriff berechtigt
Nachbedingung/Ziel	Erfolgreiche Anmeldung am Cloud-System
Nachbedingung im Sonderfall	Zugriff verweigert Erneute Aufforderung zur Anmeldung
Normalablauf	1. Benutzer startet Programm 2. Benutzer gibt Benutzernamen ein 3. Benutzer gibt Passwort ein 4. Benutzer gibt Bucket an 5. Zugangsdaten werden überprüft 6. Dateiübersicht wird angezeigt
Sonderfälle	2a. Benutzer gibt falschen Benutzernamen ein 3a. Benutzer gibt falsches Passwort ein 4a. Benutzer gibt nicht gestatteten Bucketnamen an 5a. Erneutes Laden der Loginmaske

Tabelle 2: AF-002 Authentifizierung

4.3.4 AF-003 Dateiupload:

Nummer	AF-003
Name	Dateiupload
Akteure	Benutzer
Auslöser	Benutzer möchte eine Datei in der Cloud ablegen
Vorbedingung	Benutzer hat sich erfolgreich authentifiziert Dateiübersicht wird angezeigt
Nachbedingung/Ziel	Datei wurde verschlüsselt in der Cloud abgelegt
Nachbedingung im Sonderfall	Datei kann nicht abgelegt werden
Normalablauf	<ol style="list-style-type: none"> 1. Benutzer wählt das Menü "File" aus 2. Benutzer wählt im Untermenü "Select" 3. Benutzer bewegt sich durch das eigene Dateisystem zum Speicherpfad der hochzuladenden Datei 4. Benutzer initialisiert den Dateiupload durch Auswählen der Datei und Bestätigung mit "Open" 5. Datei wird verschlüsselt 6. Datei wird hochgeladen
Sonderfälle	<ol style="list-style-type: none"> 3a. Dateisystem wird nicht unterstützt 4a. Keine Datei ausgewählt 4b. Benutzer wählt "Cancel" 5a. Datei wird nicht verschlüsselt 6a. Datei kann nicht hochgeladen werden

Tabelle 3: AF-003 Dateiupload

4.3.5 AF-004 Dateidownload:

Nummer	AF-004
Name	Dateidownload
Akteure	Benutzer
Auslöser	Benutzer möchte eine Datei aus der Cloud herunterladen
Vorbedingung	Benutzer hat sich erfolgreich authentifiziert Dateiübersicht wird angezeigt
Nachbedingung/Ziel	Datei wurde aus der Cloud heruntergeladen und entschlüsselt
Nachbedingung im Sonderfall	Datei kann nicht heruntergeladen werden Datei wird nicht entschlüsselt
Normalablauf	1. Benutzer wählt in der Dateiübersicht mit linkem Mausklick den Eintrag der entsprechenden Datei aus 2. Benutzer öffnet das Kontextmenü mit rechtem Mausklick 3. Benutzer wählt im Kontextmenü "Download" 4. Datei wird heruntergeladen 5. Datei wird entschlüsselt
Sonderfälle	1a. Kein Eintrag vorhanden 1b. Kein Eintrag ausgewählt 3a. Kontextmenü öffnet sich nicht 4a. Datei kann nicht heruntergeladen werden 5a. Datei wird nicht entschlüsselt

Tabelle 4: AF-004 Dateidownload

4.3.6 AF-005 Datei löschen:

Nummer	AF-005
Name	Datei löschen
Akteure	Benutzer
Auslöser	Benutzer möchte eine Datei aus der Cloud löschen
Vorbedingung	Benutzer hat sich erfolgreich authentifiziert Dateiübersicht wird angezeigt
Nachbedingung/Ziel	Datei wurde aus der Cloud gelöscht
Nachbedingung im Sonderfall	Datei kann nicht gelöscht werden
Normalablauf	<ol style="list-style-type: none"> 1. Benutzer wählt in der Dateiübersicht mit linkem Mausklick den Eintrag der entsprechenden Datei aus 2. Benutzer öffnet das Kontextmenü mit rechtem Mausklick 3. Benutzer wählt im Kontextmenü "Delete from cloud" 4. Delete Fenster erscheint 5. Benutzer bestätigt mit Ok 6. Datei wird aus der Cloud gelöscht 7. Dateieintrag in der Übersicht wird entfernt Schlüssel wird entfernt
Sonderfälle	<ol style="list-style-type: none"> 1a. Kein Eintrag vorhanden 1b. Kein Eintrag ausgewählt 3a. Kontextmenü öffnet sich nicht 4a. Delete Fenster erscheint nicht 5a. Benutzer wählt No 6a. Datei kann nicht aus der Cloud gelöscht werden 7a. Dateieintrag in der Übersicht wird nicht entfernt Schlüssel wird nicht entfernt

Tabelle 5: AF-005 Datei löschen

4.4 Kryptosystem

Die Verschlüsselung und das Accountmanagement basieren zurzeit ausschließlich auf den Algorithmen AES128 im CTR Modus und SHA256. Eine asymmetrische Verschlüsselung wurde in Betracht gezogen, schied jedoch aus unter der Annahme, dass der Client auf dem die Applikation läuft, als sichere Umgebung gilt. In einer späteren Weiterentwicklung ist es aber durchaus sinnvoll dies erneut zu evaluieren. Vor allem dann, wenn ein Server das Dateisystem bzw. dessen Synchronität verwaltet oder mehrere kaskadierte Verschlüsselungsebenen gebildet werden sollen, um eine erweiterte Rechteverwaltung zu ermöglichen. Die Beschreibung der Sicherungsverfahren wird nachfolgend in der Reihenfolge, in der sie von einem Benutzer ausgelöst werden, beschrieben.

4.4.1 Account anlegen

Wird die Applikation zum ersten Mal gestartet, d.h. es ist noch kein Benutzer registriert, erzeugt sie im Home-Verzeichnis des Benutzers eine eigene Ordnerstruktur, um die anfallenden Informationen zu verwalten und nach dem Schließen vorzuhalten. Nachdem der Benutzer im CreateAccountWindow seine gewünschten Benutzerdaten eingetragen hat, wird automatisch ein zufälliger Wert von 64 Byte berechnet, der sog. "Salt". Danach wird der Benutzername im Klartext in die Datei HOME/NSAb-GONE/settings.cfg eingetragen, gefolgt von einem Doppelpunkt (":") als Separator. Anschließend berechnet das Programm einen SHA256 Hashwert über ein Bytearray bestehend aus dem Salt und dem Passwort. Dieser Hash sowie der Salt werden hinter dem Doppelpunkt abgelegt. Weitere eventuell angelegte Benutzer werden auf dieselbe Weise an die Datei angehängt, die Applikation erkennt anhand des Separators und der Kenntnis über Länge von Hash und Salt, wo die Daten eines Benutzers enden und die des nächsten beginnen.

4.4.2 Login

Um sich in seinen Account einzuloggen, muss ein Benutzer seinen Benutzernamen und sein Passwort in die im LoginWindow vorgegebenen Zellen eintragen. Drückt er auf den OK Button, erstellt die Applikation ein Bytearray aus dem Passwort und dem in settings.cfg hinterlegten Salt. Dies wird gehasht und das Ergebnis mit dem

in settings.cfg eingetragenen verglichen. Stimmen die beiden überein, sind die Benutzerdaten valide und die Applikation wechselt in das MainWindow. Der Vorteil bei der Verwendung eines Salt ist in **2.3 Kryptographie** näher erläutert.

4.4.3 Dateien ablegen

Wählt der Benutzer eine Datei über das MainWindow aus, um diese in der Cloud sicher zu speichern, beginnt die Applikation zunächst damit, einen zufälligen Schlüssel von 16 Byte Länge (AES Key, 128 Bit) und einen Initialisierungsvektor (Nonce) für den CTR Modus (ebenfalls 16 Byte) zu berechnen und in einer Instanz der Klasse InformationContainer zusammen mit dem Dateinamen und weiteren Informationen zu speichern. Danach wird ein Hash über die gesamte Datei gebildet und eine temporäre Kopie angehängt, bevor diese dann selbst mit den generierten Werten verschlüsselt, der verwendete Initialisierungsvektor angehängt und sie anschließend in der Cloud abgelegt wird. Für jede Datei wird ein eigener Schlüssel berechnet, da ein Schlüssel immer unsicherer wird, je öfter er verwendet bzw. je mehr Daten damit verschlüsselt werden. Zudem ist es möglich, durch Aushändigen des Schlüssels, einem anderen Benutzer den Zugriff auf eine einzelne Datei zu gestatten.

4.4.4 Programm schließen

Wird die CLOSE Schaltfläche des MainWindow oder das X in der Ecke des Programmfensters betätigt, werden alle laufenden Verschlüsselungs- und Uploadprozesse abgearbeitet. Anschließend werden die gespeicherten InformationContainer in Bytefolgen umgewandelt und aneinander gehängt. Dieses Bytearray wird dann mittels des Benutzerpasswortes und einem weiteren zufällig generierten Initialisierungsvektor AES CTR verschlüsselt. Danach wird der Initialisierungsvektor angehängt und die Daten in die Datei HOME/NSAbGONE/<Benutzername>/data/state.cfg gespeichert. Sollte das Benutzerpasswort nicht die richtige Länge haben (genau 16 Byte, AES Schlüssellänge) wird es bei 16 Byte abgeschnitten, bzw. auf 16 Byte expandiert. Das bedeutet, es wird so oft hintereinander gehängt, bis 16 Byte erreicht oder überschritten werden. Eventuell überflüssige Zeichen werden anschließend abgeschnitten. Aus diesem Grund lohnt es sich ein ausreichend langes Passwort zu wählen.

4.4.5 Programmstart mit vorhandener state.cfg

Wird die Applikation gestartet und ein Benutzer meldet sich mit validen Daten an, wird geprüft, ob bereits eine state.cfg für diesen Benutzer existiert, also ob bereits Dateien abgelegt und entsprechend Informationen dazu gespeichert wurden. Ist dies der Fall, expandiert die Applikation das Benutzerpasswort und entschlüsselt die state.cfg. Hier wird auch klar, warum die state.cfg verschlüsselt sein muss: Es ist möglich, dass jemand die settings.cfg derart manipuliert, dass jemand sich mit einem falschen Passwort einloggen kann. Dann ist es dieser Person aber nach wie vor nicht möglich, die Dateien des eigentlichen Benutzers zu entschlüsseln, da er das echte Passwort nicht besitzt. Auch ist es zukünftig denkbar die state.cfg selbst in der Cloud abzulegen. Dann wäre es möglich von überall aus das Programm zu verwenden (etwa mitgeführt auf einem USB Stick), ohne die Informationen im Benutzerverzeichnis zu benötigen.

4.4.6 Datei herunterladen

Wählt der Benutzer eine Datei zum Download aus, wird diese zunächst heruntergeladen und in einem temporären Verzeichnis gespeichert. Mit dem hinterlegten Schlüssel und dem Initialisierungsvektor wird sie dann entschlüsselt und der Hash extrahiert. Dann wird über die Datei ein neuer Hash erzeugt und die beiden verglichen, um sicher zu stellen, dass sie nicht manipuliert wurde. Ist dies nicht der Fall wird sie in das Downloadverzeichnis abgelegt und der Benutzer kann sie verwenden.

4.5 GUI und Funktionalitäten

Nachfolgend wird die Oberfläche sowie die Funktionalität des Prototyps beschrieben.

4.5.1 Login

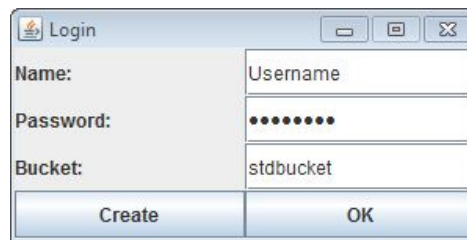
The screenshot shows a 'Login' dialog box with a title bar containing a small icon and standard window controls. The dialog has three input fields: 'Name:' with the placeholder text 'Username', 'Password:' with masked characters (dots), and 'Bucket:' with the text 'stdbucket'. At the bottom, there are two buttons: 'Create' and 'OK'.

Abbildung 9: Login

Bei jedem Systemstart öffnet sich ein PopUp-Fenster mit einer Eingabemaske für die Benutzerauthentifizierung. Existiert der eingegebene Benutzer nicht oder ist das zugehörige Passwort falsch, wird das Login-Fenster neu geladen. Mit Create kann ein neuer Benutzer angelegt werden. Desweiteren muss der Benutzer das Bucket, auf dem er arbeiten möchte, eintragen. Der Authentifizierungsvorgang beginnt durch Bestätigung von OK.

4.5.2 Account erstellen

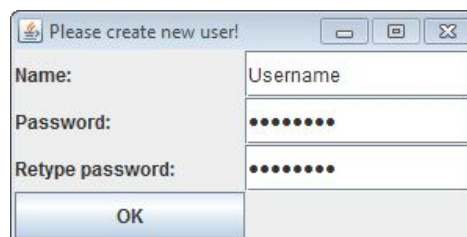
The screenshot shows a 'Please create new user!' dialog box with a title bar containing a small icon and standard window controls. The dialog has three input fields: 'Name:' with the placeholder text 'Username', 'Password:' with masked characters (dots), and 'Retype password:' with masked characters (dots). At the bottom, there is a single button labeled 'OK'.

Abbildung 10: Account erstellen

Durch betätigen der Create-Auswahl im Login-Fenster öffnet sich ein weiteres PopUp, in dem ein Benutzer angelegt werden kann. Hierfür ist ein eindeutiger Benutzername, ein Passwort und eine Bestätigung des Passwortes notwendig. Entsprechen alle eingegebenen Daten den Anforderungen, wird nach Bestätigung mit OK ein neuer Benutzer erzeugt. Anschließend kann mit diesen Daten der Login im Login-Fenster erfolgen.

4.5.3 Hauptfenster

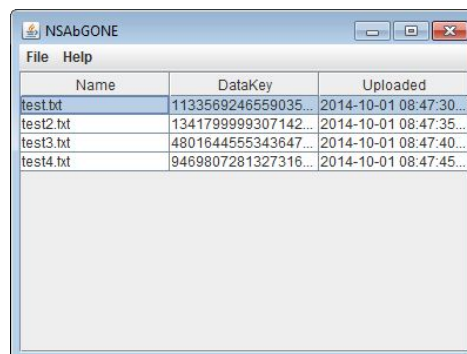


Abbildung 11: Hauptfenster

Das Hauptfenster zeigt in einer Tabelle die aktuell verfügbaren Daten mit dem Originalnamen, dem zugehörigen verschlüsselten Dateinamen und dem Zeitpunkt des Hochladevorgangs. In der oberen Menüleiste kann das zusätzliche DropDown-Menü File und Help ausgewählt werden. Durch Auswählen einer Zeile mit der linken und Betätigen der rechten Maustaste wird ein Kontextmenü mit Möglichkeiten für diese Zeile/Datei angezeigt.

4.5.4 File Menü

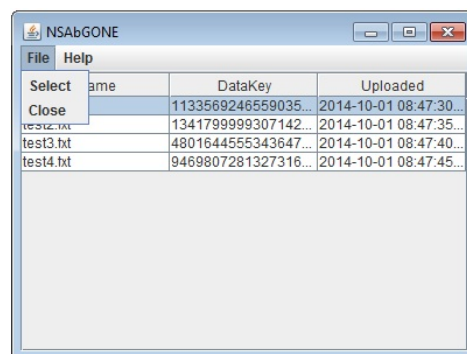


Abbildung 12: File Menü

Der Reiter File in der oberen Menüleiste beinhaltet die Auswahlmöglichkeiten Select und Close. Durch Select kann eine lokale Datei zum Verschlüsseln und Upload in die Cloud ausgewählt werden. Close schließt die Applikation. Die Anzeige der Möglichkeiten kann durch einen Klick auf eine freie Fläche innerhalb der Applikation oder einen beliebigen Klick in die Oberfläche des Betriebssystems geschlossen werden.

4.5.5 Dateiauswahl

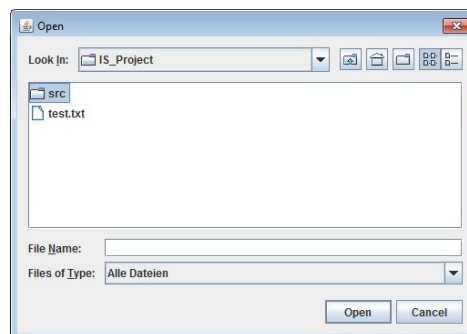


Abbildung 13: Dateiauswahl

In der Dateiauswahl Select wird die lokale Ordnerstruktur des Betriebssystems angezeigt. In dieser kann sich der Benutzer frei bewegen und in das Quellverzeichnis von Dateien wechseln. Das Auswählen einer Datei und die Bestätigung mit Open initialisiert den für den Benutzer nicht ersichtlichen Verschlüsselungsvorgang und Dateiupload. Das Auswahlfenster wird beendet. Wurde die Datei korrekt verschlüsselt und in die Cloud geladen, wird diese im Hauptfenster mit ihren Merkmalen aufgelistet. Bei großen Dateien kann dieser Vorgang eine längere Zeit in Anspruch nehmen. Schließt der Benutzer die gesamte Applikation vor Auflistung in der Dateiübersicht, so wird diese weiterhin durch Hintergrundprozesse verschlüsselt bzw. hochgeladen. Durch Cancel kann das Auswahlfenster für Dateien geschlossen werden.

4.5.6 Help Menü

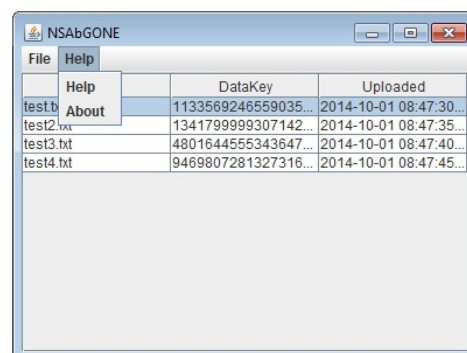


Abbildung 14: Help Menü

Der Reiter Help in der oberen Menüleiste beinhaltet die Auswahlmöglichkeiten Help und About. Die Auswahl Help öffnet ein zusätzliches Fenster mit Hilfestellungen zur

Verwendung, About ein weiteres mit Informationen zur Applikation. Die Anzeige der Möglichkeiten kann durch einen Klick auf eine freie Fläche innerhalb der Applikation oder einen beliebigen Klick in die Oberfläche des Betriebssystems geschlossen werden.

4.5.7 Help Fenster

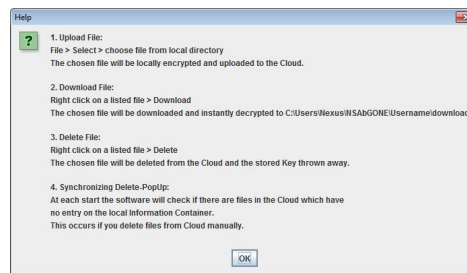


Abbildung 15: Help Fenster

Das Help Fenster beschreibt alle Funktionalitäten der Applikation sowie deren Aufrufe. Mit der Bestätigung von OK wird dieses geschlossen.

4.5.8 About Fenster

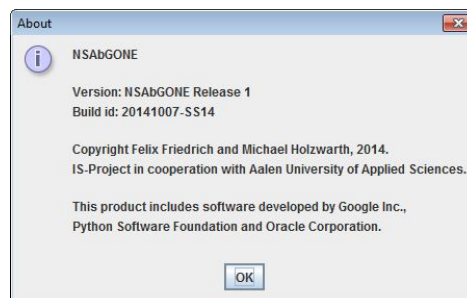


Abbildung 16: About Fenster

Das About Fenster gibt Informationen über die Version, Entwickler, den Hintergrund der Entwicklung und Einsatz von Drittanbieter-Software wieder. Durch OK wird geschlossen.

4.5.9 Kontextmenü

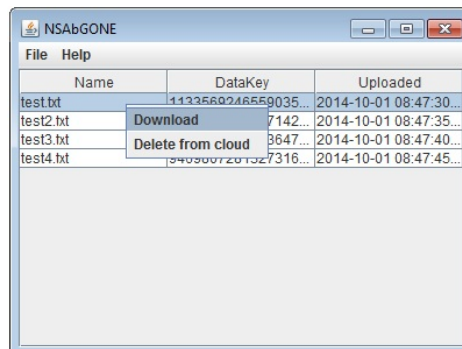


Abbildung 17: Kontextmenü

Das Kontextmenü wird durch Auswahl einer Zeile und Betätigen der rechten Maustaste ausgeführt. Es beinhaltet die Möglichkeiten Download und Delete from Cloud. Download initialisiert das temporäre Herunterladen der ausgewählten Datei mit anschließender Entschlüsselung und entschlüsselter Ablage in HOME/NSAbGONE/<Benutzername>/download.

Mit Delete from Cloud kann eine Datei aus der Cloud zusammen mit dem zugehörigem lokal gespeicherten Schlüssel gelöscht werden. Der Eintrag wird anschließend aus der Dateiübersicht entfernt. Das Kontextmenü kann durch einen Klick auf eine freie Fläche innerhalb der Applikation oder einen beliebigen Klick in die Oberfläche des Betriebssystems geschlossen werden.

4.5.10 Delete Fenster

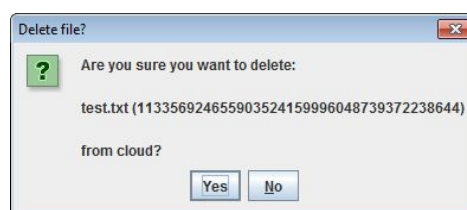


Abbildung 18: Delete Fenster

Ein durch Delete from Cloud im Kontextmenü initialisierter Löschvorgang muss in einer zusätzlichen Abfrage bestätigt werden. Angezeigt wird der originale und der kryptische Dateiname. Durch bestätigen mit YES wird die ausgewählte Datei gelöscht, bei Auswahl von No der Löschvorgang abgebrochen.

4.5.11 Synchronisierungsstatus

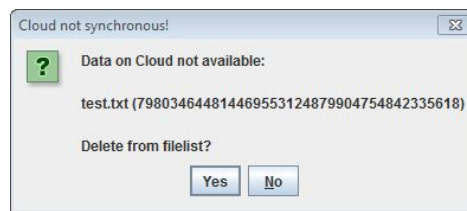


Abbildung 19: Synchronisierungsstatus

Das Fenster über den Synchronisierungsstatus kann der Benutzer selbst nicht öffnen. Es erscheint beim Start der Applikation nur dann, wenn ein lokaler Eintrag einer Datei (mit zugehörigem für den Benutzer nicht ersichtlichen Schlüssel) existiert, diese sich aber nicht mehr in der Cloud befindet. Beispiel für das Hervorrufen eines solchen Ereignisses ist das manuelle Löschen von Dateien in der Cloud-Umgebung. Durch Yes wird der Eintrag in der Dateiübersicht und Datenverwaltung der Applikation gelöscht. Ein Betätigen von No beendet das Fenster bis zum nächsten Start der Applikation ohne Ausführen von Aktionen.

5 Diskussion

Im folgenden Kapitel werden die Ergebnisse der Projektarbeit in der aktuellen Lage (NSA, Prism etc.) eingeordnet. Außerdem wird auf mögliche Erweiterungen der Applikation, sowie weitere mögliche Konzepte der verschlüsselten Datensicherung in der Cloud, eingegangen.

5.1 Bedeutung

Dass Cloud Computing in der heutigen Zeit eine immer größere Bedeutung zukommt und es durchaus seine Existenzberechtigung im Sinne von ökonomischen und ökologischen Gesichtspunkten hat, soll hier nicht weiter thematisiert werden, da es in **2.1.2 Daseinsberechtigung** ausführlich dargelegt wurde. Das essentielle Problem beim Cloud Computing, dass die meisten Anbieter im Ausland liegen und damit nicht unter deutsches Recht fallen, macht es jedoch für viele Anwendungen unbrauchbar. Auf der einen Seite ist der Datenschutz nicht gewährleistet, was gerade bei den empfindlichen deutschen Datenschutzgesetzen gefährlich sein kann. Auf der anderen Seite haben die Enthüllungen der letzten Monate gezeigt, dass sogar auf staatlicher Ebene massiv Industriespionage betrieben wird. Dies ist für die meisten deutschen Firmen, die sich ob des hohen Lohnstandards nur aufgrund von technischem Know-How und modernen Entwicklungen gegen Konkurrenten aus dem Ausland durchsetzen können, ein ausreichender Grund, Cloud Computing zu meiden. Diese Umstände werden durch Applikationen wie der, die im Rahmen dieses Projektes prototypisch entwickelt wurde, behoben. Durch die vorverschlüsselte Ablage muss die genutzte Infrastruktur nicht mehr als sicher angesehen und kann ohne Vorbehalte in dieser Hinsicht genutzt werden. Sehr bewusst wurde bei der Konzeption darauf geachtet, auch BSI Standards für die Verschlüsselung abzudecken: Weniger weil es keine weiteren guten Verschlüsselungssysteme gäbe, als vielmehr um dem deutschen Datenschutz gerecht zu werden und zu zeigen, dass es durchaus möglich ist, im Ausland Daten derart abzusichern, dass sie als sicher im Sinne der deutschen Gesetzgebung gelten.

5.2 Erweiterung der Funktionalitäten

Im folgenden werden Möglichkeiten beschrieben, die bestehende Applikation zu verbessern, ohne dass dazu die grundlegende Architektur verändert werden muss.

5.2.1 Erhöhung der Bedienerfreundlichkeit

Ausbau des Designs/Multirow

Eine sehr simple, aber dennoch äußerst nützliche Erweiterung des bestehenden Tools wäre es dem Benutzer zu ermöglichen, mehrere Zeilen der Tabelle zu markieren und zu bearbeiten. Dies würde die intuitive Nutzbarkeit erhöhen und den Umgang mit großen Dateimengen erleichtern.

Drag&Drop

Um die Benutzerfreundlichkeit des Programmes zu erhöhen wäre es vorteilhaft, Dateien per Drag&Drop in das Programmfenster hinein zu ziehen. Dadurch müsste ein Benutzer nicht jedesmal, wenn er eine Datei hinzufügen möchte, durch die Verzeichnisstruktur seines Betriebssystems navigieren.

Schnittstellen

Zurzeit muss gsutil noch manuell vorkonfiguriert werden, um es anschließend mit der Applikation als Schnittstelle zu Google Cloud Storage zu verwenden. Diese Konfiguration könnte automatisiert durch die Applikation selbst geschehen. Desweiteren haben Testfälle unter Einsatz von gsutil gezeigt, dass dieses nicht performant genug für die Verarbeitung von großen Dateien ist. Bereits bei einer Dateigröße von 50MB kam es zu langen Verzögerungen im Datei-Upload, in manchen Fällen sogar zu Abbrüchen. Dieses Verhalten war zu Beginn der Projektarbeit nicht ersichtlich und ist, auf Grund fehlender Reproduzierbarkeit, nicht zu erklären. Eine Fragmentierung hochzuladender Dateien wäre hier prinzipiell realisierbar oder eine Weiterentwicklung dieser bzw. die Wahl einer alternativen Schnittstelle möglicherweise für eine zukünftige Implementierung sinnvoll. Aktuell befindet sich das IaaS-Angebot Google Cloud Storage indirekt noch im Beta Status, weswegen eine rasche Weiterentwicklung und Optimierung durch den Anbieter und freie Entwickler zu erwarten ist.

Installations Setup

Die gesamte Installation des Programmes kann durch einen Wizard derart automatisiert werden, dass auch Zusatzsoftware wie Python installiert und die Pfade in Konfigurationsdateien hinterlegt werden. Auch wäre es möglich Python für Windowssysteme automatisch bereit zu stellen und in der .jar Datei zu verpacken. Linux und MAC Systeme besitzen ohnehin standardmäßig eine Python2x Installation und verwenden Standardpfade für installierte Anwendungen (/bin, /usr/bin etc.).

Selbstsynchronisierung

Um weitere Verwendungsfelder für die Applikation zu erschließen, ist es sinnvoll die in der Cloud hinterlegten Dateien selbstsynchron mit ihren lokalen Pendanten zu halten: Jedes Mal, wenn indizierte Dateien geändert werden, berechnet die Applikation ein Changeset der alten und der neuen Version und legt es auch verschlüsselt in der Cloud ab. Durch eine neue Ansicht kann der Benutzer dann durch Versionen gehen und alte nach Bedarf wieder herstellen. Dadurch wäre eine Art Backupsystem geschaffen, dass nicht nur Sicherheit gewährleistet, sondern gleichzeitig Redundanz und Hochverfügbarkeit für die versionierten Daten geschaffen wird.

Virtuelles Laufwerk/Synchrone Dateistrukturen

Weiterhin wäre denkbar, die Applikation derart anzulegen, dass sie den Cloudspeicher als virtuelles Laufwerk im System verankert. Daten die darin abgelegt werden, werden on the fly verschlüsselt und in der Cloud gespeichert. Dies könnte auch mittels eines automatisch synchronisierten Ordners geschehen, wie es etwa bei Dropbox der Fall ist. Dann müsste es aber Automatismen geben, die verhindern, dass Daten verloren gehen, wenn der synchrone Ordner auf mehreren Systemen gleichzeitig genutzt und zwei Personen zur selben Zeit an derselben Datei arbeiten würden. Hier wäre eine Art Versionsverwaltung von Nöten, die ähnlich wie GIT Changesets von den Dateien erzeugt und diese verfügbar hält.

Verschlüsselung von Dateistrukturen

Im Rahmen der für die Bearbeitung dieser Projektarbeit zur Verfügung stehenden Zeit, wurde eine Verschlüsselung auf dem Datei-Level gewählt. Dies bedeutet, dass gezielt einzelne Dateien zur Verschlüsselung ausgewählt werden können. Ein Kryptosystem wäre jedoch auch für den Einsatz auf weiteren Leveln möglich. Hierzu gehört die

Verschlüsselung auf dem Datenträger-Level, d.h. das gesamte Betriebssystem inklusive der beinhaltenden Anwendungen und Daten werden auf einem verschlüsselten Datenträger abgelegt. Das führt jedoch zu starken Einbußen der Performance und Zuverlässigkeit. Ein weiteres Level bildet das Dateisystem. In diesem werden komplette Verzeichnisse als Container ver- und entschlüsselt. Dieser Anwendungsfall bietet zudem die Möglichkeiten, Daten nach ihrer Sensibilität einzustufen und jeweils in ihrer Gesamtheit mit einem eigenen Schlüssel zu versehen. Das letzte Level stellt das Anwendungs-Level dar. Darin verwaltet eine Anwendung die Ver- und Entschlüsselung von Daten. [Win11, S. 141f]

Schutz der .boto

Wie in **3.3 Schnittstelle Google Cloud Storage: gsutil und boto** dargelegt ist, stehen die Verbindungsdaten, welche die Gsutil Schnittstelle nutzt, in der Datei .boto im Home Verzeichnis des Benutzers. Diese Möglichkeit erlaubt, ohne Passwordeingabe auf den Cloudspeicher von Google zuzugreifen. Daher sollte diese auch bei Beendigung des Programmes verschlüsselt werden. Dies wird zusätzlich interessant, wenn eine Benutzerverwaltung mit eigenen Benutzeraccounts und zugehörigen Buckets existiert. Jeder in der Applikation registrierte Benutzer hat dann seine eigene .boto, die nach Beendigung der Applikation in seinem Applikationsspeicher (/NSAbGO-NE/<Benutzername>/data/) verschlüsselt abgelegt wird und nur mittels eines korrekten Benutzerpasswortes zugänglich ist - ähnlich wie bei der Behandlung der state.cfg Dateien.

Parallelisierte Kryptographie

Zurzeit werden die Daten verschlüsselt indem für jeden Verschlüsselungsvorgang ein eigener Thread gestartet wird. Dies dient, wie schon erwähnt, dem flüssigeren Programmablauf, ist aber immernoch relativ langsam. In Zukunft sollte dies durch eine aufteilung in mehrere Threads geschehen um die Berechnung zu parallelisieren und den/die Prozessoren besser auslasten zu können: Die zu verschlüsselnde Datei wird in so viele Abschnitte zerteilt, wie das Arbeitssystem Prozessorkerne besitzt. Nun werden entsprechend viele Threads mit den Teilstücken gestartet und diese parallel verschlüsselt. Anschließend werden die Teile wieder zusammengesetzt und in die Cloud geladen.

5.2.2 Serverseitiges Management

Die folgenden Abschnitte sind kaum mit Quellen versehen, da sie aus eigenen Gedanken entstanden sind. Sollte es ähnliche Verfahrensweisen in anderen Systemen geben, so ist dies Zufall und bedeutet nicht, dass Quellen vorsätzlich unterschlagen wurden.

Schlüsselmanagement durch Key-Server

Ein potentieller Einsatz der Applikation auf mehreren Endgeräten macht zwangsläufig ein serverseitiges Schlüsselmanagement unabdingbar. Hinzu kommt eine lange Rückhaltezeit der Schlüssel für verschlüsselt abgelegte Dateien. Dies verringert weniger die Anzahl von abzusichernden Geheimnissen, sondern lediglich deren Größe (Schlüsselgröße meist kleiner als Dateigröße).

[Win11]

Loginserver

Um die Applikation unabhängiger von einem bestimmten Cloudanbieter zu halten, wäre ein Konzept mittels Loginserver sinnvoll: Ein Benutzer hat einen Account bei diesem und meldet sich bei Programmstart an. Möchte er eine Datei hochladen, fragt er beim Loginserver an, wohin dies geschehen soll und dieser antwortet mit einem Cloudverzeichnis. Ebenso geschieht es beim Herunterladen der Daten. Möchte der Betreiber des Systems nun einen anderen Cloud-Anbieter nutzen, etwa weil dieser ein besseres Angebot unterbreitet hat, zieht er alle Daten um und hinterlegt dies beim Loginserver. Der Benutzer merkt von dieser Änderung nichts, aber das Programm arbeitet nun auf einem anderen Speicher als zuvor. Außerdem könnte so auch eine Rechteverwaltung gestaltet werden, sodass der Loginserver den Cloudspeicher verwaltet und den Benutzern selbstständig nur dort Rechte auf dem Speicher gibt, wo sie welche haben sollen und jedesmal neue, wenn sie neue Daten ablegen wollen.

Providing Server

Die Tatsache, dass Benutzer zurzeit auf dem selben Cloudspeicher arbeiten und eventuell sogar die selben Buckets nutzen, verursacht zwei grundsätzliche Probleme: Zum Einen kann ein Benutzer theoretisch (wenn auch nicht mit der Applikation selbst, aber mittels des Webfrontends) Daten anderer Benutzer manipulieren oder löschen. Dieses Problem ließe sich, durch die von den Cloud-Anbietern selbst angebotene Rechteverwaltung innerhalb des Speichers, lösen. Zum Anderen ist es dem Cloud-Anbieter

noch möglich einzusehen, welchem Benutzer welche Dateien zuzuordnen sind, was wiederum eine Gefahr im Sinne der IT-Sicherheit darstellt: Der Anbieter (oder jede Person oder Organisation die sich auf irgendeine Weise Zugriff verschafft hat) kann Analysen über das Uploadverhalten anfertigen und dadurch Schlüsse auf den Inhalt der Daten ziehen. An dieser Stelle könnte man über den Einsatz eines Providing Servers nachdenken: Dieser Server steht zwischen dem Endanwender und dem Cloudspeicher selbst. Alle Daten werden weiterhin lokal verschlüsselt, aber nicht direkt in der Cloud abgelegt, sondern an den Providing Server gesendet, welcher sie anschließend auf den oder die Cloudspeicher legt und eine Abbildung der Zugehörigkeiten speichert. Dadurch bleiben die Daten sicher verschlüsselt und für einen Dritten nicht ersichtlich. Allerdings sind die Daten aus Sicht des Cloudanbieters derselben Person zuzuordnen und jeder der sich unrechtmäßig oder auch rechtmäßig Zugriff zu der Cloud verschaffen kann, ist nicht in der Lage sie zuzuordnen. Sollte sich eine Person oder Organisation Zugriff auf den Providing Server verschaffen, kann sie zwar Einsicht in die Zuordnung nehmen, aber sie dennoch nicht entschlüsseln. Ein potentieller Angreifer muss also zwei Server überwinden um die oben genannten Analysen anzufertigen. Im Sinne der aktuellen Enthüllungen über Geheimdienste und Polizeioorganisationen in den USA, aber auch dem Rest der Welt, wäre hier durchaus sinnvoll zwar einen der großen (und daher billigen) Cloudanbieter der USA zu wählen, die Providing Server aber national zu halten. Solange man der eigenen Nation mehr traut als den USA, wurde hier Sicherheit gewonnen. Alternativ könnte für den Providing Server auch ein Land gewählt werden, welches politisch diametral zu den westlichen Industrienationen steht. Dass es einem Geheimdienst möglich ist, durch rechtmäßige Verfahren Zugriff auf beide Datenpools zu erhalten wird so sehr unwahrscheinlich und erhöht den Aufwand weiter.

Serverseitige Rechteverwaltung/Teilen von Dateien

Sollte es einen oben genannten Providing Server geben, stellt die Rechteverwaltung kein Problem dar: Der Server verwaltet die Accounts der Benutzer und sorgt dafür, dass er zwischengeschaltet ist, dass Benutzer nur an Dateien kommen, die ihnen auch gehören bzw. für die sie Rechte von anderen Benutzern erhalten haben. Diese Berechtigungen können vergeben werden indem die Benutzer einander die Schlüssel zukommen lassen und dem Server mitteilen, dass ein weiterer Benutzer Zugriff bekommen soll. Der Schlüsselaustausch darf hier keinesfalls über den login/providing

Server geschehen, da dies dafür sorgen würde, dass Dritte uneingeschränkt Dateien lesen können (nämlich der Betreiber des Loginservers). Alternativ könnten die Schlüssel über den Server verteilt werden, dann allerdings wieder mit einer asymmetrischen Chiffre verschlüsselt. Wichtig ist hierbei, dass der Austausch der asymmetrischen Schlüssel sicher erfolgt um Man-In-The-Middle Angriffe zu verhindern. Eine Orientierung an den Schlüsselserversn des PGP Projektes ist hier sinnvoll.

5.2.3 Erweiterte Sicherheit

Im folgenden Abschnitt werden Methoden beschrieben, die die Sicherheit des Systems weiter erhöhen oder durch alternative Sicherheitsmechanismen die Anwendungsbreite der Applikation erweitern.

OpenJDK

Im Laufe der Projektarbeit kamen seitens des betreuenden Professors Zweifel auf, ob die Wahl von Java als Plattform und Programmiersprache sinnvoll war, da Java als nicht besonders sicher gilt. Grundsätzlich sollte die Sicherheit von Java in diesem Projekt keine Rolle spielen, da der Rechner auf dem die Applikation läuft, als sicher gilt und die erzeugten Chiffre überprüfbar korrekt nach den NIST Standards sind. Trotzdem sorgt die Verwendung von Java in diesem Bereich einerseits für einen bitteren Beigeschmack. Andererseits gibt es eine OpenSource Implementierung die als sicherer gilt: OpenJDK. Das Projekt wurde sowohl unter Windows und dem Oracle Java als auch unter Linux und OpenJDK entwickelt und getestet. Somit stellt OpenJDK eine mögliche Alternative dar und kann bei der Kompilation wie auch beim Betrieb der Applikation ohne Einschränkung verwendet werden.

Alternative Programmiersprachen

Im Rahmen der Überlegungen zu Java und OpenJDK kamen auch Vorschläge zu alternativen Programmiersprachen auf. Hier bietet sich vor allem C++ an, da es für alle gängigen Plattformen Compiler gibt, es strikt objektorientiert ist und sehr exakten Zugriff auf den Speicher zulässt, was zur Absicherung der Schlüssel auf dem Hostsystem wertvoll ist. Hier erhöht sich zwar der Entwicklungsaufwand (siehe **2.2 Java**), dafür kann die Applikation auf dem Hostsystem besser abgesichert werden. So wird ein Betrieb auf Fremdsystemen, zum Beispiel von einem USB Stick aus, noch interessanter.

Kaskadierte Verschlüsselung

Zwischenzeitlich gab es Überlegungen dahingehend, aus der Applikation eine Art Teamplattform mit Versionsverwaltung zu schaffen. Alleine die Berechnung und Verwaltung der Changesets hätte den Rahmen dieser Projektarbeit gesprengt, allerdings kam in diesem Zusammenhang die Idee auf, Verschlüsselungskaskaden aufzubauen um Rechte derart abzubilden, wie sie in Organisationsstrukturen vorkommen: Eine Person steht auf einer bestimmten Stufe und ist berechtigt alles zu sehen/betreten/wissen, was von seiner Sicherheitseinstufung gleichrangig oder niedriger steht. In diesem Falle ist eine asymmetrische Chiffre unabdingbar. Die Daten werden zunächst wie üblich verschlüsselt, die Schlüssel allerdings direkt mit abgelegt. Bevor dies geschieht werden die Daten asymmetrisch mit einer Art Metaschlüssel (dem öffentlichen an dieser Stelle) chiffriert. Dieser ist Teil des Ebenenschlüssels zu dem die Datei gehört. Die Ebenenschlüssel sind keine reinen Schlüssel im eigentlichen Sinne, sondern ein Schlüsselpaket, das alle asymmetrischen privaten Schlüssel umfasst, die den darunter liegenden Ebenen zugeordnet sind. Die öffentlichen Schlüssel sind allesamt frei zugänglich. Jetzt ist es möglich einer Person Rechte auf einer bestimmten Kaskade zuzuordnen und ihr damit auch Zugriff auf die darunter liegenden zu geben. Eine Person die in einer tieferen Ebene steht kann aber Daten für höhere Ebenen ablegen, wodurch sie selbst eigentlich nicht mehr an die eigene Datei heran kommt, es sei denn sie behält den symmetrischen Schlüssel selbst.

Gedacht war die Umsetzung und Nutzung derart: Ein Unternehmen, nehmen wir hier als Beispiel ein Architekturbüro, arbeitet direkt auf einem Cloudspeicher anstatt auf firmeninternen Ablagen, um die Vorteile von dessen zu nutzen und den Mitarbeitern unterwegs Zugriff auf die Daten zu gewähren. Jeder Architekt erhält nun den Ebenenschlüssel der Ebene, der er zugeordnet ist und kann durch sein Interface auf alle Daten direkt zugreifen; ob von einem Laptop, dem Büro oder dem Smartphone spielt hier keine Rolle. Steigt er eine Ebene auf oder ab, erhält er entsprechend den neuen Ebenenschlüssel. Damit ein findiger Architekt nicht hingehen kann und sein Ebenenschlüssel speichert, bevor er herabgestuft wird, werden die symmetrischen Schlüssel jede Nacht neu asymmetrisch verschlüsselt. Die Firma betreibt einen eigenen kleinen Server, der diese neuen Ebenenschlüssel erzeugt und anhand der Berechtigungen verteilt. Ausserdem wird einem entlassenen Mitarbeiter auch der gesamte Zugriff auf den Cloudspeicher verwehrt, sodass er mit seinen (noch gültigen) Schlüsseln nichts

mehr anfangen kann.

Key recovery/Masterkey

Bei allen Vorteilen die ein System mit lokaler Verschlüsselung bietet, entsteht auch immer ein Nachteil: Sollte der Benutzer (oder Kunde) seine(n) Schlüssel verlieren, sind alle gespeicherten Daten verloren sofern das System richtig arbeitet. Eine Lösung dieses Problems, die die Sicherheit zwar schmälert, aber dritten keinen Einblick in die Daten gewährt soll in diesem Abschnitt vorgestellt werden.

Voraussetzung für diese Möglichkeit ist ein System, bei dem der Benutzer seine Daten symmetrisch verschlüsselt und auch die Abbildungstabelle (Schlüssel und Zuordnung zu Dateien) asymmetrisch chiffriert in der Cloud ablegt, wie es zum Beispiel in **5.2.2 Serverseitiges Management** beschrieben wird. In diesem Fall hat der Benutzer mit dem privaten asymmetrischen Schlüssel eine Art Masterkey, mit dem auf die Gesamtheit seiner Daten zugegriffen werden kann. Die Idee ist nun, diesen Masterkey zu einem Shared-Secret zu machen. Der Benutzer generiert also selbstständig (dies ist wichtig für die Vertraulichkeit!) aus seinem Masterkey zwei Geheimnisse, die nur gemeinsam eine Wiederherstellung des Schlüssels erlauben. Den einen Teil behält er selbst. Dieser kann bei einer vertrauenswürdigen Person hinterlegt werden, im Tresor einer Bank etc.. Den zweiten Teil gibt er an eine Instanz ab, bei der er mit seinen Personalien registriert ist (erhöhte Sicherheit: biometrische Verfahren). Im Falle eines Geschäftsmodells ist dies die Firma, die den Service anbietet. Verliert der Benutzer seinen Schlüssel, ist es möglich mittels der beiden Geheimnisse seinen Masterkey erneut zu erzeugen. Gleichzeitig kann aber die Instanz mit dem zweiten Teilschlüssel alleine nichts anfangen und auch der Besitzer des ersten Teils kann sich selbstständig keinen Zugriff auf die Daten verschaffen. [Wik14b]

Fraktionierung

Einer der effektivsten Ansatzpunkte für einen Angriff auf die Verschlüsselung ist zurzeit eine Analyse der Dateigrößen und damit verbunden Rückschlüsse auf deren Inhalt. Dies ist nicht nur gefährlich, weil solche Informationen manchem Angreifer schon ausreichen, sondern auch, weil darüber Vermutungen über den Dateityp angestellt werden können. Dies wiederum bedeutet, dass Teile des Klartextes bekannt sind, da viele Dateitypen teils sehr große Header haben, die sich im Inhalt kaum un-

terscheiden. Ein Angriff auf die Verschlüsselung selbst hingegen hat sehr viel größere Chancen, wenn Teile des Klartextes bekannt sind. Diese Methode nennt sich Known-Plain-Text Angriff. Um diese Schwachstelle zu entfernen ist es nützlich, die Dateien vor der Verschlüsselung zu fraktionieren, also in Teile einer definierten Größe zu zerteilen. Ist eine Datei kleiner als die Blockgröße wird sie aufgefüllt mit später entfernbaren Bytefolgen. Nun liegen im Cloudspeicher nicht zusammenhängende Bitblöcke, die keinerlei Auskunft über Dateityp oder Inhalt zulassen. Im Falle eines Providing Servers weiß der Cloudanbieter nicht einmal wem welcher Block gehört und der providing Server nicht, welche Blöcke des Kunden zusammen gehören und eine Datei bilden oder deren Reihenfolge. Die Applikation könnte die Blöcke nämlich in zufälliger Reihenfolge verschlüsseln und hochladen.

Verschlüsselte Container

Manche Sicherheitssysteme (zum Beispiel TrueCrypt) arbeiten mit verschlüsselten Containern. Dies bedeutet, dass zunächst festgelegt wird, wie groß die verschlüsselte Datenmenge maximal sein darf. In dieser Größe wird nun eine Datei angelegt, gefüllt mit zufälligen Bits. Wenn der Benutzer eine Datei in diesen Container hinein speichern möchte, verschlüsselt er sie und überschreibt einen Teil der Zufallsbits mit dem Chiffretext. Ein Angreifer der den Container habhaft wird ist nicht in der Lage zu sagen, ob, wie viel, und wo Dateien in dem Container liegen, was die Entschlüsselung beinahe unmöglich macht. Auf das Projekt abgebildet würde dies allerdings einen Vorteil des Cloudspeichers zerstören, nämlich nur so viel Speicher bezahlen zu müssen wie tatsächlich gebraucht wird. In einem gewissen Maße bildet aber der Cloudspeicher selbst bei Fraktionierung der Daten diesen Vorteil: Der Angreifer sieht Datenfragmente und kann keine Aussage darüber treffen, wo Dateien beginnen oder enden, wieviele es gesamt oder wie groß sie sind.

Rechteverwaltung

Zurzeit gibt es in der Applikation keinerlei Rechteverwaltung auf dem Cloudspeicher selbst. Ein Benutzer mit Zugriffsrechten auf den Cloudspeicher, also ein Benutzer der Applikation, kann theoretisch per Webinterface die Daten anderer manipulieren. Dies muss auf Dauer geändert werden, entweder über die interne Rechteverwaltung der Cloudanbieter oder durch Login und Providing, wie in **5.2.2 Serverseitiges Management** beschrieben wurden. Es kann in der Applikation selbst geschehen oder

manuell im Cloudspeicher, wenn neue Benutzer hinzu kommen. Im Falle von Google Cloud Storage wäre es möglich, bei Neuanlegen eines Benutzer einen für ihn vorgesehenen Bucket zu erstellen, auf den nur der Benutzer allein Schreibrechte hat. Leserechte sind aufgrund der Verschlüsselung eher unkritisch. Außerdem ist der Hintergrund der Applikation ja das Misstrauen gegenüber dem Cloudanbieter selbst, der ohnehin Schreib- und Leserechte auf alle Dateien besitzt.

6 Literatur

- [All11] ALLIANCE, Cloud S.: *SECURITY GUIDANCE FOR CRITICAL AREAS OF FOCUS IN CLOUD COMPUTING V3.0*.
<https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf>, 2011. – [Online; letzter Zugriff: 01.10.2014]
- [Cha09] CHACON, Scott: *Pro Git*. Apress, 2009 (Books for professionals by professionals). – ISBN 9781430218333
- [EGT14] ELEKTRIZITÄT, Bundesnetzagentur für ; GAS ;
TELEKOMMUNIKATION, Post und E.: *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung*. http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/QES/Veroeffentlichungen/Algorithmen/2014Algorithmenkatalog.pdf?__jsessionid=44761A7B91153744E5BAC51D89FB1FBD?__blob=publicationFile&v=1, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [EMP13] ERL, Thomas ; MAHMOOD, Zaigham ; PUTTINI, Ricardo: *Cloud computing: concepts, technology & architecture*. 1. print. Upper Saddle River, N.J.; Munich [u.a.] : Prentice Hall, ServiceTech Press, 2013. – XXXIV, 487 S.. – ISBN 978-0-13-338752-0
- [HH11] HALPERT, Ben (Hrsg.) ; HALPERT, Ben (Hrsg.): *Auditing cloud computing: a security and privacy guide*. Hoboken, NJ : Wiley-VCH, c2011. – XVI, 206 S.. – ISBN 978-0-470-87474-5
- [HSG10] HENNEBERGER, Matthias ; STREBEL, Jörg ; GARZOTTO, Fabio: Ein Entscheidungsmodell für den Einsatz von Cloud Computing in Unternehmen. In: *HMD Praxis der Wirtschaftsinformatik* 47 (2010), Nr. 5, S. 76–84. <http://dx.doi.org/10.1007/BF03340515>. – DOI 10.1007/BF03340515. – ISSN 1436-3011
- [Inc14a] INC., Google: *gsutil Tool*.

- <https://cloud.google.com/storage/docs/gsutil>,
2014. – [Online; letzter Zugriff: 01.10.2014]
- [Inc14b] INC., Google: *Python Example*. <https://cloud.google.com/storage/docs/gspythonlibrary>,
2014. – [Online; letzter Zugriff: 01.10.2014]
- [Inc14c] INC., Google: *Using OAuth 2.0 to Access Google APIs*. <https://developers.google.com/accounts/docs/OAuth2>,
2014. – [Online; letzter Zugriff: 01.10.2014]
- [Inc14d] INC., Google: *What is Google Cloud Storage - Building Blocks*.
<https://cloud.google.com/storage/docs/overview#blocks>,
2014. – [Online; letzter Zugriff: 01.10.2014]
- [Inf14a] INFORMATIONSTECHNIK, Bundesamt für Sicherheit in d.: *Cloud Computing Grundlagen*.
https://www.bsi.bund.de/DE/Themen/CloudComputing/Grundlagen/Grundlagen_node.html,
2014. – [Online; letzter Zugriff: 01.10.2014]
- [Inf14b] INFORMATIONSTECHNIK, Bundesamt für Sicherheit in d.:
Kryptographische Verfahren: Empfehlungen und Schlüssellängen.
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102_pdf.pdf?__blob=publicationFile, 2014.
– [Online; letzter Zugriff: 01.10.2014]
- [LRWXX] LIPMAA, Helger ; ROGAWAY, Phillip ; WAGNER, David: *Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption*.
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf>, 20XX. –
[Online; letzter Zugriff: 01.10.2014]
- [LTG⁺14] LEONG, Lydia ; TOOMBS, Douglas ; GILL, Bob ; PETRI, Gregor ;
HAYNES, Tiny: *Gartner: Magic Quadrant for Cloud Infrastructure as a Service*. <http://www.gartner.com/technology/>

- reprints.do?id=1-1UKQQA6&ct=140528&st=sb, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [mic14] MICROSYSTEMS, Sun: *Java Technology: The Early Years*.
<http://web.archive.org/web/20100105045840/http://java.sun.com/features/1998/05/birthday.html>, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [Mil09] MILLER, Michael: *Cloud computing: Web-based applications that change the way you work and collaborate online ; [includes coverage of Google collaboration tools, Apple's MobileMe, and much more!]*. Indianapolis, IN : Que, 2009. – XIII, 293 S.. – ISBN 978-0-7897-3803-5
- [MSLL14] MOHAPATRA ; SANJAY ; LOKHANDE ; LAXMIKANT: *Cloud Computing and ROI*. Springer Verlag, 2014. – ISBN 9783319086637
- [MWRS11] MEINEL, C. ; WILLEMS, C. ; ROSCHKE, S. ; SCHNJAKIN, M.: *Virtualisierung und Cloud Computing: Konzepte, Technologiestudie, Marktübersicht*. Univ.-Verlag, 2011 (Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam). – ISBN 9783869561134
- [Ora14] ORACLE: *The History of Java Technology*.
<http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html>, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [Sch06] SCHNEIER, Bruce: *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C; [der Klassiker]*. München [u.a.] : Pearson Studium, 2006. – XXII, 844 S.. – ISBN 3-8273-7228-3
- [SSP08] SWOBODA, Joachim ; SPITZ, Stephan ; PRAMATEFTAKIS, Michael: *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen*. 1. Aufl. Wiesbaden : Vieweg + Teubner, 2008. – XVIII, 263 S.
http://deposit.d-nb.de/cgi-bin/dokserv?id=2844602&prov=M&dok_var=1&dok_ext=htm. – ISBN 3-8348-0248-4

- [ST01a] STANDARDS, National I. ; TECHNOLOGY: *ADVANCED ENCRYPTION STANDARD (AES)*. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, 2001. – [Online; letzter Zugriff: 01.10.2014]
- [ST01b] STANDARDS, National I. ; TECHNOLOGY: *Recommendation for Block Cipher Modes of Operation*. <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>, 2001. – [Online; letzter Zugriff: 01.10.2014]
- [ST11] STANDARDS, National I. ; TECHNOLOGY: *The NIST Definition Of Cloud Computing*. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>, 2011. – [Online; letzter Zugriff: 01.10.2014]
- [ST12] STANDARDS, National I. ; TECHNOLOGY: *Secure Hash Standard (SHS)*. <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>, 2012. – [Online; letzter Zugriff: 01.10.2014]
- [Ull12] ULLENBOOM, Christian: *Java 7 - mehr als eine Insel: das Handbuch zu den Java-SE-Bibliotheken; [Nebenläufigkeit, Datenstrukturen und Algorithmen, Ein/Ausgabe, XML, Swing, Grafik- und Netzwerkprogrammierung, RMI, JavaServer Pages und Servlets, JDBC, Reflection u.v.m.]*. 1. Aufl. Bonn : Galileo Press, 2012. – 1433 S.. – ISBN 978-3-8362-1507-7
- [Wik14a] WIKIPEDIA: *Advanced Encryption Standard*. http://de.wikipedia.org/wiki/Advanced_Encryption_Standard, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [Wik14b] WIKIPEDIA: *Shamir's Secret Sharing*. http://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing, 2014. – [Online; letzter Zugriff: 01.10.2014]
- [Win11] WINKLER, Vic: *Securing the cloud: cloud computer security techniques and tactics*. Amsterdam; Heidelberg [u.a.] : Elsevier Syngress, 2011. – XXIV, 290 S.. – ISBN 978-1-59749-592-9

- [WPG⁺10] WU, Jiyi ; PING, Lingdi ; GE, Xiaoping ; WANG, Ya ; FU, Jianqing:
Cloud Storage as the Infrastructure of Cloud Computing. In: *Intelligent
Computing and Cognitive Informatics (ICICCI), 2010 International
Conference on*, 2010

Anhang

A Installationsanleitung

In **3.3 Schnittstelle Google Cloud Storage: gsutil und boto** wird kurz beschrieben, welche Vorbedingungen für den Einsatz der Applikation erfüllt sein müssen.

- Python:
Python muss in der Version 2.6.x oder 2.7.x unter Windows oder Linux im Standard Installationsverzeichnis installiert sein.
- gsutil und boto:
Die Installation und Konfiguration von gsutil und boto entspricht der Beschreibung der Google Cloud Plattform und ist in https://cloud.google.com/storage/docs/gsutil_install detailliert einzusehen und nachzuvollziehen.
- Java:
Da die Applikation in Java erstellt wurde, wird dies zum Ausführen auf dem System benötigt. Aufgerufen werden kann NSAbGONE.jar in der Konsole durch:
`java -jar NSAbGONE.jar`.

B Digitaler Anhang

- Quellcode
- Code-Dokumentation (JavaDoc)
- UML-Diagramme
- Dokumentation (IS-Projekt-Friedrich-Holzwarth.pdf)
- Grafiken der Dokumentation
- Digitale Literatur
- Ausführbare jar-Datei (NSAbGONE.jar)