Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ernst Mattern

Clientseitige Sicherheitsarchitektur für Datenverteilung unter Einsatz eines entfernten Speichers

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ernst Mattern

Clientseitige Sicherheitsarchitektur für Datenverteilung unter Einsatz eines entfernten Speichers

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. habil. Dirk Westhoff
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 28. August 2012

Ernst Mattern

Thema der Arbeit

Clientseitige Sicherheitsarchitektur für Datenverteilung unter Einsatz eines entfernten Speichers

Stichworte

Client-Server-Architektur, Sicherheitsarchitektur, Datenverschlüsselung, Dateifreigabe durch Verschlüsselung, verschlüsselte Kommunikation, Speicher im Cloud, Dropbox, Public-Key-Infrastruktur

Kurzzusammenfassung

Diese Arbeit befasst sich mit Analyse und Umsetzung von Sicherheitsmerkmalen für eine rechnergestützte Architektur, die für eine Datenaufbewahrung auf einem entfernten Speicher im Netz, für die Dateifreigabe und für eine netzwerkbasierte Kommunikation eingesetzt werden kann. Es werden Anbieter des Speicherdienstes im Netz und ihre Schwächen in Bezug auf die Sicherheit betrachtet. Dies bedingt die Entwicklung einer clientseitigen Sicherheitsarchitektur, bei der der Benutzer in der Lage ist, selbst für seine Sicherheit zu sorgen. Unter Einsatz des Speicherdienstes von Dropbox Inc. ©wird eine prototypische Implementierung eines Softwaremoduls in Java vorgenommen, das aus zwei Komponenten besteht und mit Mitteln der Kryptographie eine sichere Datenaufbewahrung, Dateifreigabe und Kommunikation ermöglicht.

Ernst Mattern

Title of the paper

Client-Sided Architecture for a Secure Data Exchange using a Network Storage

Keywords

Client-Server-Architecture, Security-Based Architecture, Data Encryption, Filesharing by Cryptographic Means, Encrypted Network Communication, Cloud Storage, Dropbox, Public-Key-Infrastructure

Abstract

This paper deals with analysis and developement of security aspects for a computer-based

architecture, which is used for keeping data on a network storage, for the filesharing und the network communication. There are providers for a network storage service to get viewed, so their security problems can be pointed out. This forces a developement of a client-sided architekture, which allows the clients to take care of security for the private data on their own. Using a storage service of Dropbox Inc. ©an application will be implemented in Java. It consists of two components and solves the security aspects for the data storage, the filesharing und the network communication with the cryptographic methods.

Inhaltsverzeichnis

Inhaltsverzeichnis vii

Abbildungsverzeichnis viii

1. Einleitung 1
1.1. Motivation . . . 1
1.2. Gliederung der Arbeit . . . 2

2. Grundlagen 3
2.1. Client-Server-Architektur . . . 3
2.2. Sicherheitsmerkmale . . . 3
2.2.1. Zugriffskontrolle . . . 3
2.2.2. Authentizität . . . 3
2.2.3. Vertraulichkeit . . . 4
2.2.4. Integrität . . . 4
2.3. Sicherheitsanforderungen . . . 4
2.3.1. Sicheres Anmelden/Einloggen . . . 5
2.3.2. Sicherer Übertragungskanal . . . 5
2.3.3. Sichere Datenaufbewahrung . . . 5
2.3.4. Sichere Dateifreigabe . . . 5
2.4. Kryptographie . . . 6
2.4.1. Kryptographische Primitive . . . 6
2.4.1.1. Symmetrische Verschlüsselung . . . 6
2.4.1.2. Asymmetrische Verschlüsselung . . . 8
2.4.1.3. Hybride Verschlüsselung . . . 9
2.4.1.4. Hashfunktion . . . 10
2.4.2. Anwendungsfälle . . . 12
2.4.2.1. Digitale Signatur . . . 12
2.4.2.2. Digitales Zertifikat . . . 14
2.4.2.3. SSL/TLS . . . 15
2.5. Zu gesetzlichen Regelungen . . . 15
2.6. Der entfernte Speicher . . . 15
2.6.1. Anwendungsfälle eines entfernten Speichers . . . 15
2.6.2. Kommunikationsschnittstellen zu einem entfernten Speicher . . . 17
2.7. Entwicklungsumgebung . . . 17
2.7.1. Android-Plattform . . . 18

2.7.1.1. Activity . . . 19
2.7.1.2. Service . . . 19
2.7.1.3. Content Provider . . . 19
2.7.1.4. Broadcast-Receiver . . . 19
2.7.1.5. Intent-Objekt . . . 19
2.7.1.6. AndroidManifest.xml . . . 20
2.7.2. Java Cryptography Architecture und Java Cryptography Extension . . 20

3. Analyse **21**
3.1. Sicherheitsanforderungen beim Einsatz eines entfernten Speichers . . . 21
3.1.1. Sicheres Anmelden/Einloggen . . . 21
3.1.2. Sicherer Übertragungskanal . . . 22
3.1.3. Sichere Datenaufbewahrung . . . 23
3.1.4. Sichere Dateifreigabe . . . 24

4. Umsetzung **27**
4.1. Von Sicherheitsanforderungen zur Architektur . . . 30
4.1.1. Sicheres Anmelden/Einloggen . . . 30
4.1.2. Sicherer Übertragungskanal . . . 32
4.1.3. Sichere Datenaufbewahrung . . . 33
4.1.4. Sichere Dateifreigabe . . . 35

5. Realisierung **39**
5.1. Anwendungsfälle . . . 39
5.2. Komponenten . . . 40
5.2.1. Benutzer-Manager . . . 41
5.2.1.1. SSL-Listener . . . 41
5.2.1.2. UserManager . . . 43
5.2.1.3. User . . . 44
5.2.1.4. Main . . . 44
5.2.2. Benutzer-Client . . . 44
5.2.2.1. DataBaseClient . . . 45
5.2.2.2. DropBoxClient . . . 46
5.2.2.3. LocalFileListActivity . . . 47
5.2.2.4. RemoteFileListActivity . . . 47
5.2.2.5. UserListActivity . . . 47
5.2.2.6. EncodedInput . . . 47
5.2.2.7. Crypter . . . 48
5.2.2.8. Service . . . 49
5.2.2.9. LogInActivity . . . 52
5.2.2.10. AndroidManifest.xml . . . 53

6. Test **54**

7. Zusammenfassung 60

A. BAclientManager-Quellcode 62
A.1. SSL-Listener.java . . . 62
A.2. UserManager.java . . . 64
A.3. User.java . . . 68
A.4. Main.java . . . 68

B. BAclient-Quellcode 69
B.1. DataBaseClient.java . . . 69
B.2. DropBoxClient.java . . . 73
B.3. LocalFileListActivity.java . . . 75
B.4. RemoteFileListActivity.java . . . 77
B.5. UserListActivity.java . . . 78
B.6. EncodedInput.java . . . 80
B.7. Crypter.java . . . 80
B.8. Service.java . . . 83
B.9. LogInActivity.java . . . 88

Literaturverzeichnis 95

Abbildungsverzeichnis

2.1. Symmetrische Verschlüsselung[6] . . . 6
2.2. Blockchiffren[6] . . . 7
2.3. Stromchiffren[6] . . . 8
2.4. Zwei Reihenfolgen der asymmetrischen Verschlüsselung[8] . . . 9
2.5. Hybride Verschlüsselung[7] . . . 10
2.6. Nichtkollisionsresistente Hashfunktion[13] . . . 11
2.7. Digitale Signatur: Dokument signieren[1] . . . 12
2.8. Digitale Signatur: Dokument verifizieren[1] . . . 12
2.9. Digitales Zertifikat nach X.509-Standard [2] . . . 14
2.10. Anwendungsfälle für einen entfernten Speicher . . . 16
2.11. Eclipse IDE mit Android-Emulator . . . 18
2.12. JCA/JCE Architektur [9] . . . 20

4.1. Server-Speicher und Client-Architektur . . . 27
4.2. Benutzer-Manager-Komponenten . . . 28
4.3. Benutzer-Client-Komponenten . . . 29
4.4. Sequenzdiagramm: Anmelden . . . 30
4.5. Sequenzdiagramm: Initialisierung eines sicheren Übertragungskanals . . . 32
4.6. Sequenzdiagramm: Datei auf den entfernten Speicher übertragen . . . 33
4.7. Sequenzdiagramm: Datei vom entfernten auf den lokalen Speicher übertragen 34
4.8. Sequenzdiagramm: Dateifreigabe gewähren . . . 36
4.9. Sequenzdiagramm: Dateifreigabe zurücknehmen . . . 37

5.1. Zu implementierende Komponenten . . . 40
5.2. Klassendiagramm: BAclientManager . . . 41
5.3. Klassendiagramm: BAclient . . . 45

6.1. BAclientManager gestartet . . . 54
6.2. BAclient gestartet . . . 54
6.3. Keine Dateien auf dem entfernten Speicher . . . 55
6.4. Dateien auf dem lokalen Speicher . . . 55
6.5. Notification über Upload . . . 56
6.6. Upload von einem BAclient . . . 56
6.7. Freigabe keinem Benutzer gewährt . . . 57
6.8. Freigabe dem Benutzer *mok* gewährt . . . 58
6.9. Ergebnis der Freigabe für den Benutzer *mok* . . . 58

6.10. freigegebene Datei für den Benutzer *mok* 59

1. Einleitung

Schon seit einer ganzen Weile arbeitet man im Alleingang an einem spannenden rechnergestützten Projekt. In lauter Aufregung werden Kollegen, Freunde, Kommilitonen und alle, die unter anderem im Netz erreichbar sind, darüber informiert. Die Menschen zeigen Interesse und möchten sich gerne in das Projekt einbringen. Der „Urheber“ des Projekts macht sich seitdem Gedanken, wie die Projektdaten ausgetauscht werden können. Da wäre die Möglichkeit die Daten mit Hilfe eines Datenträgers - eines USB-Speichers - erst einmal in einen portablen Zustand zu bringen. Somit hätte auf eine einfache Weise erreicht werden können, dass an den Daten überall gearbeitet werden kann. Die Daten können auch selektiv oder im ganzen an andere weitergegeben werden. Die Vorgehensweise ist in studentischer Praxis keine Seltenheit. Das Problem, mit dem der Datenbesitzer hier schnell konfrontiert wird, ist die praktische Nichttauglichkeit solcher Art der Datenverteilung, da die selektive Weitergabe zu zeitaufwändig und auch nur an die Personen im gleichen physischen Raum möglich ist. Die räumliche Beschränkung wäre mit Hilfe des Internet aufgehoben, was eine Weitergabe an alle Interessierten bzw. an alle Beteiligten ermöglicht. Der Transfer der Daten zwischen zwei Punkten mit Hilfe des Internet ist auf unterschiedliche Weisen und mit unterschiedlichen Protokollen möglich. Was ist aber mit der selektiven Weitergabe? Das ist mehr als nur Datentransport. Ab einer bestimmten Datenmenge bedarf es einer Automatisierung, da sonst der Aufwand für einen Menschen zu groß wäre. Wenn Datenzugriff bzw. Datenverteilung automatisiert werden sollen, sind die Verfügbarkeit des Speichers und, wenn der Speicher nicht dem Datenbesitzer gehört, eine Art eigener Zugriffskontrolle und die Sicherheit der Daten auf dem entfernten Speicher von entscheidender Bedeutung.

1.1. Motivation

Der Begriff von Cloud Computing wird immer bekannter, und auch für Menschen, die nicht unbedingt mit technischen Grundlagen der Rechnersysteme vertraut sind, ist Cloud keine unvorstellbare, abstrakte Bezeichnung mehr. Obwohl Cloud Computing viele Technologien und Dienste vereinigt, nutzt und anbietet, wird sich in dieser Arbeit auf einen Dienst konzentriert, und zwar auf die Zurverfügungstellung des Speichers, der im Titel der Arbeit als entfernter

Speicher bezeichnet wird. Wie schon erwähnt, kann Cloud Computing unzählige Dienste anbieten, die unter Einsatz von Rechnern und Netzwerken möglich sind. Deswegen soll weder davon ausgegangen werden, dass ein entfernter Speicher unbedingt zu Cloud Computing gehört, noch muss Cloud Computing etwas mit einem entfernten Speicher zu tun haben. Der Grund, warum hier auf Cloud eingegangen wird, ist nur praktischer Natur, in der Hinsicht, dass es viele Anbieter gibt, die den Speicher sowohl umsonst, als auch günstig zur Verfügung stellen, und das zu ihren Cloud Diensten zählen. So ein entfernter Speicher soll für Datenverteilung in Betracht kommen. Unter Datenverteilung wird Datenaufbewahrung und Dateifreigabe verstanden. Was Benutzer eines solchen Dienstes allerdings nicht unbedingt beachtet, ist die Frage der Sicherheit der Daten. Worauf die Arbeit abzielt, ist eine Untersuchung der Kombinierbarkeit der Komponenten: entfernter Speicher, Datenübertragung über ein offenes Netz, Datenaufbewahrung und Dateifreigabe in Hinsicht auf Sicherheit und eine darauf basierende Entwicklung und Programmierung eines Softwaremoduls.

1.2. Gliederung der Arbeit

Die Arbeit wird in folgende Abschnitte gegliedert:

Das Kapitel 2 behandelt Grundlagen, Definitionen und Voraussetzungen, auf die im weiteren Verlauf verwiesen und aufgebaut wird. Es wird auf allgemeine Begriffe wie Client-Server-Architektur, Sicherheit, Kryptographie, und auf spezifische Begriffe wie entfernter Speicher und die aktuelle Entwicklungsumgebung eingegangen.

Im Kapitel 3 wird eine Analyse einiger aktueller Anbieter des Speicherdienstes in Bezug auf die in Grundlagen identifizierten Sicherheitsmaßnahmen vorgenommen und eine Wertung dazu abgegeben.

Das Kapitel 4 wird aufgrund der Analyse relevante Schlussfolgerungen ziehen, und in Bezug auf die in Grundlagen identifizierten Sicherheitsmaßnahmen die Probleme formulieren und mit der Umsetzung einer konkreten Idee versuchen, zu deren Lösung beizutragen.

Das Kapitel 5 wird die im Kapitel Umsetzung dargestellte Lösungsidee durch Realisierung eines prototypischen Softwaremoduls unterstützen.

Das Kapitel 6 wird einen Test des im Kapitel Realisierung dargestellten Softwaremoduls demonstrieren und die wichtigen Stationen im Ablauf mit Screenshots und Beschreibungen kommentieren.

Das Kapitel 7 wird eine Zusammenfassung dieser Arbeit enthalten.

2. Grundlagen

2.1. Client-Server-Architektur

Rechnersysteme, die miteinander kommunizieren wollen, können nach dem Client-Server-Modell organisiert werden. Der Server ist dabei eine Instanz, die Dienste anbietet und auf Abruf aktiviert wird. Ein Client ist diejenige Instanz, die die Dienste des Servers abruft bzw. nutzt. Die Kommunikation zwischen den beiden Instanzen verläuft über eine Verbindung, die in ihrem Ursprung als unsicher gilt [6]. Inwieweit die Kommunikationsinstanzen für sich selbst sicher oder unsicher sind, ist aus Bedrohungsszenarien abzuleiten, auf die hier nicht weiter eingegangen wird. Sie werden als vorgegeben angenommen, sodass wenn Sicherheit beim Einsatz eines Client-Server-Models ein Teil der Architektur sein soll, festgestellt werden muss, wie diese Sicherheit in genügendem Maße definiert werden kann, was zur Formulierung der Sicherheitsmerkmale führt.

2.2. Sicherheitsmerkmale

2.2.1. Zugriffskontrolle

Wenn ein Client auf bestimmte Dienste eines Servers zugreifen möchte, sind aus Schutzzwecken Mechanismen notwendig, die steuern, wie der Zugriff erlaubt ist. Das bedeutet die Vergabe von Zugriffsrechten und Kontrolle der Autorisierung [6]. Der Einsatz dieser Mechanismen liegt im Verantwortungsbereich des Servers.

2.2.2. Authentizität

Wenn ein Client von einem Server einen Dienst anfragt, kann es von Bedeutung sein, dass der Server den Client eindeutig identifizieren kann, um entweder Zugriffsrechte auf bestimmte Dienste zu erteilen oder auch Missbrauch zu unterbinden. Dazu ist die Authentizität des Clients nachzuweisen, was beim Anmelden/Einloggen am Server stattfindet. Ein Client muss dazu definierte Eigenschaften vorweisen, die zusammen mit geeigneten Prüfmechanismen

eine Identifizierung erlauben [6]. Das soll mit Kryptographie möglich sein (Digitales Zertifikat 2.4.2.2).

2.2.3. Vertraulichkeit

Ein Server kann als Dienstleister mehrere Clients bedienen. Dabei sind die Clients voneinander unabhängig und sollen als eigenständige Instanzen gegen unautorisierte Offenlegung der privaten Daten geschützt werden. Entsprechende Maßnahmen sind sowohl durch den Server als auch durch den Client umsetzbar. Das soll mit Kryptographie möglich sein (Kryptographische Primitive 2.4.1).

2.2.4. Integrität

Ein Server kann als Dienstleister mehrere Clients bedienen. Es kann eine Situation auftreten, in der ein Client eine zweifelsfreie Bestätigung braucht, dass bestimmte Daten echt bzw. das Original sind, denn wenn mehrere Benutzer einen Zugang zum System haben, ist eine unautorisierte Datenmanipulation denkbar, die erkannt werden soll [6]. Das soll mit Kryptographie möglich sein (Hashfunktion 2.4.1.4).

Zusammenfassung

Die erwähnten Sicherheitsmerkmale werden im Kontext dieser Arbeit für relevant gehalten und stellen entsprechend keinen Anspruch auf Vollständigkeit im Sinne aller möglichen Sicherheitsmaßnahmen für rechnergestützte Kommunikationssysteme.
Die Festlegung dieser Sicherheitsmerkmale führt zur Formulierung von Sicherheitsanforderungen an ein konkretes System.

2.3. Sicherheitsanforderungen

Da diese Arbeit im Kontext von Client-Server-Architekturen entsteht 2.1, wird im weiteren nur auf Sicherheitsmechanismen innerhalb dieses Kontextes eingegangen. Eine Sicherheitsarchitektur wird immer ausgehend davon entworfen, welche Art von Bedrohung für das System in Betracht kommt [10] bzw. welche Sicherheitsmerkmale 2.2 erfüllt werden sollen.
Aus den Sicherheitsmerkmalen sollen Sicherheitsanforderung abgeleitet werden.

2.3.1. Sicheres Anmelden/Einloggen

Ein Anmelden und Einloggen sorgt für die Identifikation und Autorisation zu bestimmten Rechten innerhalb eines System. Die Authentizität der Benutzer bzw. der Clients ist für den Aufbau und die Funktionsweise des System zu garantieren (Authentizität 2.2.2).

2.3.2. Sicherer Übertragungskanal

Ein sicherer Übertragungskanal kann gleich mehrere sicherheitsrelevante Maßnahmen durchsetzen: Authentizität 2.2.2, Vertraulichkeit 2.2.3 und Integrität 2.2.4 [6]. Falls also ein Datentransfer über ein offenes Netzwerk stattfindet, ist darauf zu achten, dass ein sicherer Übertragungskanal zum Einsatz kommt.

2.3.3. Sichere Datenaufbewahrung

Die Aufbewahrung der Daten auf einem entfernten Speicher wurde in der Einleitung wegen der steigenden Netzwerkgeschwindigkeit und der sinkenden Investitionsanforderungen als eine Alternative zu einem lokalen Speicher, wie die Festplatte oder der Flashspeicher, genannt. Auf der lokalen Festplatte werden die Daten für sicher gelagert gehalten, wenn der Rechner, in dem sie installiert ist, als Benutzer interpretiert wird. Es soll davon abstrahiert werden, dass ein Rechner auch als Mehrnutzersystem eingerichtet werden kann. Die Sicherheit der Datenaufbewahrung auf dem lokalen Speicher (Festplatte) soll analog auf den Einsatz eines entfernten Speichers übertragen werden, sodass für den Benutzer transparent bleibt, ob der lokale oder der entfernte Speicher zum Einsatz kommt. Um die Analogie erfüllen zu können, müssen die Daten auf dem entfernten Speicher für unberechtigte Benutzer genauso unerreichbar sein wie die Daten auf dem lokalen Speicher (Vertraulichkeit 2.2.3).

2.3.4. Sichere Dateifreigabe

Für den Datenaustausch zwischen verschiedenen Benutzern ist die Dateifreigabe zuständig. Da ein entfernter Speicher permanent verfügbar sein soll, ist es sinnvoll die Dateifreigabe darüber einzurichten. Es müssen Leserechte (Vertraulichkeit 2.2.3), Schreibrechte und Löschrechte (Zugriffskontrolle 2.2.1) geregelt werden. Das heißt, die Benutzer müssen verwaltet werden. Es darf nicht möglich sein, fremde Daten zu löschen oder zu verändern. Außerdem darf es nicht möglich sein, auf fremde Daten ohne eine explizite Erteilung der Rechte zugreifen zu können.

Zusammenfassung

Die dargestellten Sicherheitsanforderungen werden für das System für notwendig gehalten, und die Kryptographie wird als ein Mittel angenommen, mit dessen Hilfe diese Sicherheitsanforderungen erreicht werden können.

2.4. Kryptographie

2.4.1. Kryptographische Primitive

2.4.1.1. Symmetrische Verschlüsselung

Bei dem symmetrischen Verschlüsselungsverfahren wird sowohl für die Verschlüsselung als auch für die Entschlüsselung der gleiche Schlüssel verwendet. Die beiden Operationen können effizient in Hardware und Software implementiert werden. [6]. Wegen der Effizienz sind die Verfahren geeignet, große Datenmengen zu verarbeiten, so dass sie für die Erfüllung der Sicherheitsanforderungen aus 2.3 eingesetzt werden können.

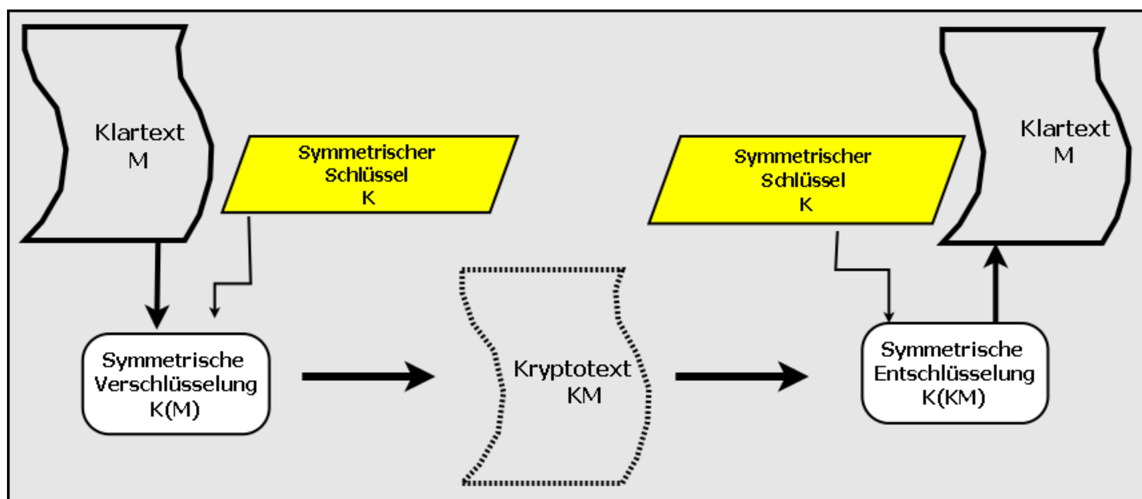


Abbildung 2.1.: Symmetrische Verschlüsselung[6]

Blockchiffren

Die Blockchiffren arbeiten auf Eingabedaten fester Länge - auf den Blöcken. Jeder Block wird auf gleiche Weise verschlüsselt. Diese Voraussetzung führt dazu, dass ein Klartext bei der Eingabe in Blöcke aufgeteilt werden muss. Das bedeutet, der Klartext kann ein vielfaches der Blockgröße sein. Da dies aber keine Einschränkung für die Anwendung der Algorithmen sein soll, wird ein *Padding* verwendet. Der Padding füllt den letzten unvollständigen Block bis auf die vorgegebene Länge auf [6].

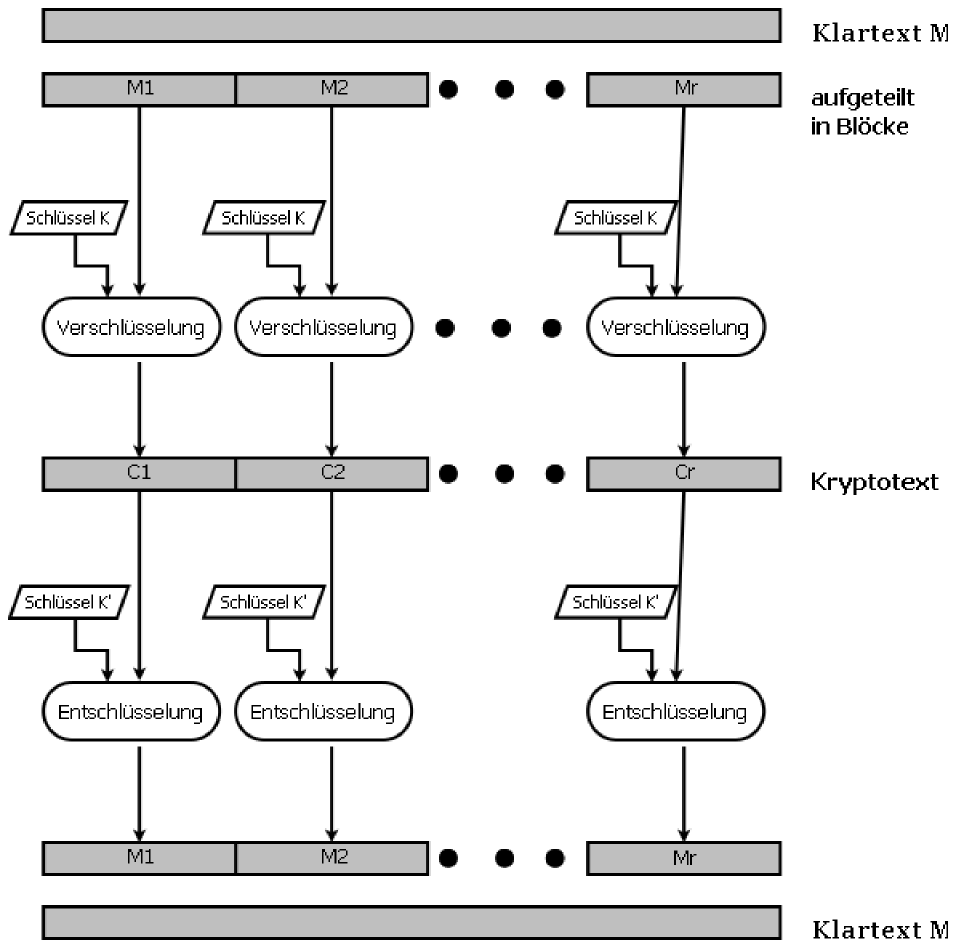


Abbildung 2.2.: Blockchiffren[6]

Stromchiffren

Die Stromchiffren verarbeiten die Eingabedaten zeichenweise. Dazu müssen der Encoder und der Decoder jeweils den gleichen Pseudozufallszahlen-Generator besitzen. Die Pseudozufallszahlenfolge wird ausgehend von einem Initialwert generiert. Damit die Entschlüsselung möglich ist, müssen sich der Encoder und der Decoder über den Initialwert einigen [6].

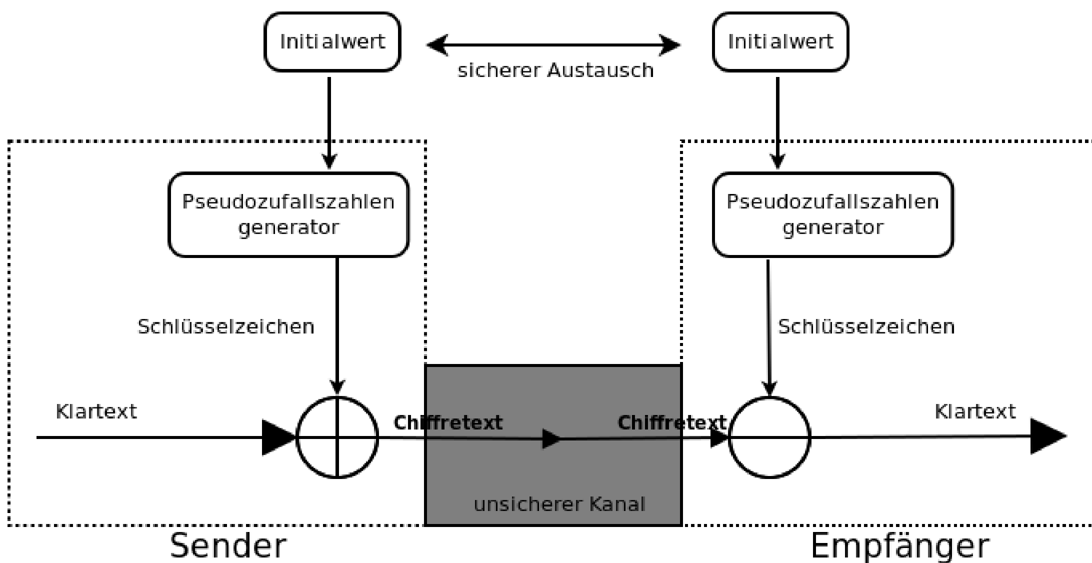


Abbildung 2.3.: Stromchiffren[6]

2.4.1.2. Asymmetrische Verschlüsselung

Bei dem asymmetrischen Verschlüsselungsverfahren wird ein Schlüsselpaar verwendet, das aus einem privaten und einem öffentlichen Schlüssel besteht. Der zentrale Gedanke, der hier zu verstehen ist, liegt darin, dass es nicht wirklich um Verschlüsselung bzw. Entschlüsselung eines Klartextes geht, sondern um die abwechselnde Anwendung der Schlüssel. Das heißt, wenn zuerst der private Schlüssel angewendet wird, muss danach auf das Chiffrat der öffentliche Schlüssel angewendet werden, damit die Anfangseingabedaten wieder erzeugt werden. Es gilt auch die Umkehrung dieser Reihenfolge. Je nach Wahl der Reihenfolge ergeben sich verschiedene Einsatzbereiche.
Der Nachteil der asymmetrischen Verschlüsselungsverfahren liegt darin, dass sie erheblich

langsamer sind als die symmetrischen Verschlüsselungsverfahren [6]. Asymmetrische Verfahren können für die Erfüllung der in 2.3 Sicherheitsanforderungen sorgen.

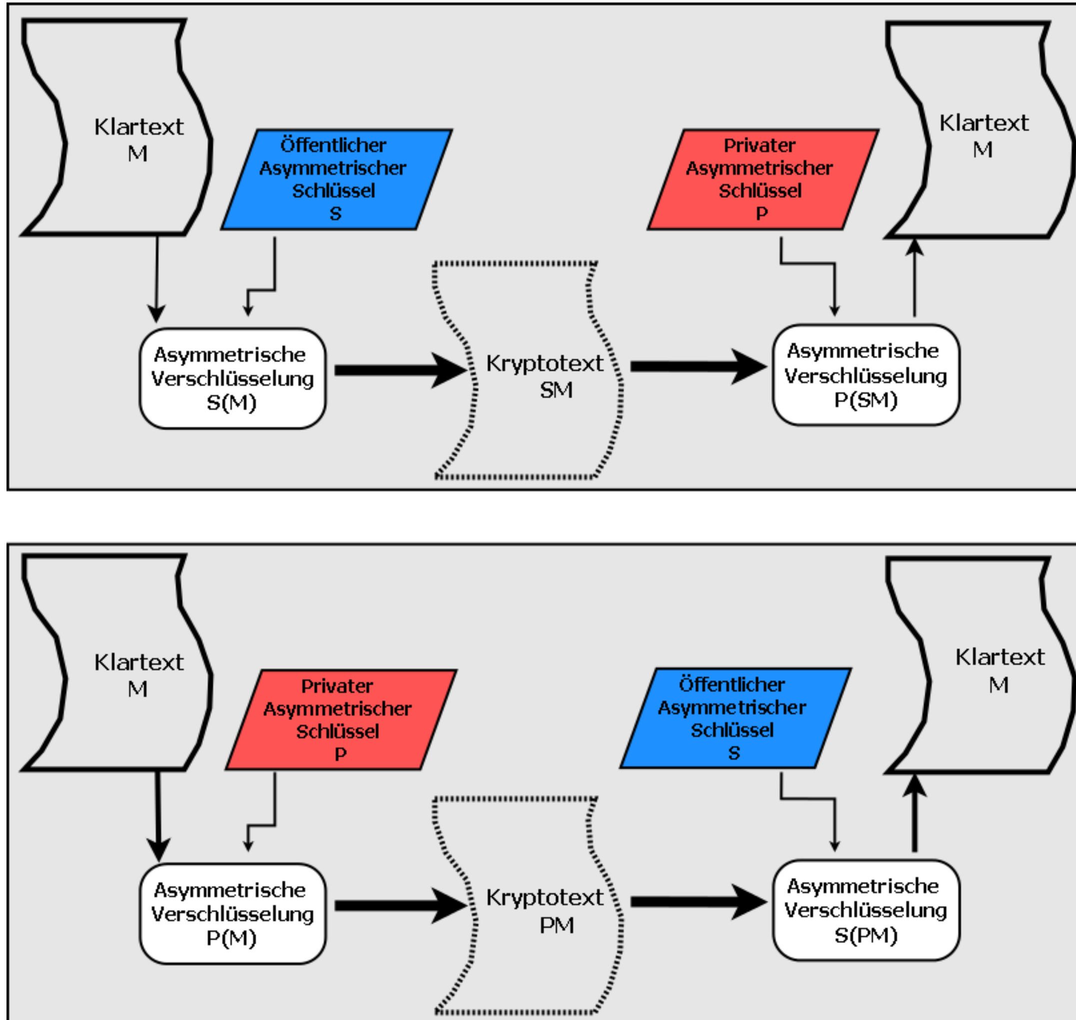


Abbildung 2.4.: Zwei Reihenfolgen der asymmetrischen Verschlüsselung[8]

2.4.1.3. Hybride Verschlüsselung

Die hybriden Verschlüsselungsverfahren sind eine Kombination aus den symmetrischen und asymmetrischen Verschlüsselungsverfahren. Ihr Ziel ist es, die Vorteile der beiden zugrunde liegenden Verfahren auszunutzen. Die symmetrische Verschlüsselung ist effizient, sodass große Datenmengen damit bearbeitet werden können, allerdings erfordert sie einen Schlüsselaustausch, der auf einem geheimen Weg über ein offenes Netzwerk stattfinden muss.

Das asymmetrische Verschlüsselungverfahren ist so langsam, dass es auf große Daten nicht anwendbar ist, aber durch seinen Einsatz wird das Problem des Schlüsselaustausches für symmetrische Verschlüsselung gelöst. Wenn der symmetrische Schlüssel mit einem öffentlichen Schlüssel chiffriert und über ein offenes Netzwerk verschickt wird, ist nur derjenige Empfänger in der Lage ihn zu dechiffrieren, der den zu dem öffentlichen Schlüssel passenden privaten Schlüssel besitzt. Die hybriden Verschlüsselungsverfahren sollen zur Erfüllung der Sicherheitsanforderungen 2.3 beitragen.

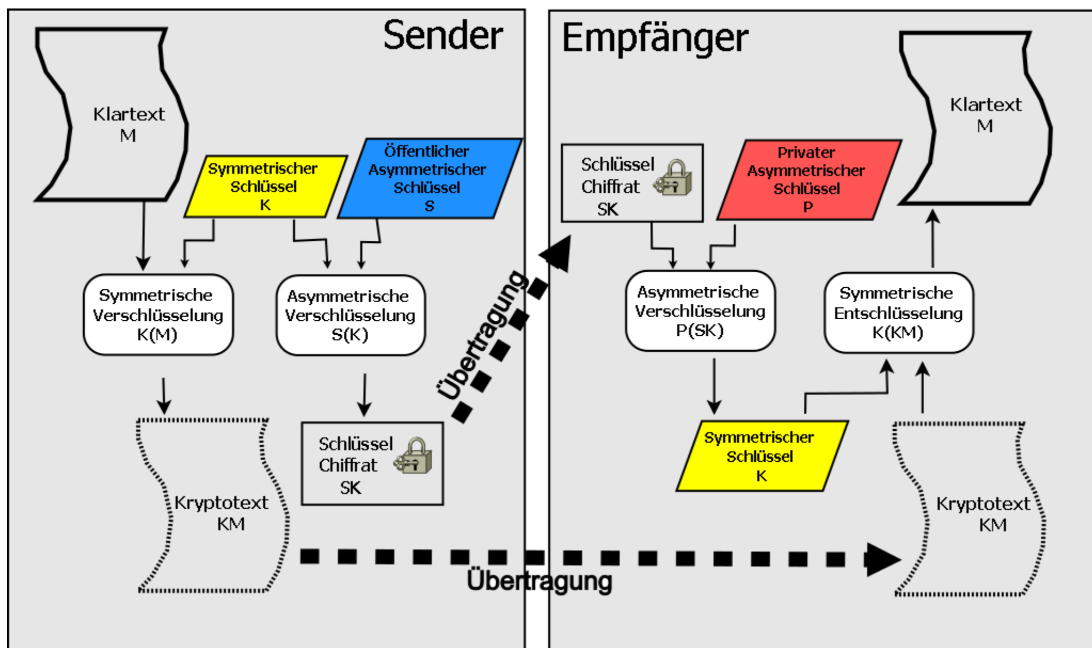


Abbildung 2.5.: Hybride Verschlüsselung[7]

2.4.1.4. Hashfunktion

Mit einer Hashfunktion wird ein Wert berechnet, der ein Objekt eindeutig charakterisiert. Somit ist es möglich mit Hilfe des Hash-Wertes die Integrität der Daten zu gewährleisten 2.2.4. Der Hash-Wert von zwei identischen Objekten ist gleich, und es ist unmöglich aus einem Hash-Wert das Objekt abzuleiten, aus dem es erstellt wird. Andererseits darf eine Hashfunktion niemals aus zwei verschiedenen Objekten den gleichen Hash-Wert erzeugen. Dies ist jedoch möglich. Dieses wird Kollision genannt und durch zwei Strategien behandelt: Kollisionsbehandlung und Kollisionsvermeidung. Für die Überprüfung der Eindeutigkeit eines

Objekts ist die Strategie der Kollisionsvermeidung einzusetzen. Es sind kollisionsresistente Einwegfunktionen notwendig, bei denen die Wahrscheinlichkeit der Kollision für klein genug geschätzt wird. Diese Schätzung ändert sich entsprechend den Sicherheitsanforderung und dem Stand der Technik. Für langfristige Sicherheit wird eine Hashwertlänge von mindestens 256 Bit empfohlen [6]. Die Hashwertbildung kann im Anwendungsfall einer digitalen Signatur 2.4.2.1 eingesetzt werden.

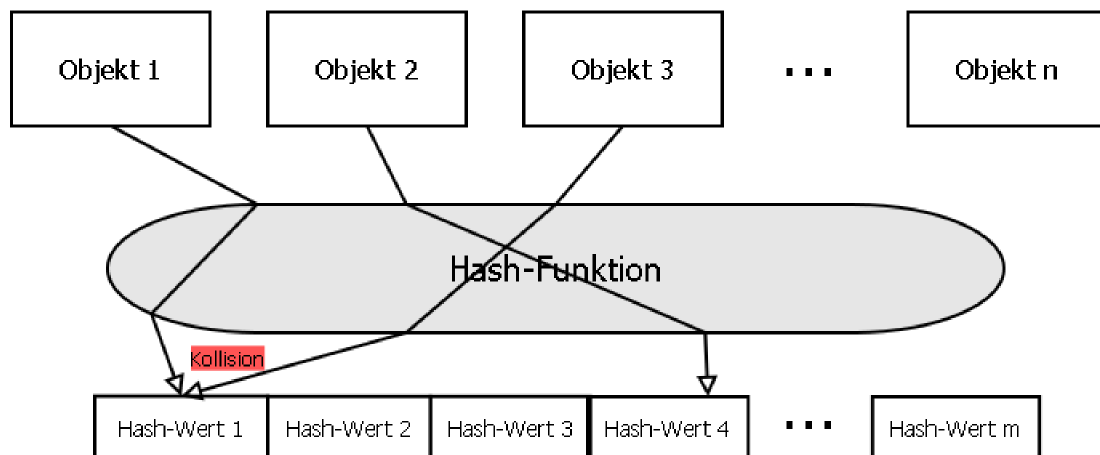


Abbildung 2.6.: Nichtkollisionsresistente Hashfunktion[13]

2.4.2. Anwendungsfälle

2.4.2.1. Digitale Signatur

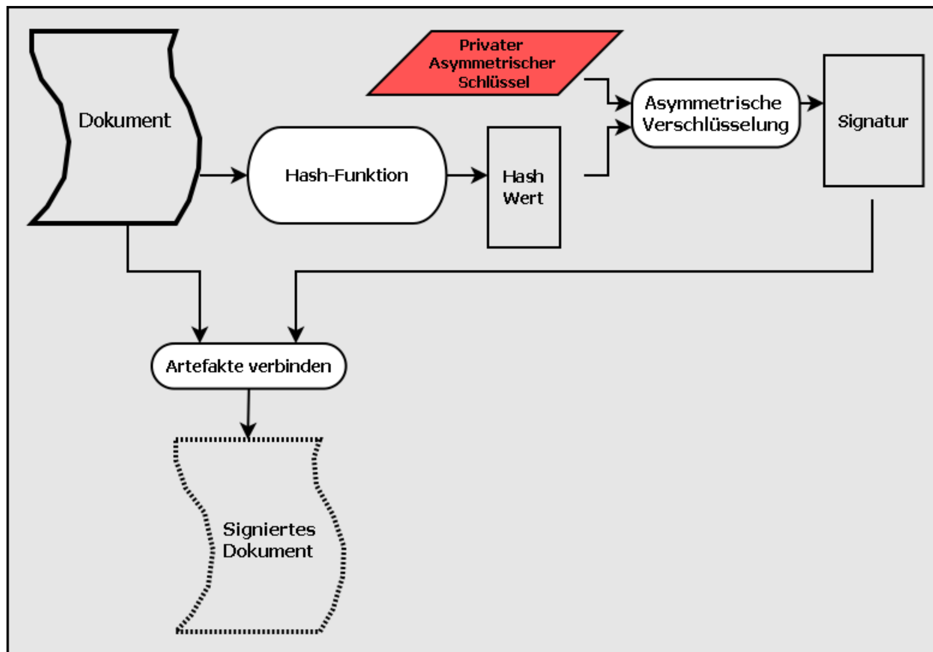


Abbildung 2.7.: Digitale Signatur: Dokument signieren[1]

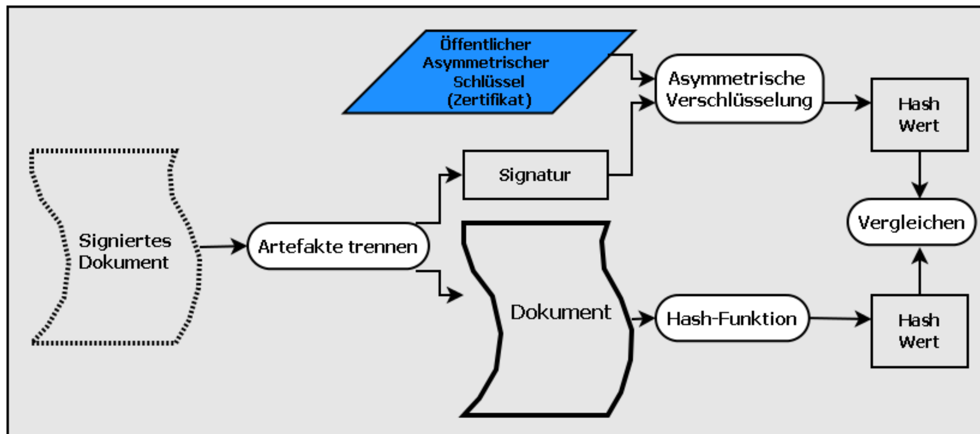


Abbildung 2.8.: Digitale Signatur: Dokument verifizieren[1]

Mit Hilfe der digitalen Signatur ist es möglich den Eigentümer der Nachricht zu bestätigen. Das Verfahren basiert auf dem asymmetrischen Verschlüsselungsverfahren. Der Eigentümer einer Nachricht bildet einen Hash-Wert von der Nachricht und wendet anschließend seinen privaten Schlüssel auf diesen Hash-Wert an. Das Ergebnis ist eine Digitale Signatur, die zusammen mit der Nachricht an den Empfänger gesendet wird. Der Empfänger muss einen Hash-Wert von der empfangenen Nachricht berechnen und parallel dazu den öffentlichen Schlüssel des Eigentümers auf die empfangene Signatur anwenden. Als Ergebnis beider Vorgänge stehen zwei Hash-Werte zum Vergleich an. Die Gleichheit garantiert die Integrität der Nachricht. Damit auch die Authentizität der Nachricht festgestellt werden kann, muss das asymmetrische Schlüsselpaar eindeutig zu einer Quelle bzw. einer Person zugeordnet werden können. Für diese Aufgabe ist eine Vertrauensinstanz notwendig, die die Zuordnungen von Schlüsseln zu ihren Besitzern verwaltet. Der Sinn der digitalen Signatur soll in der zu der Handunterschrift gleichwertigen Lösung für digitale Dokumente sein.[10]. Mit Hilfe der digitalen Signatur soll das Sicherheitsmerkmal 2.2.4 Integrität und die daraus entstandenen Sicherheitsanforderungen erfüllt werden.

2.4.2.2. Digitales Zertifikat

```
Certificate:
Data:
Version: 0 (0x0)
Serial Number: 988 (0x3dc)
Signature Algorithm: md5withRSAEncryption
Issuer: C=ZA, SP=Western Cape, L=Cape Town, O=Thawte Consulting
cc, OU=Certification Services Division, CN=Thawte Server
CA/Email=server-certs@thawte.com
Validity
Not Before: May 26 16:57:21 1997 GMT
Not After : May 26 16:57:21 1998 GMT
Subject: C=DE, SP=Bavaria, L=Regensburg, O=WWW-Service AG,
OU=Secure Service, CN=www.www-service.de

Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Modulus:

00:de:16:f9:6a:73:59:9f:b6:f5:48:56:e6:30:f7:
d2:e0:3f:97:d8:6b:68:27:8a:cf:29:d4:9e:fb:d1:
38:38:42:d7:da:c6:94:9d:49:98:33:cf:bb:e8:b2:
6a:36:34:98:23:f1:4b:17:c7:3e:a0:5d:0f:04:09:
69:aa:a3:53:5b

Exponent: 65537 (0x10001)
Signature Algorithm: md5withRSAEncryption

30:42:20:b2:7e:3d:a4:63:44:1c:b7:08:ad:b6:a6:c5:a8:c3:
40:12:45:3c:34:9d:8b:a6:de:47:1b:3b:a9:c4:ac:63:8a:e0:
cb:dd:ac:22:35:12:16:d5:1b:c7:8a:9d:1b:5a:6b:84:1e:f3:
d5:88:4c:33:d7:49:3c:8b:97:6f:74:16:70:42:92:7e:14:a1:
af:5b:08:5e:b0:c3:58:88:4a:16:d9:56:c9:74:91:fa:fe:31:
89:51:ad:7f:37:c2:68:6f:5e:c3:95:d0:bd:13:0d:5e:35:e3:
5e:b8:39:87:64:f1:87:54:95:2c:1c:4f:6e:22:70:64:e0:f9:
b5:c1
```

Abbildung 2.9.: Digitales Zertifikat nach X.509-Standard [2]

Das digitale Zertifikat sorgt für eine Zuordnung eines öffentlichen Schlüssels zu einem Benutzer des Systems. Es wird durch eine Certification Authority (CA) ausgestellt. Der Benutzer selbst wird durch einen privaten Schlüssel identifiziert [10]. Je nach Abstraktionsebene kann diese Aufgabe mit der Benutzerverwaltung zusammengelegt werden, weil sowohl die CA als auch die Benutzerverwaltung als Vertrauensinstanzen angesehen werden. Des Weiteren verfügt die Vertrauensinstanz über ihr eigenes Zertifikat, mit dem sie sich authentifiziert. Zur Vollständigkeit muss gesagt werden, dass es nicht darauf eingegangen wird, von wem die CA/-Benutzerverwaltung zertifiziert wird. Dies ist nicht relevant, da die CA/Benutzerverwaltung von Anfang an als vertrauenswürdig angesehen wird. Das Prinzip dieses Anwendungsfalls soll zur Erfüllung des Sicherheitsmerkmals 2.2.2 Authentizität und der daraus abgeleiteten Sicherheitsanforderungen in einer vereinfachten Form eingesetzt werden.

2.4.2.3. SSL/TLS

Das Protokoll wird zur Sicherung des Übertragungskanals und zur serverseitigen (und optional zur clientseitigen) Authentisierung eingesetzt. Dabei schützt die Authentisierung die Endsysteme innerhalb der Client-Server-Architektur, und die Verschlüsselung des Übertragungskanals sorgt für die Vertraulichkeit bei der Kommunikation [6]. Der Anwendungsfall wird zur Erfüllung der Sicherheitsanforderung 2.3.2 Sicherer Übertragungskanal eingesetzt.

Zusammenfassung

Aufgrund der Eigenschaften der kryptographischen Primitive und ihrer Anwendungsfälle wird es für möglich gehalten, die Gesamtheit der Sicherheitsanforderungen an eine Client-Server-Architektur mit ihrer Hilfe im Kontext dieser Arbeit zu erfüllen.

2.5. Zu gesetzlichen Regelungen

Sowohl der Dienstanbieter als auch der Benutzer des Dienstes sind an bestimmte gesetzliche Regelungen gebunden, die je nach Land variieren und hier nur angedeutet werden sollen, ohne auf exakte Darstellung und Ausnahmen einzugehen.

- Falls ein Benutzer fremde, ihm anvertraute Daten verwaltet, hat er für ihre Sicherheit zu sorgen. Falls die Daten auf einem entfernten Speicher aufbewahrt werden, ist der Benutzer trotzdem verantwortlich, obwohl er sich auf den Dienstanbieter verlässt [4].
- In den USA sollen *The Patriot Act* und *The Fourth Amendment* unter bestimmten Umständen die Dienstanbieter dazu zwingen können, die Daten ihrer Benutzer zur Verfügung zu stellen, auch wenn die Daten unter anderem in Europa oder Deutschland aufbewahrt werden [4].

Die Schlussfolgerung ist, dass der Benutzer selbständig für die Sicherheit der Daten sorgen sollte, falls es möglich ist, statt sich auf den Dienstanbieter zu verlassen.

2.6. Der entfernte Speicher

2.6.1. Anwendungsfälle eines entfernten Speichers

Cloud-Anbieter stellen vielfältige Dienste zur Verfügung. Es soll in dieser Arbeit auf den Dienst des Speichers eingegangen werden. Dementsprechend wird mit jedem Bezug auf

Cloud-Anbieter bzw. Cloud-Dienste nur der Speicher (als abstraktere Bezeichnung: entfernter Speicher) gemeint. Die Netzwerk-Transferraten werden immer höher. Die Preise für den Speicher sinken. Wenn noch in Betracht gezogen wird, dass die Daten von überall erreichbar sind, steigt die Attraktivität dieser Art der Dateilagerung, und sie wird zur einer Alternative zur lokalen Speicherung auf der Festplatte oder dem Flashspeicher.

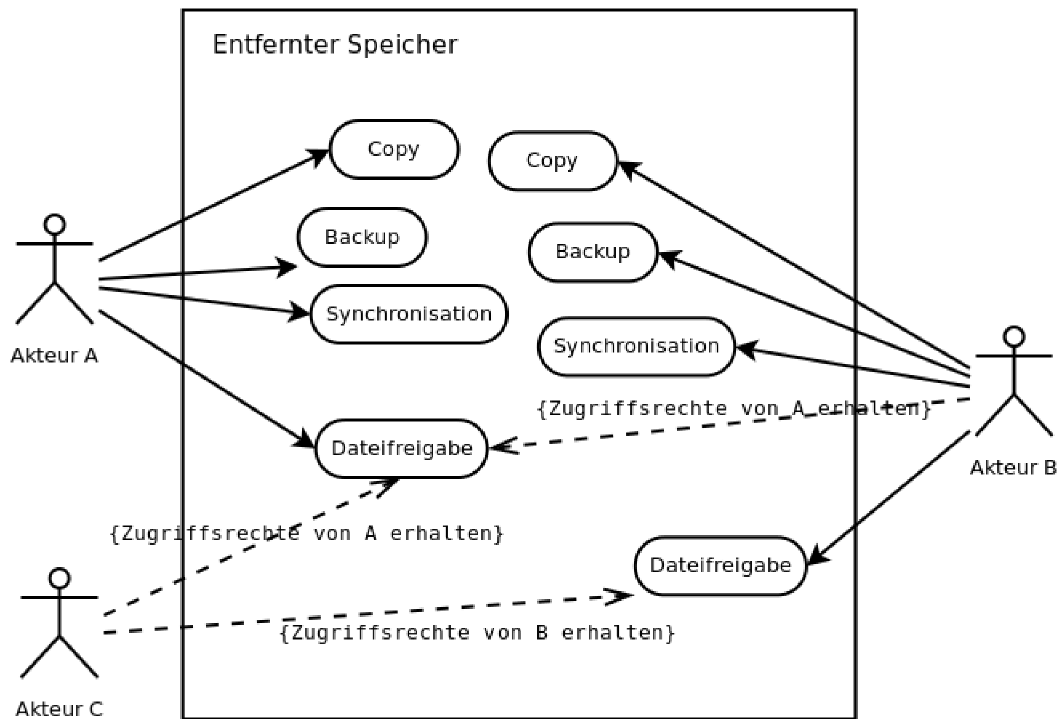


Abbildung 2.10.: Anwendungsfälle für einen entfernten Speicher

Copy

Die letzte Version von Daten wird von einem lokalen Gerät auf den entfernten Speicher übertragen.[4]

Backup

Alle Versionen von Daten werden auf den entfernten Speicher übertragen und da verwaltet.[4]

Synchronisation

Endgeräte, die mit dem entfernten Speicher verbunden sind, werden synchronisiert, sodass jedes Endgerät die gleichen Daten enthält.[4]

Sharing
: Benutzer, die mit dem entfernten Speicher verbunden sind, haben Zugriff auf die Daten eines anderen Benutzers unter der Voraussetzung, dass der Zugriff erlaubt wurde.[4]

Die Betrachtung der Anwendungsfälle eines entfernten Speicher hat insofern eine Bedeutung, dass in der Umsetzungsphase und in der Realisierungsphase der clientseitigen Sicherheitsarchitektur darauf zurückgegriffen wird.

2.6.2. Kommunikationsschnittstellen zu einem entfernten Speicher

Für die praktische Umsetzung der Kommunikation zu einem entfernten Speicher stehen generell folgende Schnittstellen zur Verfügung, die je nach Anbieter zum Einsatz kommen können:

Anbietereigene Endbenutzersoftware
: Die Endbenutzersoftware ist vom Dienstanbieter entsprechend den Möglichkeiten des Dienstes entworfen. [4]

Internet-Browser
: Aufgrund der Tatsache, dass Internet-Browser auf den meisten Endbenutzersystemen installiert sind, ist der Zugriff auf die Dienste relativ systemunabhängig, allerdings sind nur diejenigen Dienste verfügbar, die im Browser durch seine Architektur und Protokolle technisch umsetzbar sind. [4]

API
: Der Dienstanbieter stellt ein SDK[1] zur Verfügung, mit dessen Hilfe ein Software-Entwickler die Dienste in das eigene Programm einbringen und entsprechend den Anforderungen des Programms einsetzen kann. [4]

2.7. Entwicklungsumgebung

Die Arbeit wird auf einem Linux-Rechner erstellt. Die Schreibumgebung ist Texmaker mit LaTeX. Die Diagramme werden mit Dia gezeichnet. Die Software-Entwicklung findet mit Eclipse 3.7 statt. Die Programmiersprache ist Java. Es werden Android-SDK [14] für den Einsatz auf einer Android-Plattform 2.7.1 und SpongyCastle[2] [12] als kryptographischer Provider für

[1] Software Developement Kit
[2] angepasste Bouncy Castle[5]

JCA/JCE 2.7.2 eingebunden. Die Android-SDK bietet einen Emulator an, in dem die Anwendung ausgeführt wird.

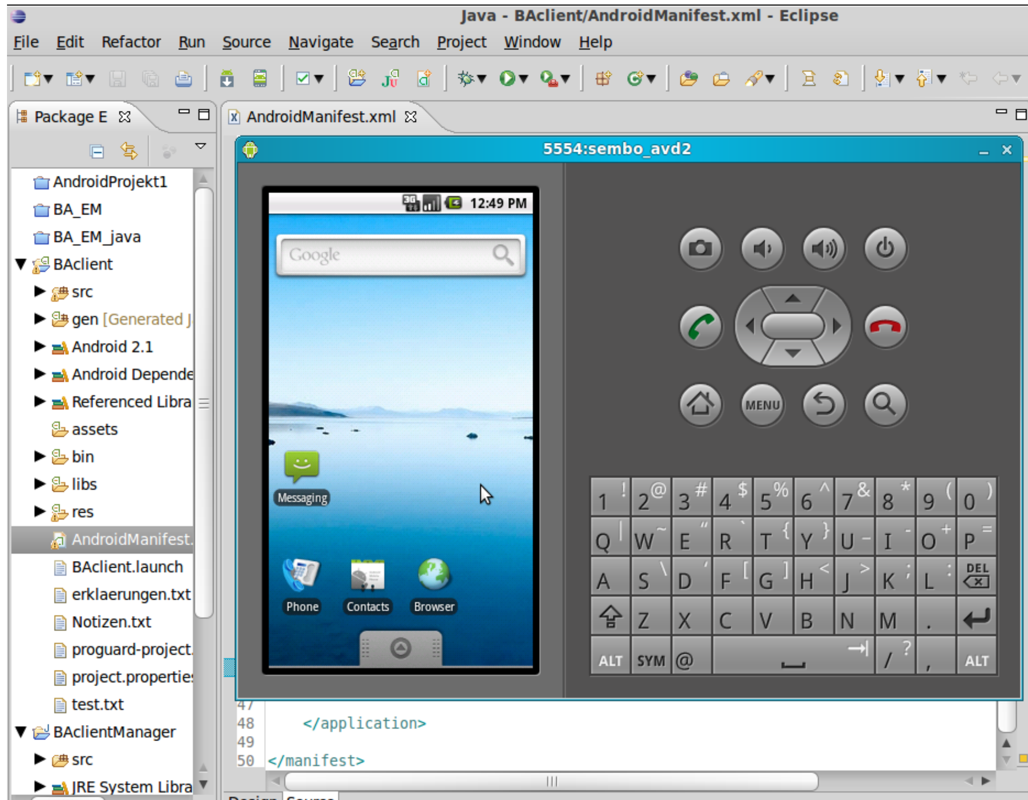


Abbildung 2.11.: Eclipse IDE mit Android-Emulator

2.7.1. Android-Plattform

[14]
Android ist ein Mehrbenutzer-Betriebssystem, das auf Linux aufgebaut ist, und in dem jede Anwendung einen Benutzer darstellt und in einem eigenen Prozess läuft. Jeder Prozess und somit jede Anwendung wird in einer isolierten Laufzeitumgebung (Virtual Machine) ausgeführt. Auf diese Weise kann eine Anwendung auf keine Systembereiche zugreifen, für die sie keine Rechte besitzt. Die Anwendungen sind durch

- Vergabe der gleichen systeminternen Benutzer-ID und somit durch eine Ausführung in der gleichen Laufzeitumgebung
- Gewährung der Rechte während der Installation

in der Lage Daten auszutauschen.

Android-Anwendungen werden mit Hilfe von vier vordefinierten Komponenten aufgebaut:

2.7.1.1. Activity

Activity ist eine graphische Benutzerschnittstelle (GUI). Das, was auf einem Bildschirm zu einem bestimmten Zeitpunkt angezeigt wird, ist eine einzelne Activity, sodass eine Anwendung aus vielen Activities bestehen kann. Programmiertechnisch ist das eine Ableitung von der *Activity*-Klasse.

2.7.1.2. Service

Mit einem Service werden Hintergrundprozesse bzw. lang laufende Prozesse realisiert, die keine Interaktion mit dem Benutzer erfordern. Programmiertechnisch wird von der *Service*-Klasse abgeleitet.

IntentService
: IntentService ist ein Service, der alle seine Aufgaben einer Warteschlange hinzufügt und entsprechend der Reihenfolge erfüllt. Somit ist kein Multithreading erforderlich.

2.7.1.3. Content Provider

Content Provider übernehmen die Aufgaben der persistenten Speicherung und des Auslesens der Daten innerhalb des Systems oder einer Anwendung und sind Unterklassen von *ContentProvider*.

2.7.1.4. Broadcast-Receiver

Ein Broadcast-Receiver empfängt und beantwortet System-Events, indem dieser darauf mit Nachrichten oder Initiierung einer Anwendung reagiert. Er wird von der *BroadcastReceiver*-Klasse abgeleitet.

2.7.1.5. Intent-Objekt

Die Komponenten einer Anwendung können aus anderen Anwendungen aufgerufen werden. Sie werden aber nicht in dem Prozess der initiierenden Anwendung sondern in dem Prozess der Anwendung ausgeführt, dem die Komponenten gehören. Der Aufruf wird nicht direkt an eine Komponente sondern an das Android-System gerichtet, der die Komponente je nach

Zugriffsrechten ansteuert. Eine Anwendung signalisiert während der Laufzeit ihren Bedarf an einer anderen Komponente mit Hilfe eines *Intent*-Objekts.

2.7.1.6. AndroidManifest.xml

Die Datei *AndroidManifest.xml* enthält startrelevante Einstellungen. Alle Komponenten der Anwendung müssen hier deklariert werden. Des Weiteren wird hier festgelegt, welche Zugriffsrechte und Hardwarekomponenten die Anwendung benötigt.

2.7.2. Java Cryptography Architecture und Java Cryptography Extension

Java Cryptography Architecture und Java Cryptography Extension bieten die Basisfunktionalität für Kryptographie in Java. Sie abstrahieren von konkreten Algorithmen und definieren die Schnittstellen (API) für kryptographische Verfahren, an die sich die Anbieter der Verfahren halten müssen. Der Softwareentwickler, der die Kryptographie einsetzt, bedient sich den von JCA und JCE angebotenen Methoden, ohne auf konkrete Implementierung der Algorithmen eingehen zu müssen. Die Anbieter, genannt Provider, werden den JCA/JCE hinzugefügt.[9]

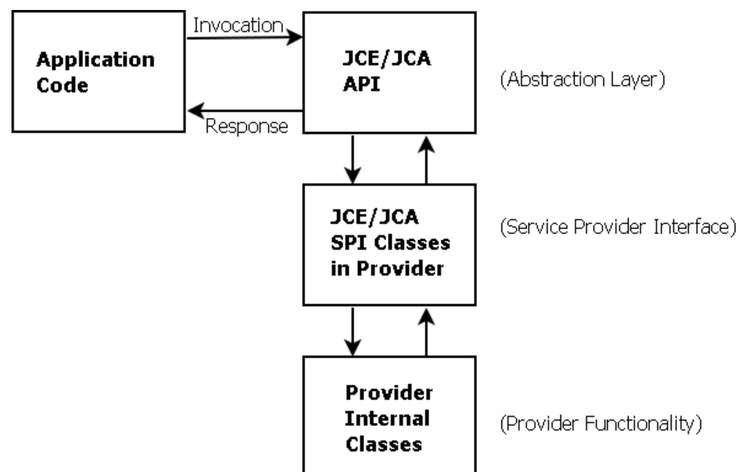


Abbildung 2.12.: JCA/JCE Architektur [9]

3. Analyse

3.1. Sicherheitsanforderungen beim Einsatz eines entfernten Speichers

Der entfernte Speicher steht als Service zur Verfügung. Der Service kann sowohl von einzelnen Personen für private Zwecke als auch von Unternehmen als Teil ihrer Betriebsinfrastruktur eingesetzt werden. Daraus resultieren verschiedene Anforderungen an den Service-Anbieter. Der Benutzer eines solchen Service ist je nach Sensibilität der Daten gezwungen, die Anbieter zu analysieren, ob sie seine Anforderungen erfüllen. Eine Alternative zu der Analyse und dem vollständigen Verlass auf die Kompetenz der Anbieter ist der Einsatz eines eigenes Sicherheitssystems. Es soll untersucht werden, inwiefern die Anbieter die Anforderungen im Kontext dieser Arbeit zufriedenstellend erfüllen, und ob die Alternative mit dem eigenen Sicherheitssystem sinnvoll ist.

Es wird davon ausgegangen, dass ein Speicherdienst nicht ein Teil der Rechner-Infrastruktur ist, die im Administrationsbereich des Benutzers liegt. Dieser ist komplett abgekoppelt, sodass der Benutzer keine Eingriffsmöglichkeiten in die Sicherheitssysteme hat.
Es gibt eine Reihe von Anbieter solcher Dienste. Durch die Analyse der für diese Arbeit relevanten Sicherheitsmerkmale dieser Dienste, soll aufgezeigt werden, dass der Aufbau einer clientseitigen Sicherheitsarchitektur von Bedeutung ist. Des Weiteren soll unterschieden werden, welche Sicherheitsmerkmale mit der clientseitigen Sicherheitsarchitektur erfüllbar sind und welche nur durch aktive Mitarbeit des Dienstanbieters erreicht werden können.

3.1.1. Sicheres Anmelden/Einloggen

Bei der Anmeldung bei einem Dienstanbieter wird ein bestimmter Satz von persönlichen Daten verlangt. Wenn ein Benutzer dies selbständig erledigt, verlässt er sich auf eine Instanz, die nicht als eine Vertrauensinstanz definiert wird. Es werden Identifikationstokens wie Name und Password erstellt. Die Verantwortung liegt beim Benutzer, inwieweit persönliche Daten angegeben und die Stärke des Passworts gewählt werden [4]. Die Inkompetenz des Benutzers und des Dienstanbieters kann hier zum Risikofaktor werden. Somit stellt sich die Frage, ob das

Risiko der Inkompetenz mit Hilfe einer Komponente der Client-Architektur reduziert werden kann.

3.1.2. Sicherer Übertragungskanal

Bei Inanspruchnahme eines Dienstes überträgt der Benutzer seine persönliche Daten, Authentifizierungsanfragen und Dateien über ein Netzwerk. Es ist festzustellen, dass ein Benutzer keinen Einfluss darauf hat, wie der Dienstanbieter diesen Vorgang regelt. Somit beschränken sich die Sicherheitsmaßnahmen durch den Benutzer auf die Wahl des richtigen Dienstanbieters. Rechnergestützte clientseitige architektonische Maßnahme sind hier nicht anwendbar.

- Der Dienstanbieter CloudMe[1] bietet keinen sicheren Übertragungskanal weder während der Anmeldung noch bei Inanspruchnahme der Dienste [4]. Ohne das Eingehen auf weitere Sicherheitsmerkmale sollte diese Tatsache einen potenziellen Benutzer davon abhalten, den Dienst zu nutzen.
- Der Dienstanbieter CrashPlan[2] bietet keinen sicheren Übertragungskanal. Er setzt auf ein proprietäres Protokoll [4], was aber nicht für ausreichend sicher eingestuft wird.
- Der Dienstanbieter Dropbox[3] bietet einen mit TLS/SSL gesicherten Übertragungskanal[4].
- Der Dienstanbieter Mozy[4] bietet einen mit TLS/SSL gesicherten Übertragungskanal[4].
- Der Dienstanbieter TEAMDRIVE[5] bietet keinen sicheren Übertragungskanal. Er setzt auf ein proprietäres Protokoll[4], was nicht für ausreichend sicher eingestuft wird.
- Der Dienstanbieter Ubuntu One[6] bietet einen mit TLS/SSL gesicherten Übertragungskanal[4].
- Der Dienstanbieter Wuala[7] bietet keinen sicheren Übertragungskanal. Er setzt auf ein proprietäres Protokoll [4], was nicht für ausreichend sicher eingestuft wird.

Zusammenfassung: Ein Übertragungskanal, der Authentizität, Geheimhaltung und Integrität gewährleistet, ist eine Voraussetzung für den sicheren Einsatz eines entfernten Speichers. Da clientseitige Sicherheitsmaßnahmen an dieser Stelle nicht anwendbar sind, ist bei der Wahl des Dienstanbieters darauf zu achten, dass ein sicherer Übertragungskanal zur Verfügung steht, der wie man sieht, nicht standardmäßig zum Einsatz kommt.

[1] http://www.cloudme.com/
[2] http://www.crashplan.com/
[3] https://www.dropbox.com/
[4] http://www.mozy.com
[5] http://www.teamdrive.com/
[6] https://one.ubuntu.com/
[7] http://www.teamdrive.com/

3.1.3. Sichere Datenaufbewahrung

Bei der Aufbewahrung der Daten auf einem entfernten Speicher muss sichergestellt werden, dass sie für unberechtigte Benutzer unerreichbar sind wie auf einem lokalen Speicher. Da die Kontrolle über den entfernten Speicher nicht bei dem Benutzer liegt, muss sich auf die Zusicherungen des Dienstanbieters verlassen werden. Es soll untersucht werden, inwieweit Maßnahmen zur sicheren Datenaufbewahrung bei den Dienstanbietern Standard sind, und was sie leisten.

- Der Dienstanbieter *CloudMe* bietet keine sichere Datenaufbewahrung[4], sodass beim Erlangen des Zugriffsrechts auf den Speicher die Daten in lesbarer Form zur Verfügung stehen.
- Der Dienstanbieter *CrashPlan* bietet eine sichere Datenaufbewahrung. Die Daten werden verschlüsselt abgelegt, und der Schlüssel kann durch ein Passwort geschützt bei dem Anbieter hinterlegt oder durch den Benutzer selbst verwaltet werden[4].
- Der Dienstanbieter *Dropbox* verschlüsselt die Daten serverseitig [4]. Das heißt, die Daten sind gegen einen anbieterexternen Angriff geschützt, allerdings hat der Anbieter selbst die Möglichkeit die Daten zu lesen, was als nicht ausreichend sicher eingestuft wird.
- Der Dienstanbieter *Mozy* bietet eine clientseitige Verschlüsselung auf zwei Arten an[4]. Der Benutzer kann einen vom Anbieter vorgeschlagenen Schlüssel einsetzen, was bedeutet, der Anbieter hat Zugriff auf die Daten. Die Alternative ist, der Benutzer wählt selbst einen Schlüssel, was eine sichere Dateilagerung ermöglicht.
- Der Dienstanbieter *TEAMDRIVE* bietet eine clientseitige Verschlüsselung an. Es werden jedoch nicht einzelne Dateien Verschlüsselt, sondern *Spaces*, denen wie Ordnern Dateien hinzugefügt werden können. Alle Dateien innerhalb eines Space haben den gleichen Schlüssel. Die Schlüssel werden zu TEAMDRIVE nicht übertragen, sodass der Anbieter keinen Zugriff auf die Daten hat[4].
- Der Dienstanbieter *Ubuntu One* bietet keine sichere Datenaufbewahrung[4], sodass beim Erlangen des Zugriffsrechts auf den Speicher die Daten in lesbarer Form zur Verfügung stehen.
- Der Dienstanbieter *Wuala* bietet eine sichere Datenaufbewahrung. Die Daten werden verschlüsselt abgelegt, allerdings wird ein kryptographisches Dateisystem verwendet auf Basis von *Cryptree*, dass bestimmte Angriffsarten begünstigt, bzw. ermöglicht[4].

Zusammenfassung: eine im Kontext dieser Arbeit genügend sichere Datenaufbewahrung, sodass niemand außer des Dateneigentümers Zugriff auf den Inhalt hat, bieten nur die Minderheit der Anbieter. Dabei müssen allerdings die Möglichkeiten der Dateifreigabe 3.1.4 betrachtet werden. Hier wird der Einsatz einer rechnergestützte clientseitigen Architektur zur Erfüllung des Sicherheitsmerkmals *Sichere Datenaufbewahrung* für möglich gehalten.

3.1.4. Sichere Dateifreigabe

Es gibt die Möglichkeit die Dateifreigabe über den Dienstanbieter zu gestalten, und zwar in folgenden Arten:

- Registrierte Benutzer des konkreten Anbieters, die durch den Dateneigentümer zum Zugriff zugelassen wurden, können auf die Daten zugreifen.
- Eine Gruppe von vordefinierten Benutzern kann auf die Daten zugreifen, unabhängig davon, ob sie beim Anbieter registriert sind.
- Alle Benutzer können auf die Daten zugreifen (Die Daten sind frei zugänglich).

Im Kontext dieser Arbeit wird folgende Dateifreigabe für sinnvoll gehalten: Es dürfen nur die Benutzer eines konkreten Anbieters auf die freigegebenen Dateien zugreifen. Das heißt, es wird erwartet, dass jeder Benutzer, der an der Dateifreigabe beteiligt wird, bei dem Dienstanbieter registriert und autorisiert sein muss. Das erfüllt die Anforderung, Schreib- und Löschrechte dem Fremdnutzer zu entziehen, die nur mit Hilfe des Anbieters durchgesetzt werden kann. Und nur die durch den Dateneigentümer zum Zugriff zugelassenen Benutzer können auf die Daten zugreifen. Es soll festgestellt werden, ob die Dateifreigabe der ausgewählten Anbieter entsprechend den Anforderungen zufriedenstellend ist.

- Der Dienstanbieter CloudMe soll Funktionalität der Dateifreigabe für registrierte Benutzer nicht in ausreichendem Maße zur Verfügung gestellt haben[4].
- Der Dienstanbieter CrashPlan bietet keine Dateifreigabe an[4], da das Problem der Schlüsselverteilung nicht gelöst wird. Die Daten sind mit einem einzigen Schlüssel verschlüsselt, den man dem Dienstanbieter oder dem anderen Benutzer persönlich übergeben müsste. Bei der ersten Alternative macht die Datenverschlüsselung keinen Sinn, da sie vom Anbieter gelesen werden können, und bei der zweiten Alternative führt die Zurücknahme der Dateifreigabe zur einer kompletten Neuverschlüsselung des Datenbestandes. Dieses wird für keine praxistaugliche Lösung gehalten.

- Der Dienstanbieter Dropbox bietet die Dateifreigabe für registrierte Benutzer an[4]. Die Schreib- und Löschzugriffe sind somit durch nur den Dateneigentümer kontrollierbar, was als erwartete Anforderung angesehen wird. Einzelne Kontakte sind wählbar.
- Der Dienstanbieter Mozy bietet keine Dateifreigabe an[4], da das Problem der Schlüsselverteilung bei der clientseitigen Schlüsselwahl nicht gelöst wird.
- Der Dienstanbieter TEAMDRIVE bietet die Dateifreigabe für registrierte Benutzer und ausgewählte Kontakte an. Die Dateifreigabe wird mit kryptographischen Mitteln gelöst, indem jeder Benutzer über einen öffentlichen und einen privaten Schlüssel 2.4.1.2 verfügt. Die Freigabe basiert auf der Weitergabe des *AES*-Schlüssels eines Space, sodass alle Dateien innerhalb des Space lesbar sind. Die Zurücknahme der Freigabe ist durch das Einschränken der Zugriffsrechte innerhalb der proprietären Client-Anwendung gelöst. Dies bedeutet, der von der Dateifreigabe ausgeschlossener Benutzer hat immer noch den Schlüssel zu den Dateien, und er kann theoretische alle Dateien, die neu hinzukommen, lesen[4]. Die Lösung ist nicht zufriedenstellend.
- Der Dienstanbieter Ubuntu One bietet die Dateifreigabe für registrierte Benutzer an[4], womit die Schreib- und Löschrechte durchgesetzt werden können, und einzelne Kontakte wählbar sind.
- Der Dienstanbieter Wuala bietet die Dateifreigabe für registrierte Benutzer an und einzelne Kontakte sind wählbar. Die Dateifreigabe wird mit kryptographischen Mitteln gelöst, indem jeder Benutzer über einen öffentlichen und einen privaten Schlüssel 2.4.1.2 verfügt. Der *AES*-Schlüssel der Datei, der freigegeben werden soll, wird mit dem öffentlichen Schlüssel des fremden Benutzers chiffriert. Die Rücknahme der Freigabe wird ebenfalls durch kryptographische Mittel so gelöst, dass die *AES*-Schlüssel der neuen Dateien nicht mit dem öffentlichen Schlüssel des fremden Benutzers chiffriert werden[4]. Die Lösung wird als zufriedenstellend angesehen.

Es gibt sowohl Anbieter, die eine zufriedenstellende Lösung für die Dateifreigabe anbieten, als auch Anbieter mit mangelhafter oder gar keiner Lösung.

Zusammenfassung

In Bezug auf die Gesamtheit der Sicherheitsanforderungen - Einloggen/Anmelden, Übertragungskanal, Datenaufbewahrung, Dateifreigabe - ist feststellbar, dass bei einer im Sinne dieser Arbeit richtigen Umsetzung bestimmter Anforderungen für andere Anforderungen Nachteile

entstehen. Bei Inanspruchnahme eines Dienstes soll vom Benutzer die Erfüllung der Sicherheitsanforderungen überprüft werden. Auf bestimme Sicherheitsmaßnahmen hat der Benutzer jedoch keinen Einfluss, weil das Zusicherungen des Anbieters sind, auf die sich der Anbieter unter bestimmten Umständen nicht halten muss oder darf 2.5. Eine technische Prüfung der Sicherheitssysteme durch den Benutzer ist kaum machbar. Insgesamt entstehen Risiken. Ein Einsatz einer rechnergestützten clientseitigen Sicherheitsarchitektur zur Reduzierung der Risiken wird für möglich und sinnvoll gehalten.

4. Umsetzung

Es wird eine Idee für eine Client-Server Anwendung entwickelt, bei der ein entfernter Speicher als bereits fertiger Server zur Verfügung steht und eine Zusammensetzung bestimmter Komponenten als Client gesucht wird. Diese Komponenten sollen im Kontext dieser Arbeit eine clientseitige Sicherheitsarchitektur verwirklichen.

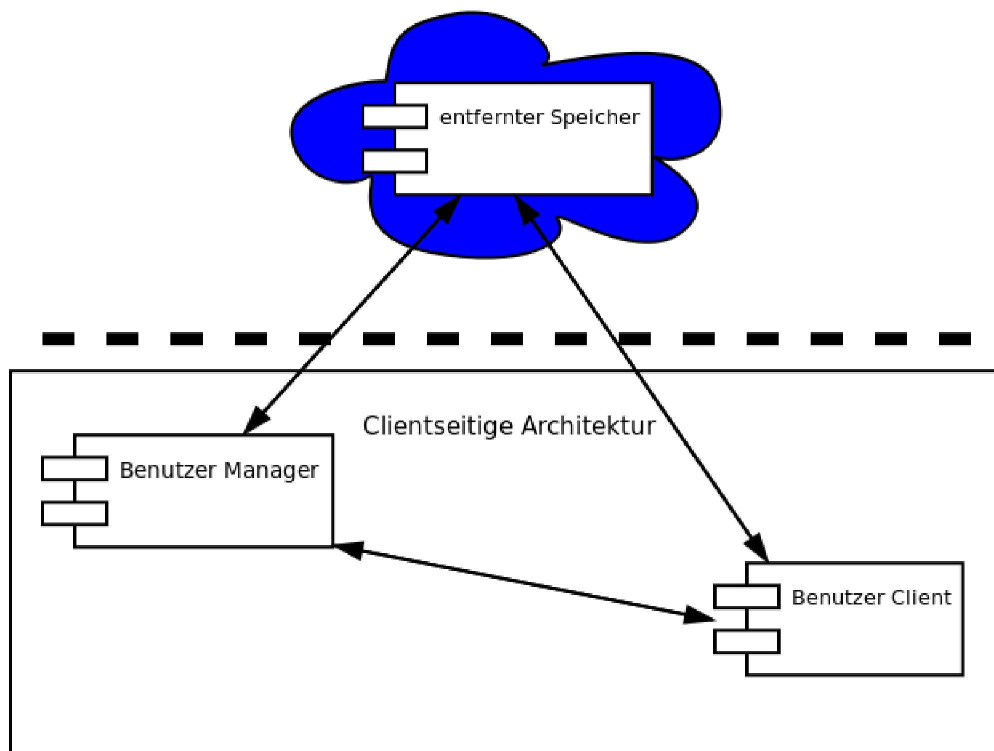


Abbildung 4.1.: Server-Speicher und Client-Architektur

Der entfernte Speicher bzw. sein Anbieter muss sicherheitsrelevante Merkmale erfüllen, die im Kapitel 2 Grundlagen definiert wurden. Es wurde im Kapitel 3 Analyse festgestellt, dass keiner der Anbieter diese Merkmale in hier für sinnvoll gehaltenem Maße erfüllt, sodass eine

clientseitige Sicherheitsarchitektur aufgebaut werden soll. Es sollen kryptographische Prozesse eingesetzt werden, die für den Benutzer transparent bleiben.

Von einem Anbieter wird verlangt:

- Der Übertragungskanal zwischen dem Server und dem Client soll gesichert sein.
- Um unautorisiertes Schreiben und Löschen zu verhindert, stellt der Anbieter entsprechende Benutzer-Zugriffsrechte zur Verfügung.
- Dateifreigabe (Lesezugriff) für alle registrierten Benutzer.

Die clientseitige Architektur besteht auf der obersten Abstraktionsebene aus zwei Komponenten. Der Benutzer-Manager ist eine Vertrauensinstanz. Sie stellt eine Benutzer-Datenbank zur Verfügung und sorgt für die Anonymität der Benutzer beim entfernten Speicher, indem eine *Abbildung* von realen persönlichen Daten auf Benutzer-Manager-Daten eines Benutzers stattfindet.

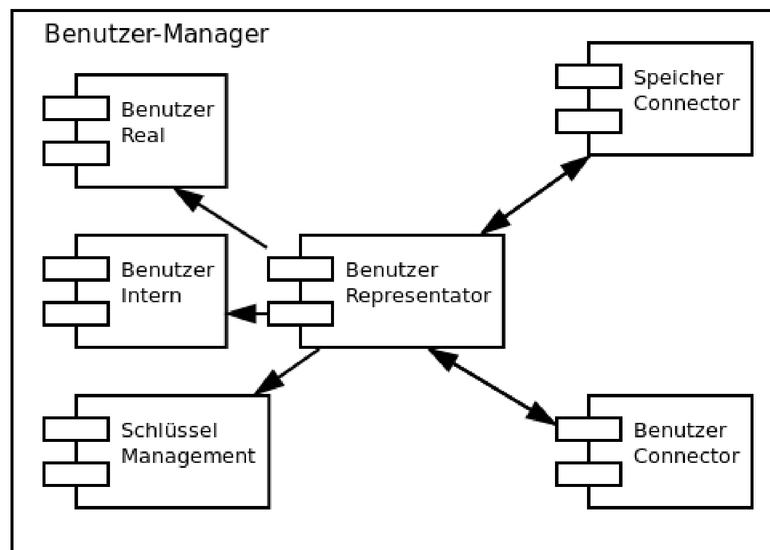


Abbildung 4.2.: Benutzer-Manager-Komponenten

Der Speicher-Connector ist für die Netzwerk-Kommunikation mit dem entfernten Speicher zuständig.

Der Benutzer-Connector ist für die Kommunikation mit dem Client eines Benutzers zuständig.

Der Benutzer-Representator stellt mit Hilfe einer Abbildung zwischen realen und internen Benutzerdaten und mit Zuordnung zu einem Schlüssel einen Benutzer dar.

Die zweite Komponente - der Benutzer-Client - agiert auf dem Endgerät eines Benutzer, und ist ein Client sowohl zum Benutzer-Manager als auch zu dem entfernten Speicher.

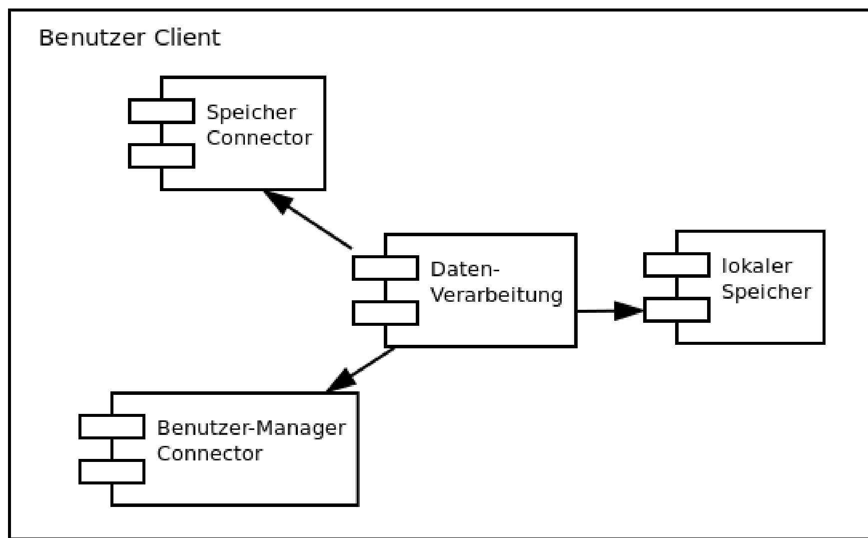


Abbildung 4.3.: Benutzer-Client-Komponenten

Der Speicher-Connector ist für die Netzwerk-Kommunikation mit dem entfernten Speicher zuständig.

Der Benutzer-Manager-Connector ist für die Kommunikation mit dem Benutzer-Manager zuständig.

Die Datenverarbeitung-Komponente ist für kryptographische Verarbeitung der Daten zuständig, die vom lokalen Speicher auf einen entfernten Speicher übertragen werden sollen bzw. andersrum.

4.1. Von Sicherheitsanforderungen zur Architektur

4.1.1. Sicheres Anmelden/Einloggen

Im Kapitel Analyse zum sicheren Anmelden/Einloggen 3.1.1 wurde festgestellt, dass eine mögliche Inkompetenz des Benutzers und des Dienstanbieters ein Risikofaktor ist. Dies kann dadurch reduziert werden, indem eine Vertrauensinstanz den Vorgang der Anmeldung übernimmt. Dadurch ist eine Identitätsgeheimhaltung der Benutzer gegenüber dem Dienstanbieter erreichbar und die Stärke der Identifikationstokens garantiert. Also eine Komponente auf der Client-Seite trägt die Verantwortung für die Identitäten bei der Kommunikation mit einem entfernten Speicher. Für die spätere Authentifizierung werden die vom Dienstanbieter erhaltenen Sicherheitstokens verwendet.

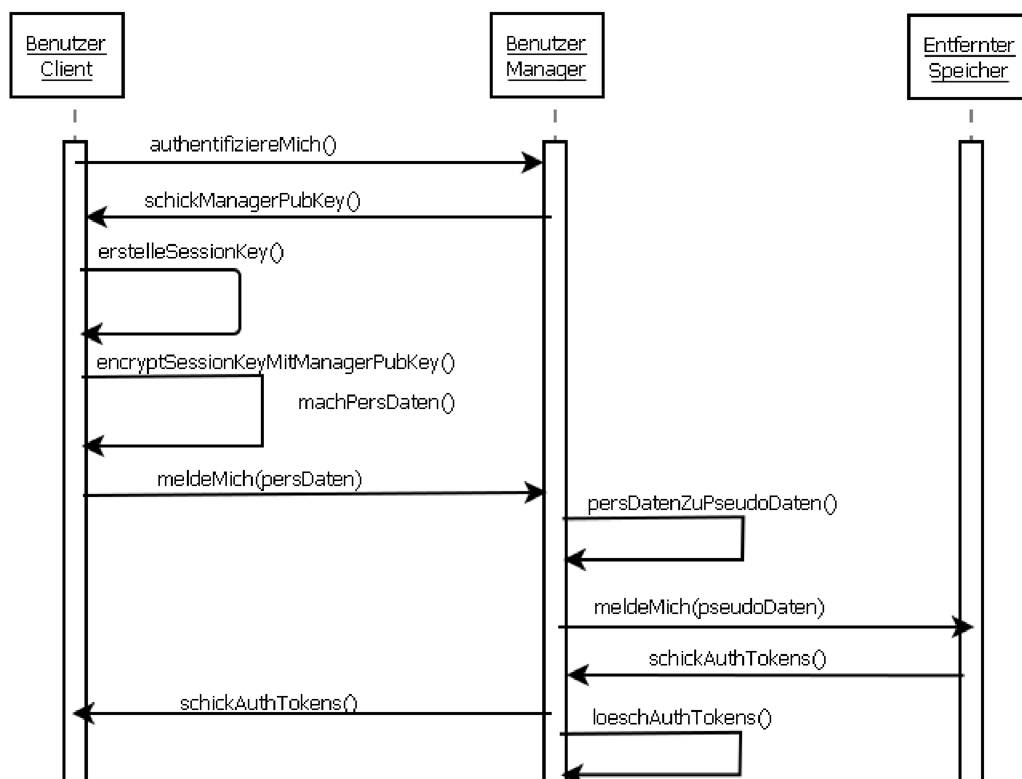


Abbildung 4.4.: Sequenzdiagramm: Anmelden

Anmerkung: es wird nicht darauf eingegangen, wie die Identität eines Benutzers bei der Anmeldung festgestellt werden kann. Eine Authentifikation kann mit einem passwortbasierten Verfahren oder durch lokale Speicherung eines Tokens erfolgen

Der Anmeldevorgang bei einem entfernten Speicher wird auf folgende Weise vorgenommen: Ein Benutzer plant ein Mitglied der Gruppe zu sein, die ihre Daten auf einem entfernten Speicher aufbewahrt. Sein Client erstellt persönliche Identifikationsdaten (inklusive eines privaten und öffentlichen Schlüssel) und schickt die für die Anmeldung am entfernten Speicher und für das clientseitige Benutzer-Management benötigten Daten zu dem Benutzer-Manager. (Der private Schlüssel wird nicht geschickt). Der Benutzer-Manager registriert den neuen Benutzer bei sich und erstellt für ihn die mit Original-Identifikationsdaten verknüpfte Pseudo-Identifikationsdaten. Mit den Pseudo-Identifikationsdaten wird der Benutzer am entfernten Speicher registriert. Als Antwort werden Sicherheitstokens zurückgeschickt, mit denen ein Einloggen am entfernten Speicher möglich ist. Die Sicherheitstokens werden an den Benutzer-Client weitergeleitet und bei dem Benutzer-Manager nicht aufbewahrt.
Anmerkung: Falls die Sicherheitstokens bei dem Benutzer-Manager nicht vernichtet werden, wird das als ein akzeptables Risiko eingestuft, weil der Benutzer-Manager als eine Vertrauensinstanz angesehen wird. Im Falle eines Angriffs wäre ein Vernichten der Daten, jedoch kein Lesen und kein unbemerktes Verändern möglich. Ein Backup der wichtigen Daten sollte immer für einen Notfall erstellt werden.

4.1.2. Sicherer Übertragungskanal

Ein sicherer Übertragungskanal bei der Kommunikation über ein offenes Netzwerk ist einzusetzen. Wie im Kapitel Analyse 3.1.2 festgestellt wurde, ist die Kommunikation zwischen einem entfernten Speicher und dem Benutzer-Client nur dann gesichert, wenn der Dienstanbieter dies ermöglicht. Daraus folgt, der Benutzer soll bei der Wahl des Dienstanbieter darauf achten.

Die Kommunikation zwischen dem Benutzer-Manager und dem Benutzer-Client findet ebenfalls über ein offenes Netzwerk statt. Da jedoch beides Komponenten der clientseitigen Sicherheitsarchitektur sind, ist das Einbringen eines sicheren Übertragungskanals zu realisieren.

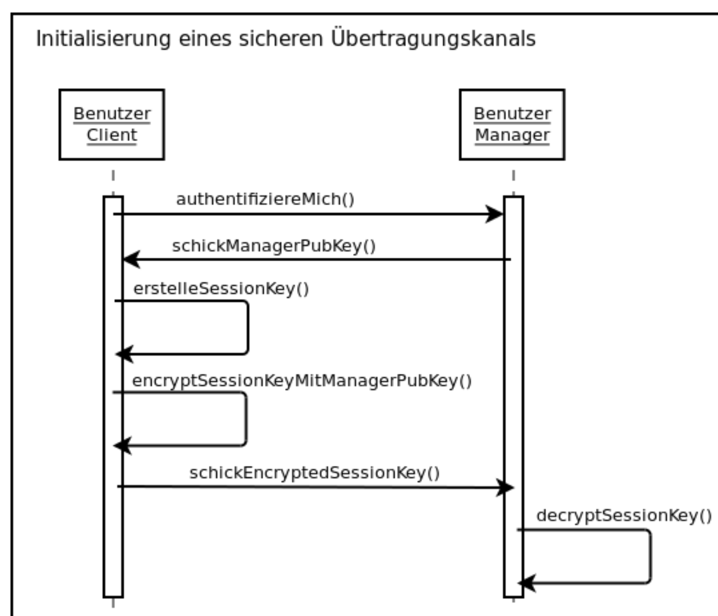


Abbildung 4.5.: Sequenzdiagramm: Initialisierung eines sicheren Übertragungskanals

Ein sicherer Übertragungskanal zwischen dem Benutzer-Manager und dem Benutzer-Client wird auf folgende Weise initialisiert: Wie schon in Sicheres Anmelden/Einloggen angemerkt, wird nicht darauf eingegangen, wie die Authentifikation zwischen dem Benutzer-Manager und dem Benutzer-Client abläuft. Nachdem die Authentifikation abgeschlossen ist, erhält der Benutzer-Client den öffentlichen Schlüssel des Benutzer-Managers. Daraufhin wird vom Benutzer-Client ein Schlüssel zu einem symmetrischen Verschlüsselungsverfahren erstellt, Session-Key genannt, mit dem öffentlichen Schlüssel des Benutzer-Managers chiffriert und das Chiffrat an den Benutzer-Manager übertragen. Es fand somit ein Schlüssel-Austausch statt, so

dass die Nachrichten zwischen den beiden Kommunikationspartners verschlüsselt übertragen werden können.

4.1.3. Sichere Datenaufbewahrung

Im Kapitel Analyse 3.1.3 zur sicheren Datenaufbewahrung wurde festgestellt, dass sich der Benutzer eines entfernten Speichers entweder auf die Zusicherungen des Anbieters verlassen muss oder mit Hilfe einer clientseitigen Architektur für die Sicherheit der Daten selbst sorgen kann. Der Aufbau der Sicherheitslösung auf der Client-Seite wird für die zu wählende Alternative gehalten. Die sichere Datenaufbewahrung wird mit Hilfe kryptographischer Primitive 2.4 erreicht. Die Ausgangssituation dabei ist, dass der Benutzer-Client bereits über ein Schlüssel-Paar zum Einsatz eines asymmetrischen Verschlüsselungsverfahrens verfügt. Dieses Schlüssel-Paar wurde vor dem Anmelden am Benutzer-Manager und dem entfernten Speicher erstellt.

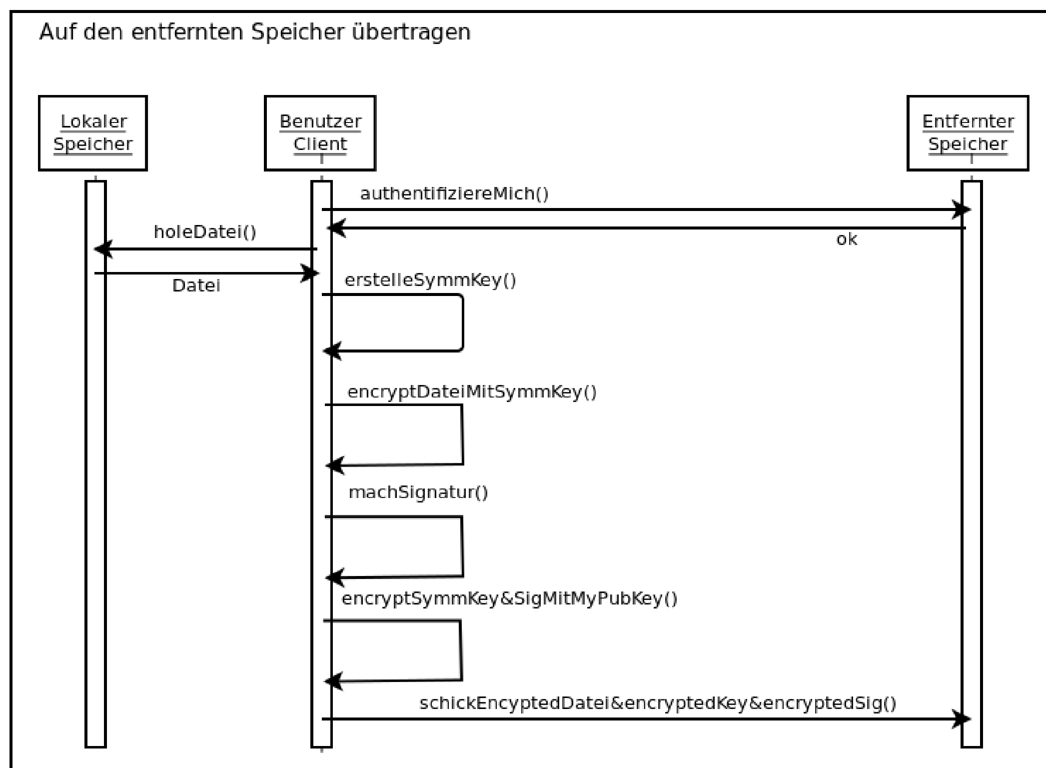


Abbildung 4.6.: Sequenzdiagramm: Datei auf den entfernten Speicher übertragen

Mit den bei der Anmeldung erhaltenen Sicherheitstokens authentifiziert sich der Benutzer-Client beim entfernten Speicher. Es wird eine Datei auf dem lokalen Speicher ausgewählt, die auf den entfernten Speicher übertragen werden soll. Es wird ein einmaliger Schlüssel für eine symmetrische Verschlüsselung erstellt und die Datei mit dem Schlüssel verschlüsselt. Von der verschlüsselten Datei (bzw. der unverschlüsselten Datei) wird eine Signatur erstellt, sodass ihre Herkunft überprüft werden kann. Dies kann unter anderem eine Dateimanipulation seitens Benutzer-Manager aufdecken, was aber als Risiko für unwahrscheinlich gehalten wird. Der symmetrische Schlüssel zu der Datei und die Signatur werden mit dem eigenen öffentlichen Schlüssel chiffriert und die drei Artefakte auf den entfernten Speicher übertragen. Die Sicherheitstokens zu der Authentifikation bei dem entfernten Speicher und der private Schlüssel können auf ein anderes Endgerät des Benutzers platziert werden, sodass ein Zugriff von mehreren Geräten möglich wird.

Eine Datei wird vom dem entfernten auf den lokalen Speicher auf folgende Weise übertragen:

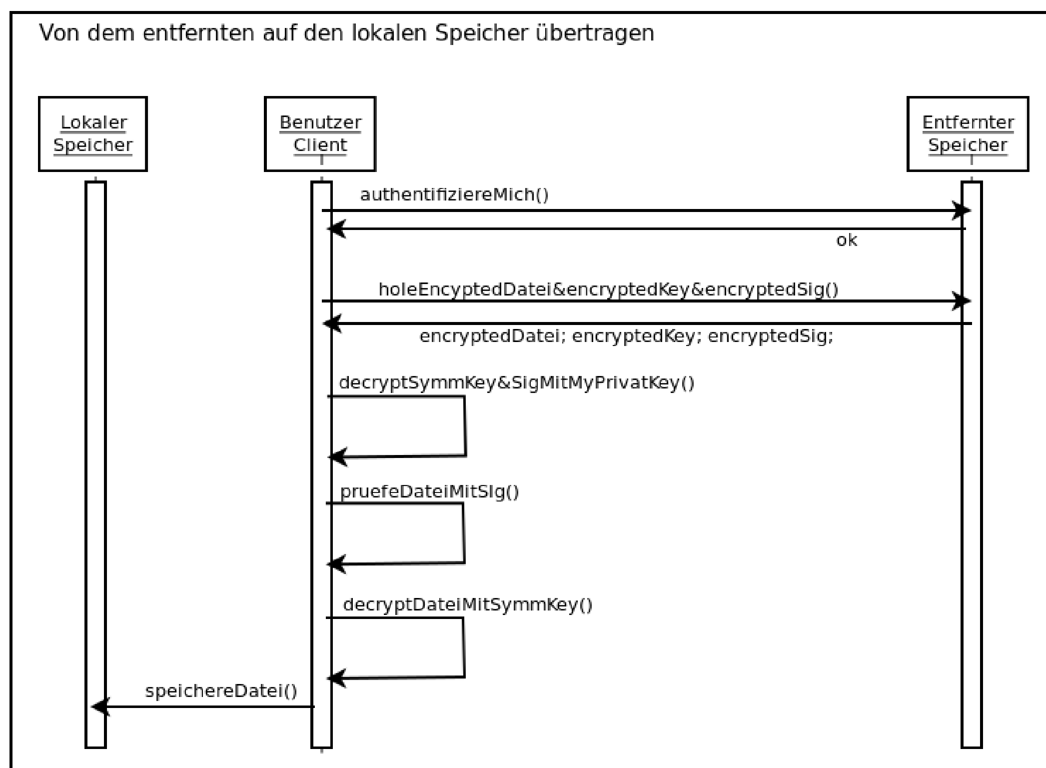


Abbildung 4.7.: Sequenzdiagramm: Datei vom entfernten auf den lokalen Speicher übertragen

Die Dateien liegen auf dem entfernten Speicher mit Hilfe der kryptographischen Primitive 2.4 gesichert. Um sie im Klartext auf den lokalen Speicher zu übertragen, werden nach dem Authentifizierungsvorgang der mit eigenem öffentlichen Schlüssel chiffrierter symmetrischer Schlüssel zu der Datei, die mit dem öffentlichen Schlüssel chiffrierte Signatur und die Datei selbst heruntergeladen. Mit dem eigenen privaten Schlüssel wird die Signatur und der symmetrische Schlüssel dechiffriert. Durch Abbildung der Datei auf einen Hash-Wert und einen Vergleich mit dem Signierten Hash-Wert, wird die Herkunft der Datei bestätigt. Da der symmetrische Schlüssel jetzt im Klartext vorliegt, wird durch seine Anwendung innerhalb eines entsprechenden symmetrischen Verschlüsselungsverfahrens die Datei entschlüsselt.

4.1.4. Sichere Dateifreigabe

Die Dateifreigabe wird zwar als eine abgegrenzte Anforderung an die Architektur in dieser Arbeit formuliert, doch sie setzt in starkem Maße auf die anderen Anforderungen. Die Dateifreigabe soll Unbefugten keine Information darüber liefern, wer mit wem kommuniziert - hier ist die Geheimhaltung der wahren Identitäten nötig. Da es dabei auch um den Transfer der Daten geht, ist ein sicherer Übertragungskanal 2.3.2 aufzubauen. Da es nicht erkennbar sein soll, welche Daten wem freigegeben werden, ist eine sichere Datenaufbewahrung 2.3.3 einzusetzen. Im Kapitel Analyse wurde festgestellt, dass bei der Dateifreigabe entweder stark auf die Zusicherungen der Anbieter in Bezug auf die Sicherheit verlassen werden muss, oder es wird keine Dateifreigabe angeboten, oder in Verbindung mit den anderen Anforderungen eine Dateifreigabe aus der Sicht dieser Arbeit als mangelhaft realisiert wird. Es soll eine alternative Lösung gesucht werden, die mit Hilfe einer clientseitigen Architektur die Nachteile der Standardlösungen verringert und die Gesamtsicherheit der Anwendung erhöht. Die Freigabe soll so erfolgen, dass sowohl ihre Gewährung als auch Rücknahme keine der in Analyse angesprochenen Nachteile bewirkt. Das heißt, das Problem der Schlüsselverteilung soll gelöst werden. Es darf bei der Rücknahme der Freigabe keine Neuverschlüsselung des Datenbestandes notwendig sein, der Schlüssel darf nicht an den Dienstanbieter ausgegeben werden. Dazu wird eine clientseitige Architektur, die mit Hilfe der kryptographischen Primitive 2.4 für die Sicherheit sorgt, für eine geeignete Lösung gehalten.

Die Freigabe wird auf folgende Weise gewährt:

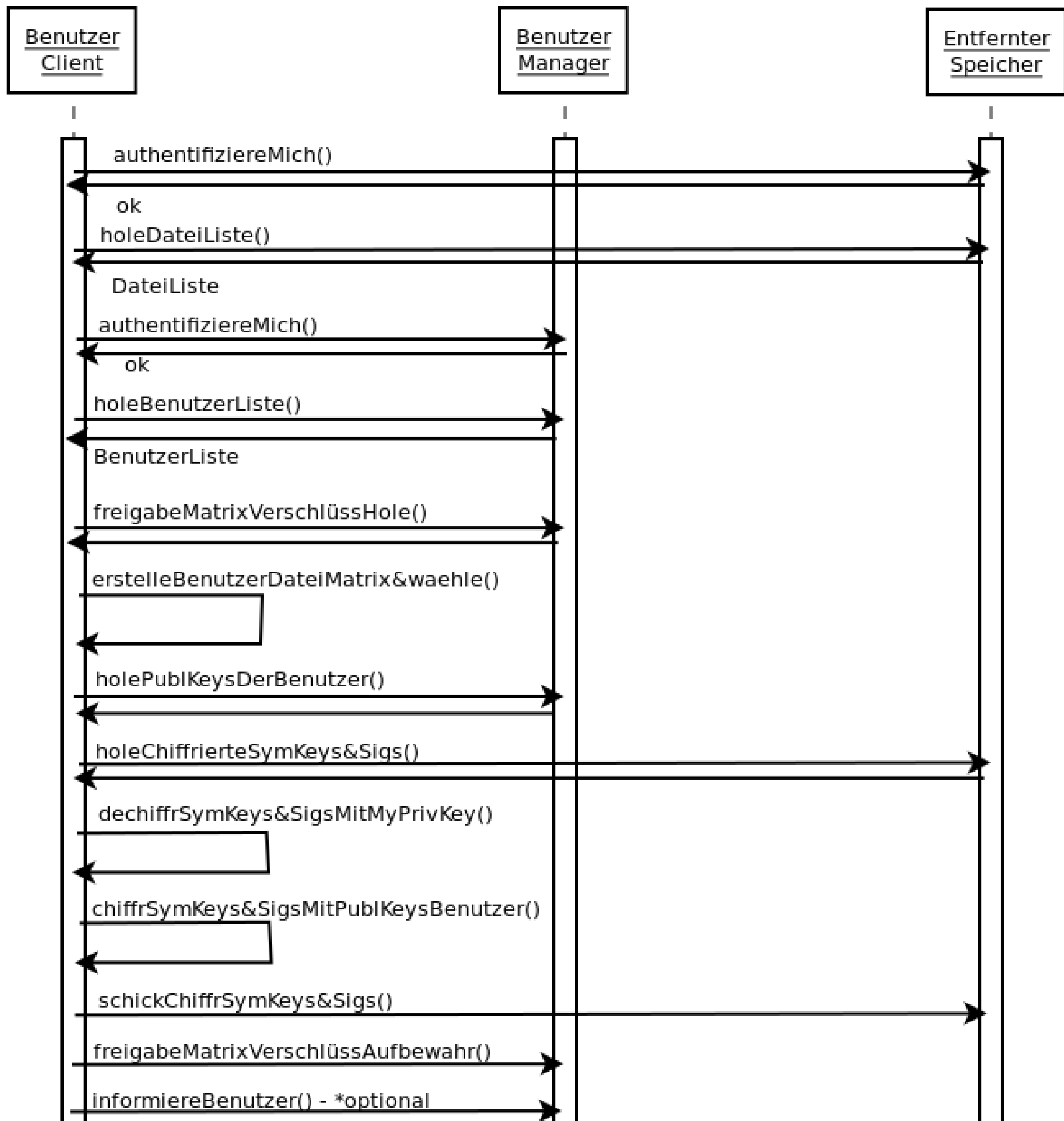


Abbildung 4.8.: Sequenzdiagramm: Dateifreigabe gewähren

Für Zugriffe auf den entfernten Speicher und den Benutzer-Manager muss sich der Benutzer-Client bei ihnen authentifizieren. Es wird die Liste der eigenen Dateien auf dem entfernten Speicher und die Liste der beim Benutzer-Manager angemeldeten Benutzer geholt. In Form

einer zweidimensionalen Matrix ist es möglich einzelne Dateien und Benutzer auszuwählen. Es wird eine bereits erstellte Freigabe-Matrix, die in verschlüsselter Form beim Benutzer-Manager aufbewahrt wird, geholt. Falls neue Dateien oder Benutzer dazukamen, wird die Matrix aktualisiert. Entsprechend der Wahl werden die öffentlichen Schlüssel der Benutzer vom Benutzer-Manager und die Datei-Schlüssel und die Datei-Signaturen, die mit dem eigenen öffentlichen Schlüssel chiffriert sind, vom entfernten Speicher auf den Client heruntergeladen. Die Datei-Schlüssel und Datei-Signaturen werden mit dem eigenen privaten Schlüssel dechiffriert und mit den öffentlichen Schlüsseln der ausgewählten Benutzer chiffriert. Sie werden zu dem entfernten Speicher geschickt. Die neue Freigabematrix wird, mit dem eigenen öffentlichen Schlüssel chiffriert, zu dem Benutzer-Manager übertragen. Optional können die Benutzer mit Hilfe des Benutzer-Managers darüber informiert werden, da der Benutzer-Manager die realen Identitäten verwaltet.

Zurücknahme der Freigabe findet auf diese Weise statt:

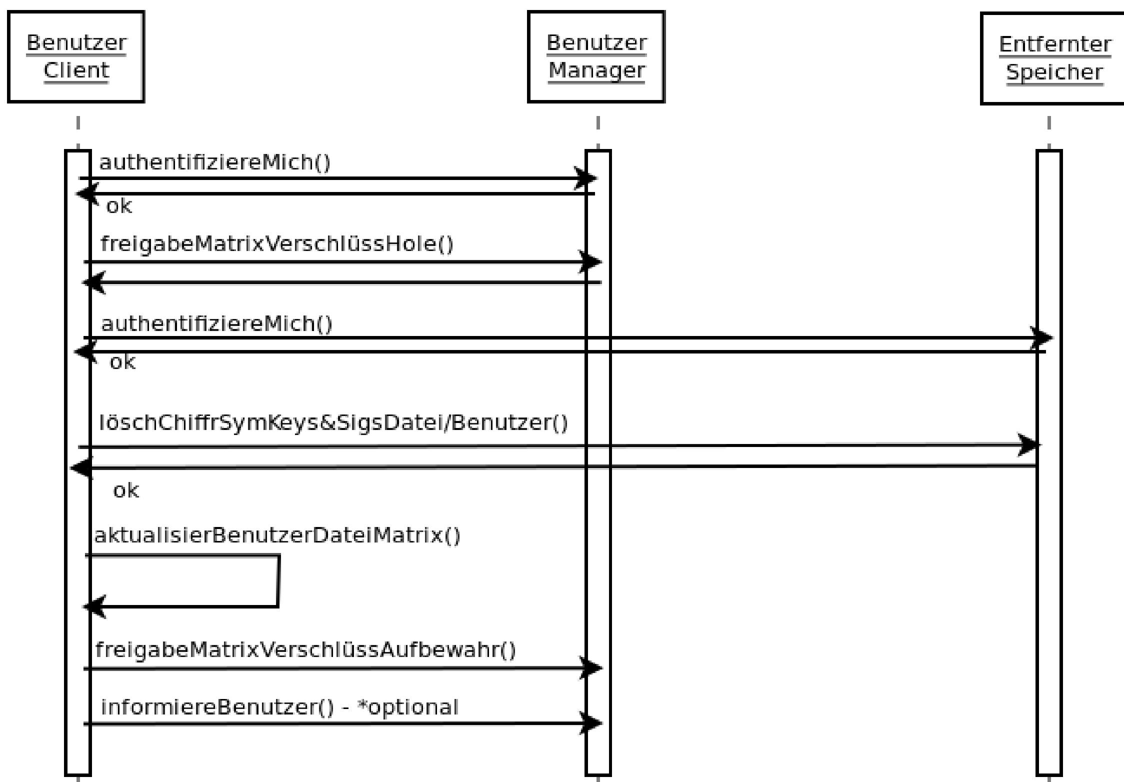


Abbildung 4.9.: Sequenzdiagramm: Dateifreigabe zurücknehmen

Für den Zugriff auf den Benutzer-Manager und den entfernten Speicher muss sich der Benutzer-Client bei den beiden authentifizieren. Es wird vom Benutzer-Manager die Freigabe-Matrix geholt, mit der die Freigabe zurückgenommen werden kann. Entsprechend der Wahl werden pro Benutzer die Datei-Schlüssel, die mit dem öffentlichen Schlüssel des Benutzers chiffriert sind, vom entfernten Speicher gelöscht. Die Freigabe-Matrix wird aktualisiert und bei dem Benutzer-Manager verschlüsselt aufbewahrt.

Zusammenfassung

Die Verschlüsselung wird auf der Dateiebene durchgeführt. Die Methode soll flexibel sein. Jede Datei wird einzeln verschlüsselt und auf den entfernten Speicher übertragen. Bei Änderung einer Datei muss nur diese eine Datei neu verschlüsselt und übertragen werden. Die Dateifreigabe bei dieser Methode erfordert einen Schlüssel-Management [4]. Der Schlüssel-Management muss unabhängig vom Anbieter des entfernten Speichers aufgebaut werden.
Die Dateifreigabe wird mit Hilfe der kryptographischen Prozesse 2.4 gelöst.
Der Entzug der Zugriffsrechte auf die Dateien für den eingeladenen Benutzer wird durch *lazy revocation*[1] [3] gewährleistet.
Die Verwaltung des privaten Schlüssels auf den Endgeräten eines einzelnen Benutzers soll möglich sein. Wenn der private Schlüssel auf einem der Geräte erzeugt wurde, soll es möglich sein auch auf einem anderen Gerät den Schlüssel zu erhalten, da sonst der Zugriff auf eigene Daten, die auf einem entfernten Speicher liegen, ausgeschlossen ist.
Die Sicherheit während der Übertragung ist einzusetzen. Dazu steht das SSL/TLS-Verfahren zur Verfügung. Die Übertragung wird sowohl zu dem entfernten Speicher (Sicherheitsanforderung an den Anbieter) als zu dem Benutzer-Management (Sicherheitsanforderung an die clientseitige Architektur) verschlüsselt.

[1] es findet keine Neuverschlüsselung statt, da angenommen wird, dass die Daten bereits entnommen wurden

5. Realisierung

An dieser Stelle soll eine prototypische Realisierung der in 4 Umsetzung entwickelten Sicherheitsarchitektur in Form eines Softwaremoduls erfolgen. Das heißt, es werden nicht alle Elemente der Architektur realisiert, sodass ähnliche Bausteine und Abläufe nicht wiederholt vorkommen. Es werden als Zielplattform das Android-Betriebssystem und als Anbieter eines entfernten Speicher Dropbox Inc.©gewählt. Es gilt eine Client-Server Architektur zu implementieren. Dabei wird auf zwei Ebenen abstrahiert. Die erste Ebene der Client-Server Architektur stellen der entfernte Speicher - Dropbox als Server, und die lokale Anwendung - Benutzer-Client und Benutzer-Manager gemeinsam als Client dar. Zur der zweiten Ebene gehören Benutzer-Manager als Server und Benutzer-Client als Client. Der Titel der Arbeit bezieht sich auf die Abstraktion der ersten Ebene, sodass der Benutzer-Manager und der Benutzer-Client zusammen als ein Client zu dem Dropbox-Dienst agieren.

5.1. Anwendungsfälle

In Abschnitt Anwendungsfälle eines entfernten Speichers wurden Anwendungsfälle angesprochen, von deren vollständiger Definition abstrahiert wird, sodass bei der konkreten Realisierung folgende Anwendungsfälle implementiert werden:

Upload
: Eine Datei auf dem lokalen Gerät wird ausgewählt und als eine Kopie auf den entfernten Speicher übertragen.

Download
: Eine Datei auf dem entfernten Speicher wird ausgewählt und ihre Kopie auf das lokale Gerät übertragen.

Sharing (Dateifreigabe)
: Der Benutzer kann aus einer Benutzer-Datenbank einen Mitglied auswählen und ihn somit zu seiner Liste der vertrauten Benutzer hinzufügen. Durch einen kryptographischen Prozess 2.4 erhalten die vertrauten Benutzer den Zugriff auf alle Daten des Eigentümers.

Arten der Dateifreigabe

Datei-Benutzer-Zuordnung
Es besteht die Möglichkeit eine Datei freizugeben. Damit haben alle Benutzer Zugriff auf diese eine Datei.

Benutzer-Datei-Zuordnung
Es wird ein Benutzer ausgewählt. Er erhält Zugriff auf alle Dateien des Eigentümers. (Diese Art wird implementiert.)

Kombination
Die beiden Arten können so gemeinsam eingesetzt werden, dass nur bestimmte Dateien für jeweils einen Benutzer freigegeben werden. Im Kapitel 4 Umsetzung wird dies als *Freigabe-Matrix* bezeichnet.

5.2. Komponenten

Die Realisierung der Software sieht eine Implementierung von zwei Komponenten vor:

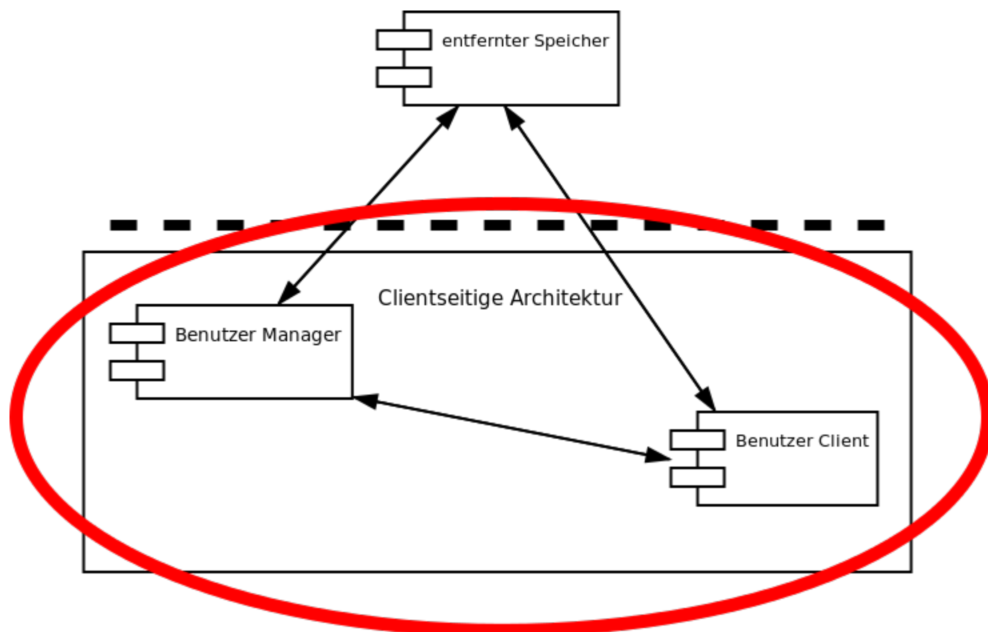


Abbildung 5.1.: Zu implementierende Komponenten

5.2.1. Benutzer-Manager

Der Benutzer-Manager ist ein Teil der clientseitigen Sicherheitsarchitektur, der als Ergebnis der Analyse und der Umsetzung entstanden ist.

Der **BAclientManager** übernimmt die Aufgabe einer Vertrauensinstanz, die die Schlüsselverwaltung und die Benutzerverwaltung steuert und wird nach dem Server-Prinzip der Client-Server-Architektur implementiert.

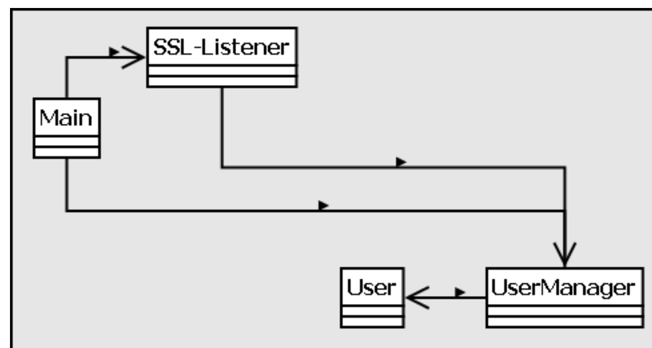


Abbildung 5.2.: Klassendiagramm: BAclientManager

Das Paket *de.sembo.manager* enthält folgende Quellcode-Dateien:

5.2.1.1. SSL-Listener

Der SSL-Listener wartet auf einem vorgegeben Port auf ankommende Verbindungsanfragen und leitet die Kommunikation an einen neuen Thread weiter. Es wird das TCP/IP - Protokoll eingesetzt.

```
1 server = new ServerSocket(port);
2   while (true) {
3      System.out.println("Listening...");
4      client = server.accept();
5      /* start a new thread to handle this client */
6      Thread t = new Thread(new ClientThread(client));
7      t.start();
8   }
```

Die innere Klasse

```
class ClientThread implements Runnable {
}
```

regelt die Kommunikation mit einem anfragenden Client. Es wird ein eigenes Protokoll verwendet, das aber nicht zu einer Erhöhung der Sicherheit führen soll. Dies wäre eine Verletzung des *Kerckhoffs' Prinzips*[1] [6]. Damit die Sicherheit des Übertragungskanals garantiert ist, soll das SSL - Protokoll verwendet werden. Die konkrete Implementierung wird an diese Stelle nicht vorgenommen, da die Schritte denen in der Implementierung von BAclient ähnlich sind. Trotzdem soll aufgezeigt werden, wie der Ablauf ist:

Es wird angenommen, ein Client, der mit BAclientManager kommunizieren will, hat ein Authentifizierungsvorgang bei dem BAclientManager vorgenommen. Dadurch erhält der Client vom BAclientManager dessen digitales Zertifikat. Zur Initialisierung der Kommunikation erstellt der Client einen Sitzungsschlüssel auf Basis eines symmetrischen Verschlüsselungsverfahrens und chiffriert ihn mit dem öffentlichen Schlüssel des BAclientManagers, der mit dem erhaltenen Zertifikat zur Verfügung steht. Es wird *encodedAesKey* von dem BAclientManager empfangen:

```
dis.read(encodedAesKey, 0, size);
Key decodedAesKey = decodeWith_RSA(encodedAesKey);
encodeWith_AES(decodedAesKey, answer1);
```

Der BAclientManager dechiffriert den Sitzungsschlüssel zu *decodedAesKey*. Beide Kommunikationspartner haben einen geheimen Schlüssel ausgetauscht, mit dem ihre Nachrichten verschlüsselt werden.

Zur Kommunikation zwischen den BAclientManager und BAclient wurde folgendes Protokoll implementiert:

addUserToDB
soll eine Anfrage zum Hinzufügen des anfragenden Benutzers zur Datenbank sein.

getUserList
liefert die Liste der in der Datenbank registrierten Benutzer.

addOrRemoveToMySharedUsers
fügt bzw. entfernt einen Benutzer zu bzw. aus der Liste der Benutzer, denen der Anfragende Dateien-Zugriff im Sinne von Sharing erlaubt. Es funktioniert wie ein Trigger[2].

[1] Die Sicherheit eines Verschlüsselungsverfahrens soll auf der Geheimhaltung des Schlüssels und nicht auf der Geheimhaltung des Algorithmus/Protokolls beruhen
[2] Umschalter zwsichen zwei Zuständen

getPublicKeyFor
liefert dem Anfragenden den öffentlichen Schlüssel eines bestimmten Benutzers.

getMySharedUsers
liefert die Liste der Benutzer, mit denen die Dateifreigabe des Anfragenden stattfindet.

Jede Anfrage verursacht den Aufbau einer TCP Verbindung und das Rücksenden einer Antwort, nach der die Verbindung geschlossen wird, sodass keine dauerhafte Verbindung für mehrere Anfragen aufrechterhalten werden muss.

5.2.1.2. UserManager

Der UserManager hat zur Aufgabe die Benutzerverwaltung und die Schlüsselverwaltung. Er wird vom *ClientThread* aufgerufen. Die Datenbank wird als Dateiliste organisiert. Für jeden Benutzer werden zwei Dateien erstellt.

“name„.publicKey
bedeutet, der Benutzer mit dem Namen “name„ ist in der Datenbank registriert. Die Datei enthält den öffentlichen Schlüssel des Benutzers “name„.

“name„.users
enthält die Namen der Benutzer, mit denen der Benutzer “name„ Dateifreigabe unterhält.

Die Klasse UserManager implementiert folgende Methoden:

```
1 public String addUserToDB(String ID, byte[] data) {
2 }
```

Die Parameter der Methode sind der Name und der öffentliche Schlüssel des Benutzers, der vom Client erzeugt und zur Datenbank geschickt wird.

```
1 public String getUserList(String myID) {
2 }
```

Der Parameter der Methode ist der Benutzername, der es bei der Suche nach den registrierten Benutzern erlaubt, den anfragenden Client auszuschließen.

```
1 public String getMySharedUsers(String myID) {
2 }
```

Der Parameter der Methode ist der Benutzername, dessen Liste der zur Dateifrage zugelassenen Benutzer gesucht wird.

```java
public String addOrRemoveToMySharedUsers(String myID, String guestID) {
}
```

Die Parameter der Methode sind der Name des anfragenden Benutzers und der Name des Benutzers, für den die Dateifrage stattfinden darf, bzw. dem die Zugriffsrechte wieder entzogen werden sollen.

```java
public byte[] getPublicKeyFor(String userID){
}
```

Der Parameter der Methode ist der Name des anfragenden Benutzers. Die Rückgabe ist die Referenz auf den eigenen öffentlichen Schlüssel.

5.2.1.3. User

Die Klasse User repräsentiert den Benutzer mit den Eigenschaften *UserID* und *publicKey*.

5.2.1.4. Main

Die Klasse Main startet den BAclientManager.

5.2.2. Benutzer-Client

Der Benutzer-Client ist ein Teil der clientseitigen Sicherheitsarchitektur, der als Ergebnis der Analyse und der Umsetzung entstanden ist.

Der **BAclient** wird auf dem Endgerät eines Benutzers installiert und kommuniziert als Client im Sinne einer Client-Server-Architektur 2.1 sowohl mit dem Benutzer-Manager 5.2.1 als auch dem entfernten Speicher (Anbieter: Dropbox Inc.©). Als Kommunikationsschnittstelle mit dem entfernten Speicher wird die vom Anbieter zur Verfügung gestellte API[3] verwendet.

*Für den Start wird eine Text-Datei **tcp.txt** benötigt, die in der ersten Zeile die Angabe der IP-Adresse der Form **ipAddress:141.22.88.81** und in der zweiten Zeile die Angabe des Ports **ipPort:32000** des Rechners mit laufendem BAclientManager enthält. Die Datei wird auf dem Android-Gerät im Ordner **sdcard/BAclient/** platziert.*

[3] application programming interface

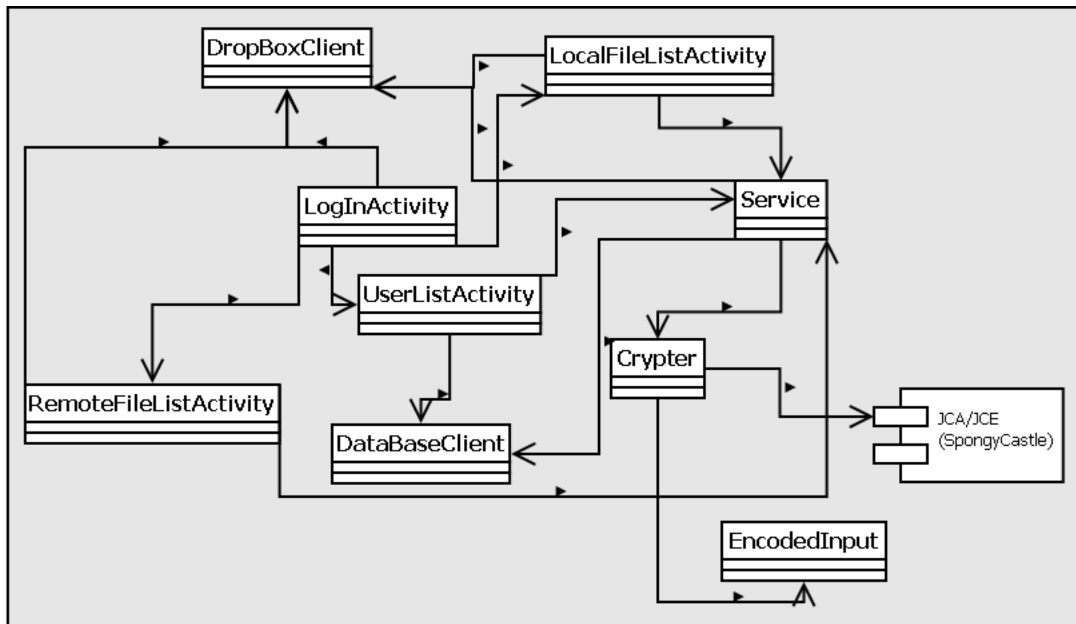


Abbildung 5.3.: Klassendiagramm: BAclient

Das Paket *de.sembo* enthält folgende Quellcode-Dateien:

5.2.2.1. DataBaseClient

Der BAclient agiert als Client zum BAclientManager, und die Klasse DataBaseClient ist für die TCP/IP-Kommunikation verantwortlich. Es werden die zu den in der Klasse UserManager vorhandenen Methoden entsprechenden clientseitigen Methoden implementiert. Hier folgt nur eine Auflistung, da die Funktionsweise in 5.2.1.2 nachgelesen werden kann:

```
static public ArrayList<String> getUserList()
static public ArrayList<String> getSharedUserList()
static public String addOrRemoveToMySharedUsers(String guestID)
static public byte[] getPublicKeyFor(String guestID)
static public String addUserToDB(String ID, byte[] publicKey)
```

5.2.2.2. DropBoxClient

Die Klasse ist auf Basis der von Dropbox Inc.©zur Verfügung gestellten API[4] für die Kommunikation mit dem entfernten Speicher verantwortlich.

Es werden folgende Methoden implementiert:

- ```
  static public ArrayList<String> getRemoteListing() {
  }
  ```

  Die parameterlose Methode greift auf den entfernten Speicher zu, erfragt die Liste der vorhandenen Dateien und liefert ihre Namen als ArrayList von Strings zurück.

- ```
  static public void uploadInputStream(String fileName, InputStream is,
      long size) {
  }
  ```

  Die Methode hat als Parameter den Namen der Datei, unter dem auf dem entfernten Speicher abgelegt wird, einen InputStream, der von der lokalen Datei liest, und die Größe der lokalen Datei.

- ```
  static public byte[] downloadByteArray(String fileName) {
  }
  ```

  Die Methode greift auf die Datei mit dem als Parameter vorgegebenen Namen auf dem entfernten Speicher zu und überträgt sie als Byte-Array zur weiteren Verarbeitung in den lokalen Speicher.

- ```
  static public boolean downloadOutputStream(String fileName, OutputStream
      os) {
  }
  ```

  Die Methode greift auf die Datei mit dem als Parameter vorgegebenen Namen auf dem entfernten Speicher zu und leitet die Daten in den lokalen OutputStream, der ebenfalls in der Parameterliste auftritt und mit einer lokalen Datei verbunden ist.

- ```
  static public void uploadByteArray(byte[] bytearray, String fileName) {
  }
  ```

  Die Methode überträgt einen Byte-Array auf den entfernten Speicher als Datei mit dem Namen “fileName„.

[4] application programming interface
  ```

```
static public void deleteFile(String fileName){
}
```

Die Methode löscht die Datei mit dem vorgegebenen Namen “fileName„ auf dem entfernten Speicher.

5.2.2.3. LocalFileListActivity

Die Klasse ist eine Android-Activity Klasse 2.7.1.1, die eine Liste von Dateien auf dem lokalen Gerät darstellen soll. Dabei findet ein Zugriff auf DropBoxClient 5.2.2.2 statt, der eine Liste von Dateien auf dem entfernten Speicher zurückliefert. Die Dateien werden verglichen, sodass der Benutzer wahrnehmen kann, welche der lokalen Dateien bereits auf dem entfernten Speicher abgelegt wurden und eine wiederholte Übertragung verhindert wird. Auf die Dateinamen werden *OnClickListener* registriert, die einen Upload initiieren können.

5.2.2.4. RemoteFileListActivity

Die Klasse ist eine Android-Activity Klasse 2.7.1.1, die eine Liste von Dateien auf dem entfernten Speicher darstellen soll. Der Zugriff erfolgt über DropBoxClient 5.2.2.2. Zusätzlich wird verglichen, ob eine Datei bereits lokal existiert, womit eine wiederholte Übertragung verhindert wird. Auf die Dateinamen werden *OnClickListener* registriert, die einen Download initiieren können.

5.2.2.5. UserListActivity

Die Klasse ist eine Android-Activity Klasse 2.7.1.1, die eine Liste von den im UserManager 5.2.1.2 des Benutzer-Managers 5.2.1 registrierten Benutzern darstellen soll. Der Zugriff erfolgt über DataBaseClient 5.2.2.1. Es wird zusätzlich abgefragt, welchen Benutzern der Abfragende das Zugriffsrecht auf seine Dateien erteilt hat. Diese Information fließt in die Darstellung ein. Auf die Benutzernamen werden *OnClickListener* registriert, die ein Hinzufügen bzw. Entfernen eines Benutzer zu der bzw. aus der Liste der Benutzer (siehe 5.2.1.2) erlaubt, mit denen Dateifreigabe unterhalten wird.

5.2.2.6. EncodedInput

EncodedInput ist eine Klasse mit den Eigenschaften zur Beibehaltung eines *AES*-Schlüssels als Byte-Array und eines verschlüsselten InputStreams.

5.2.2.7. Crypter

Die Klasse Crypter ist eine Tool-Klasse, bei der alle Methoden als statisch definiert sind. Sie übernimmt die Aufgaben der kryptographischen Ebene und setzt auf *Java Cryptography Architecture (JCA)* und *Java Cryptography Extension (JCE)* (siehe 2.7.2). Als *cryptographic service provider* wurde zuerst Java SDK Standard-Provider eingesetzt, was sich beim Einsatz auf einem Android-Endgerät nicht als funktionsfähig erwies. Daraufhin fiel die Wahl auf *Spongy Castle*-Provider [12]- eine an Android angepasste Version von *Bouncy Castle* [5].

```
static {
     Security.addProvider(new org.spongycastle.jce.provider.
         BouncyCastleProvider());
  }
```

Die Methoden der Klasse Crypter:

- ```
  public static EncodedInput getEncodedInput(InputStream inputStream) {
  }
  ```

  Der Parameter der Methode ist ein InputStream, der zu einem mit *AES* verschlüsselten InputStream umgewandelt wird. Der Rückgabewert ist ein Objekt der Klasse *EncodedInput* (siehe 5.2.2.6) mit dem aktuellen *AES*-Schlüssel und dem verschlüsselten InputStream.

- ```
  static public OutputStream getDecodedOutputStream(byte[] aeskey,
      OutputStream os) {
  }
  ```

  Die Parameter der Methode sind ein Byte-Array, aus dem ein *AES*-Schlüssel erstellt wird, und ein OutputStream, der zu einem verschlüsselten OutputStream als Rückgabewert umgewandelt wird.

- ```
  static public void createRSAKeys() {
  }
  ```

  Die Methode erstellt ein Schlüsselpaar für das asymmetrische Verschlüsselungsverfahren, in diesem Fall - *RSA*.

- ```
  static public byte[] cryptAsymmetric(byte[] data, byte[] rsaKey, String
      publicORprivate) {
  }
  ```

Die Methode erhält als Parameter einen Byte-Array, aus dem ein *RSA*-Schlüssel erstellt wird, einen Schalter als String, der signalisiert, ob der Algorithmus mit einem öffentlichen bzw. einem privaten Schlüssel arbeiten soll, und ein Byte-Array, der chiffriert wird.

```
static private PublicKey initRSAPublicFromBytes(byte[] data){
}
```

Die private Methode erstellt aus einem Byte-Array einen öffentlichen Schlüssel für das asymmetrische Verschlüsselungsverfahren.

```
static private PrivateKey initRSAPrivateFromBytes(byte[] data){
}
```

Die private Methode erstellt aus einem Byte-Array einen privaten Schlüssel für das asymmetrische Verschlüsselungsverfahren.

5.2.2.8. Service

Die Klasse Service ist eine Android-IntentService Klasse 2.7.1.2. Der Service wird von einem *OnClickListener* initiiert.
Es werden folgende Anfragen der Activity-Klassen behandelt:

Einen Benutzer zu der Liste für die Dateifreigabe hinzufügen

Die Methode

```
private void addShares4User(String guestID) {}
```

erfragt bei dem BAclientManager über DataBaseClient den öffentlichen Schlüssel des als Parameter übergebenen Benutzers:

```
byte[] publicKeyAsBytes = DataBaseClient.getPublicKeyFor(guestID);
```

Es wird über DropBoxClient die Liste aller Dateien erfragt, die dem zur Freigabe einladenden Benutzer gehören, und von jeder Datei wird ihr *Schlüssel-Chiffrat*[5] von dem entfernten Speicher geholt:

```
byte[] data = DropBoxClient.downloadByteArray(fileName);
```

[5] Als Schlüssel-Chiffrat wird hier das Ergebnis der Chiffrierung eines Schlüssels vom symmetrischen Verfahren mit einem öffentlichen Schlüssel vom asymmetrischen Verfahren bezeichnet

Durch Anwendung des eigenen privaten *RSA*-Schlüssels wird der *AES*-Schlüssel als Klartext erhalten

```
data = Crypter.cryptAsymmetric(data, Crypter.privateKey.getEncoded(), "
    private");
```

Der *AES*-Schlüssel als Klartext wird nun mit dem öffentlichen *RSA*-Schlüssel des fremden Benutzers chiffriert:

```
data = Crypter.cryptAsymmetric(data, publicKeyAsBytes, "public");
```

Das resultierende Chiffrat wird auf den entfernten Speicher übertragen:

```
InputStream is = new ByteArrayInputStream(data);
DropBoxClient.uploadInputStream(fileNameOnDBox, is, data.length);
```

Einen Benutzer aus der Liste für die Dateifreigabe entfernen

Die Methode

```
private void deleteShares4User(String guestID) {}
```

erfragt bei dem entfernten Speicher über DropBoxClient die Liste aller *Schlüssel-Chiffrate* [6], die von dem einladenden Benutzer für den als Parameter übergebenen fremden Benutzer bis jetzt erstellt wurden:

```
ArrayList<String> fileNames = DropBoxClient.getRemoteListing();
```

und initiiert ihre Löschung auf dem entfernten Speicher:

```
DropBoxClient.deleteFile(fileName);
```

Download einer Datei

Die Methode

```
private void downloadFile(String fileName) {
}
```

[6] Als Schlüssel-Chiffrat wird hier das Ergebnis der Chiffrierung eines Schlüssels vom symmetrischen Verfahren mit einem öffentlichen Schlüssel vom asymmetrischen Verfahren bezeichnet

bereitet eine Übertragung der Datei mit dem als Parameter übergebenen Namen von dem entfernten Speicher auf den lokalen Speicher vor. Als erstes wird das *Schlüssel-Chiffrat*[7] der Datei heruntergeladen:

```
1 byte[] data = DropBoxClient.downloadByteArray(keyFileName);
```

Durch Anwendung des eigenen privaten *RSA*-Schlüssels wird der *AES*-Schlüssel für die Datei erhalten:

```
1 data = Crypter.cryptAsymmetric(data, Crypter.privateKey.getEncoded(), "
    private");
```

Mit Hilfe von Crypter und des *AES*-Schlüssels wird ein verschlüsseltes OutputStream erzeugt, der beim Durchleiten der Zieldatei vom entfernten Speicher eine Entschlüsselung vornimmt. Das Herunterladen wird von DropBoxClient 5.2.2.2 übernommen:

```
1 OutputStream cipherOutputStream = Crypter.getDecodedOutputStream(data,
    outputStream);
2 DropBoxClient.downloadOutputStream(aesFileName, cipherOutputStream);
```

Upload einer Datei

Die Methode

```
1 private void uploadFile(String fileName) {
2 }
```

bereitet eine Übertragung der Datei mit dem als Parameter übergebenen Namen auf den entfernten Speicher vor. Mit Hilfe von Crypter 5.2.2.7 wird die Datei in einen verschlüsselten InputStream geleitet.

```
1 fileInputStream = new FileInputStream(f);
2 encodedInput = Crypter.getEncodedInput(fileInputStream);
```

Es muss berücksichtigt werden, ob es sich um eine Blockchiffre 2.4.1.1 oder eine Stromchiffre 2.4.1.1 handelt. Es wird im Blockchiffre-Modus mit 128Bit (16Byte) Blockgröße verschlüsselt, sodass die Größe der resultierenden Datei inklusive Padding2.4.1.1 ausgerechnet werden muss:

```
1 long filePaddedSize = (16 - (f.length() % 16)) + f.length();
```

[7] Als Schlüssel-Chiffrat wird hier das Ergebnis der Chiffrierung eines Schlüssels vom symmetrischen Verfahren mit einem öffentlichen Schlüssel vom asymmetrischen Verfahren bezeichnet

Die Datei wird als *AES*-verschlüsselter InputStream mit Hilfe von DropBoxClient 5.2.2.2 auf den entfernten Speicher übertragen.

```
DropBoxClient.uploadInputStream(
  fileNameOnDBox, encodedInput.inputStream, filePaddedSize);
```

Der *AES*-Schlüssel wurde dabei in einem Objekt der Klasse EncodedInput 5.2.2.6 abgelegt. Er wird mit dem öffentlichen *RSA*-Schlüssel des Inhabers der Datei verschlüsselt und mit Hilfe von DropBoxClient 5.2.2.2 auf den entfernten Speicher übertragen:

```
byte[] encryptedData =
  Crypter.cryptAsymmetric(
    encodedInput.keyAsByteArray, Crypter.publicKey.getEncoded(), "public
       ");
InputStream is = new ByteArrayInputStream(encryptedData);
DropBoxClient.uploadInputStream(
  fileNameOnDBox, is, encryptedData.length);
```

Anschließend wird eine Dateifreigabe für die eingeladenen Benutzer gewährt.

5.2.2.9. LogInActivity

Die Klasse LogInActivity ist eine Unterklasse von Activity 2.7.1.1 und ist als Starter-Klasse in AndroidManifest.xml festgelegt.

Wenn die Anwendung gestartet wird, wird während der Initialisierungsphase geprüft, ob ein Benutzer lokal existiert, und ob der Benutzer über seine privaten und öffentlichen Schlüssel verfügt. Wenn das nicht der Fall ist, wird das Schlüsselpaar mit Hilfe der Klasse *Crypter* 5.2.2.7 erstellt:

```
if (!privateKeyFile.exists() || !publicKeyFile.exists()) {
        Crypter.createRSAKeys();
        ...
}
```

Es folgt über DataBaseClient eine Registrierung bei dem BaClientManager 5.2.1:

```
DataBaseClient.addUserToDB(LogInActivity.myID, Crypter.publicKey.getEncoded());
```

Mit Hilfe von DropBoxClient und der Dropbox-API wird eine Authentifikation auf dem entfernten Speicher vorgenommen:

```
AndroidAuthSession session = buildSession();
```

```
DropBoxClient.mApi = new DropboxAPI<AndroidAuthSession>(session);
...
DropBoxClient.mApi.getSession().startAuthentication(LogInActivity.this);
...
DropBoxClient.mApi.getSession().finishAuthentication();
```

[11]©

Danach ist die Applikation einsatzbereit, sodass auf die lokalen bzw. entfernten Dateien und auf die Benutzerliste zugegriffen werden kann.

5.2.2.10. AndroidManifest.xml

In der Datei werden Einstellungen eingetragen, die beim Start notwendig sind 2.7.1.6.
Benötigte Rechte:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Die startende Activity:

```
<activity android:name=".LogInActivity" android:label="@string/app_name"
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Die anderen Activities:

```
<activity android:name=".UserListActivity" />
<activity android:name=".RemoteFileListActivity" />
<activity android:name=".LocalFileListActivity" />
```

und der Service:

```
<service android:name=".Service" />
```

6. Test

Der BAclientManager wird gestartet und hört auf ankommende Verbindungsanfragen von Clients:

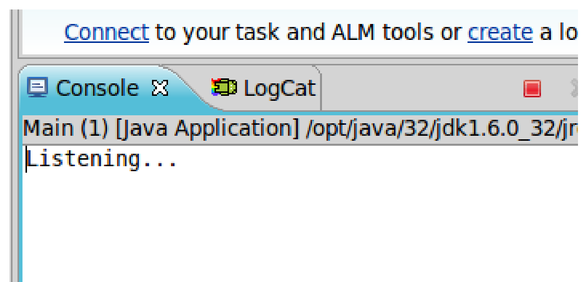
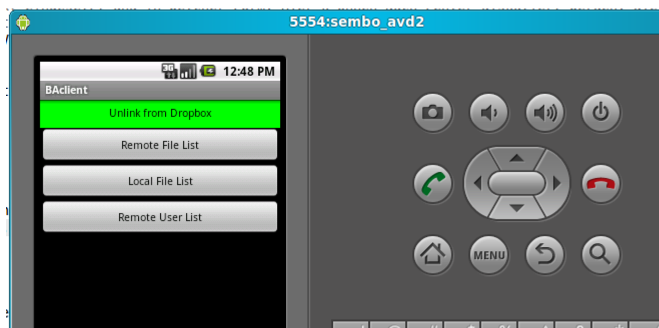
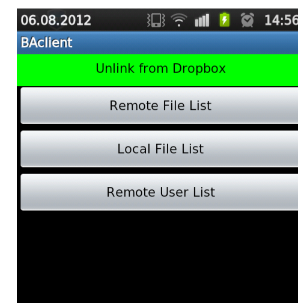


Abbildung 6.1.: BAclientManager gestartet

Es werden zwei Instanzen von BAclient gestartet:



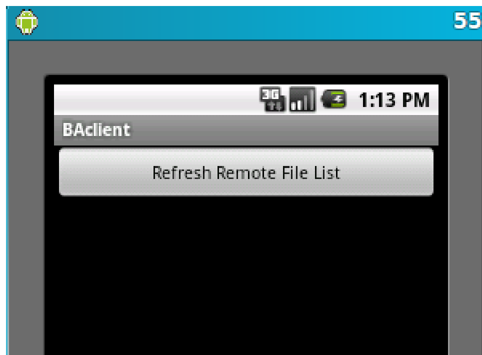
(a) Android-Emulator



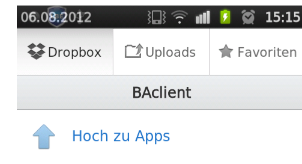
(b) Android-Handy

Abbildung 6.2.: BAclient gestartet

Der entfernte Speicher enthält momentan keine Dateien:



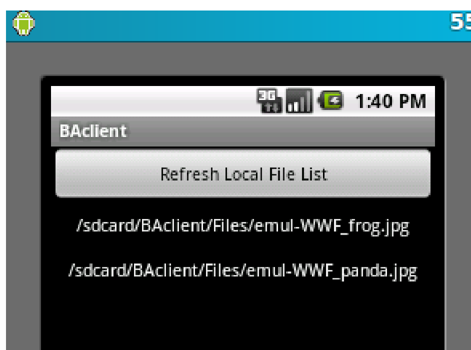
(a) Android-Emulator



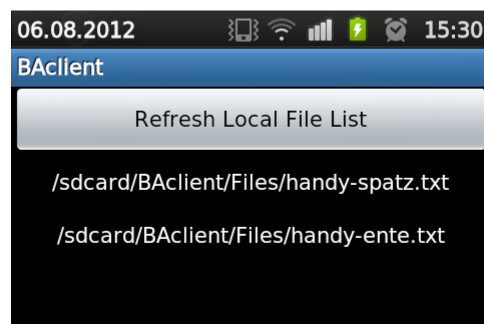
(b) Dropbox-App

Abbildung 6.3.: Keine Dateien auf dem entfernten Speicher

Der lokale Speicher enthält jeweils folgende Dateien:



(a) Android-Emulator



(b) Android-Handy

Abbildung 6.4.: Dateien auf dem lokalen Speicher

Eine Datei auf dem lokalen Speicher kann durch ihre Auswahl auf den entfernten Speicher übertragen werden. Es wird testweise eine Datei auf dem Emulator angeklickt. Der Emulator-Client soll die Datei verschlüsseln und auf die Dropbox schicken. Es wird durch eine Notification signalisiert, dass ein Upload stattfand:

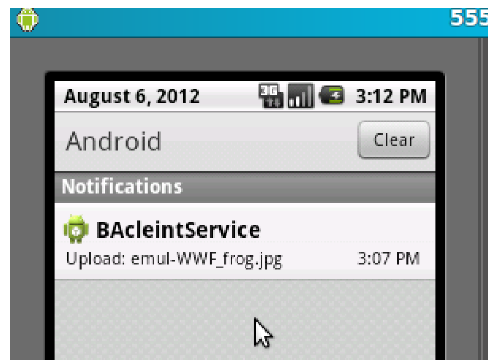


Abbildung 6.5.: Notification über Upload

Eine Überprüfung der Liste der Dateien mit dem Dropbox-Client zeigt, dass sich folgende Dateien auf dem entfernten Speicher befinden:

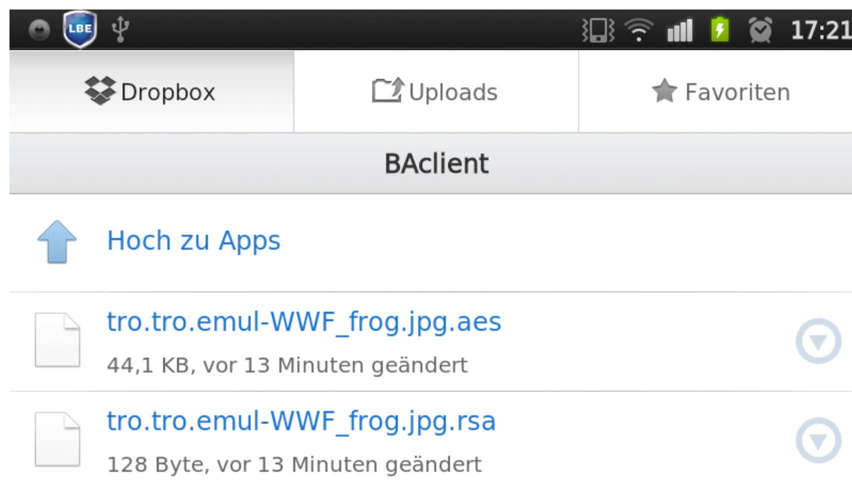


Abbildung 6.6.: Upload von einem BAclient

Die Datei mit der Endung *aes* ist die mit dem symmetrischen AES-Verfahren verschlüsselte Quelldatei, die in BAclient des Emulators angeklickt wurde. Die Datei mit der Endung *rsa* ist der mit dem asymmetrischen RSA-Verfahren chiffrierter *AES*-Schlüssel zu der Quelldatei. Dadurch, dass die *rsa*-Datei mitgeschickt wird, ist es dem Eigentümer möglich, auf die Quelldatei von einem beliebigen anderen BAclient zuzugreifen - unter der Voraussetzung, dass der private *RSA*-Schlüssel in dem anderen BAclient vorhanden ist. Der *AES*-Schlüssel wird nicht lokal aufbewahrt. Jede Datei, die übertragen wird, wird mit einem anderen zufälligen *AES*-Schlüssel

verschlüsselt.
Neben dem Transfer der Dateien fand eine Überprüfung der Liste der Benutzer statt, denen der Eigentümer Freigabe gewährt. Das ist in der Konsole des BAclientManagers zu sehen:

```
Listening...
Init ClientThread
Listening...
request is: getMySharedUsers
request is: close
close command
```

Eine Überprüfung der Liste der Benutzer zeigt, dass keine Benutzer für die Freigabe zugelassen sind (sonst mit gelb markiert):

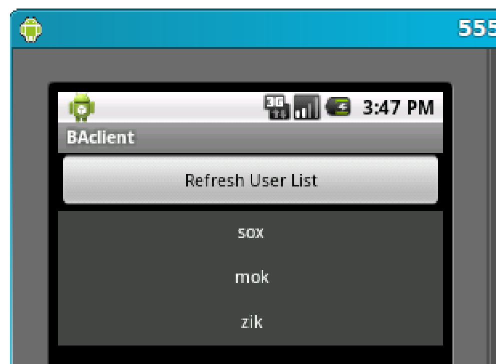


Abbildung 6.7.: Freigabe keinem Benutzer gewährt

Bei der Abfrage der Dateiliste auf dem entfernten Speicher durch den BAclient mit dem Namen *mok* auf dem Handy werden keine Dateien sichtbar sein und stehen somit nicht zum Download zur Verfügung. Dies soll allerdings nur eine benutzerfreundliche Darstellungsform sein, denn die Dateien sind vorhanden, aber sie sind nicht entschlüsselbar.

Der Eigentümer der Datei kann durch die Wahl eines anderen Benutzers, diesem die Dateifreigabe gewähren. Die Markierung wird gelb:

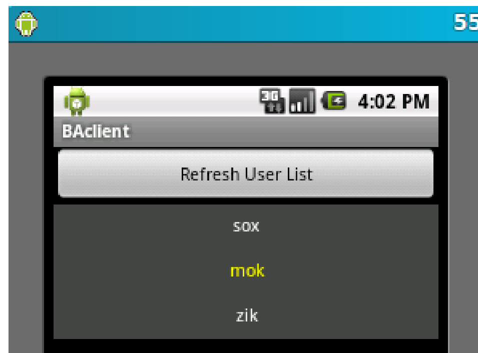


Abbildung 6.8.: Freigabe dem Benutzer *mok* gewährt

Dabei findet ein Zugriff auf den BAclientManager statt, der den öffentlichen *RSA*-Schlüssel des Benutzers *mok* erfragt und damit den *AES*-Schlüssel für die Quelldatei chiffriert. Als Ergebnis wird mit dem Dropbox-Client sichtbar, dass eine zusätzliche *rsa*-Datei auf den entfernten Speicher übertragen wurde:

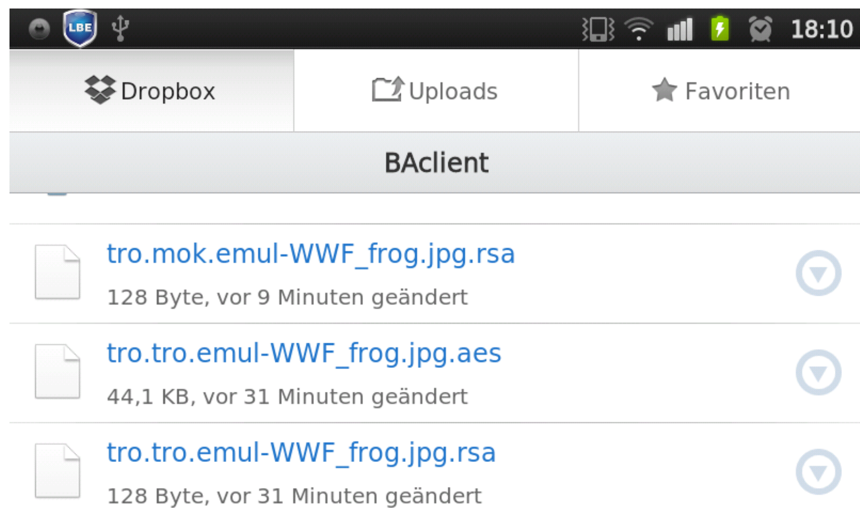


Abbildung 6.9.: Ergebnis der Freigabe für den Benutzer *mok*

Der Zugriff mit einem BAclient auf dem Handy-Endgerät des Benutzers *mok* zeigt, dass für ihn eine Datei freigegeben wurde:

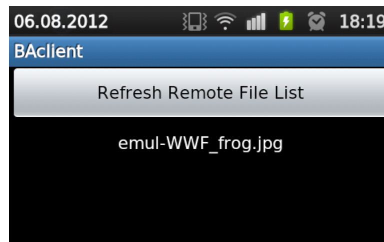


Abbildung 6.10.: freigegebene Datei für den Benutzer *mok*

Durch Anklicken der Datei ist es dem Benutzer *mok* mit dem BAclient möglich, diese auf seinen lokalen Speicher zu übertragen. Es bleibt dem Benutzer verborgen, dass für den Zugriff auf die Quelldatei zuerst die *rsa*-Datei, die vom Eigentümer extra für ihn erstellt wurde, verarbeitet wird, sodass daraus der *AES*-Schlüssel zu der Quelldatei dechiffriert wird, und anschließend eine Entschlüsselung der Quelldatei während des Transfers stattfindet.

Der Eigentümer kann die Freigabe dem Benutzer *mok* wieder entziehen, indem der Benutzer in der Liste durch ein erneutes Anklicken abgewählt wird. Dabei wird die *rsa*-Datei für den Benutzer *mok* vom entfernten Speicher gelöscht. Die Quelldatei wird nicht neu verschlüsselt, da davon ausgegangen werden kann, dass der Benutzer *mok* sie schon heruntergeladen und entschlüsselt hat.
Diese Vorgehensweise gilt entsprechend, wenn der Upload und die Freigabe von einer Menge beliebiger anderen Benutzer initiiert wird, allerdings wurden die Tests beidseitig nur für zwei Benutzer durchgeführt.

7. Zusammenfassung

Rechnergestützte Datenverarbeiteung ist längst ein Teil des privaten und des geschäftlichen Lebens eines jeden Menschen geworden. Wie das Wort Datenverarbeitung bereits sagt, geht es um Daten und Dokumente, die in digitaler Form existieren. Sie müssen digital aufbewahrt und übertragen werden. Fortschritt und neue Technologien bieten den Menschen ein hohes Maß an Mobilität. Dabei stellt sich die Frage, auf welche seine Daten und Dokumente er immer Zugriff haben muss. Diese sind dann entweder physikalisch mitzunhemen, was der Mitnahme eines lokalen Speichers entsprechen würde, oder der Aufbewahrungsort muss so organisiert werden, dass es einen Verbindungskanal dazu gibt. Des Weiteren ist die Komponente der Interaktion zwsichen den Menschen genauso wichtig, sodass eine Verteilung und eine Weitergabe der Daten und Dokumente möglich sein soll. Die Überlegungen zu den Begriffen Datenaufbewahrungort, genannt entfernter Speicher, und Datenverteilung liegen zugrunde dieser Arbeit. Wenn der Aufbewahrungsort nicht lokal ist, stellt sich die Frage, wie eine Übertragung organisiert werden soll. Und wenn der Aufbewahrungsort nicht dem Eigentümer der Daten gehört, muss über die Form ihrer Speicherung nachgedacht werden, sodass die Daten immer noch privat bleiben. Und wenn die Daten weitergegeben werden sollen, ist festzustellen, wie das selektiv nur an ganze bestimmte Identitäten möglich gemacht werden kann. Es soll untersucht werden, inwiefern die Aufbewahrung, die Übertragung und die Verteilung der Daten auf eine sichere Art und Weise geschehen kann.
Die festgelegten Anforderung lassen sich zu einem Szenario zuordnen, dass auf einer Client-Server-Architektur basiert, denn der Aufbewahrungsort ist der dienstanbietende Server, und der Egentümer der Daten ist der Client, der den Serverdienst in Anspruch nimmt. Ausgehend vom Konzept der Client-Server-Architektur wurden Merkmale festgestellt, die aus Sicherheitsgründen erfüllt sein müssen. Dazu zählen Zugriffskontrolle, Authentizität, Vertraulichkeit und Integrität. Mit Hilfe der Zugriffskontrolle soll der der Server kontrollieren, welchen Clients welche Dienste und Rechte erlaubt sind. Dazu muss ein Client seine Authentizität nachweisen können. Bei mehreren Clients muss durch die abgeschirmte Bedienung die Vertraulichkeit für einzelne Clients garantiert werden und die Intergrität - die Echtheit der Daten überprüfbar sein. Aus den zu erfüllenden Sicherheitsmerkmalen wurden die Sicherheitsanforderungen abgelei-

tet: sicheres Einloggen/Anmelden für Identifikation und Autorisation eines Clients, sicherer Übertragungskanal auf dem offenen Netz, sichere Datenaufbewahrung für die Vetraulichkeit und sichere Dateifreigabe für ausgewählte Clients. Eine Reihe Anbieter wurden analysiert, inwiefern sie die Sicherheitsmerkmale erfüllen. Als Gesamtbild stellte sich heraus, dass nicht alle Sicherheitsanforderungen erfüllt werden, sodass eine eigene clientseitige Sicherheitsarchitektur für sinnvoll gehalten wurde. Eine Idee für ihre Umsetzung wurde entwickelt, und entsprechende Komponenten wurden vorgestellt: ein Benutzer-Manager ist eine Vetrauensinstanz, die eine Datenbank mit registrierten Clients verwaltet und ihre Anonymität gegenüber dem Speicherdienstranbieter ermöglicht. Ein Benutzer-Client wird auf dem Endgerät des Benutzers installiert. Er nimmt die Dienste des entfernten Speichers und des Benutzer-Managers in Anspruch. Die Kryptographie wurde als Mittel zur Erfüllung der Sicherheitsanforderungen ausgewählt, sodass symmetrische und asymmetrische Verschlüsselungsverfahren und Hashfunktionen zum Einsatz kommen. Eine Realisierung eines prototypischen Softwaremoduls wurde vorgenommen, der einen konkreten Dienstanbieter für den entfernten Speicher einsetzt und mit Hilfe seiner API die kommunikation vom dem Benutzer-Client initiiert. Der Benutzer-Client ist eine Android-Anwendung, die Dateien auf den entfernten Speicher hochladen bzw. davon herunterladen und für andere Clients freigeben kann. Für die Dateifreigabe muss die Benutzerdatenbank des Benutzer-Managers abgefragt werden. Für den Nachrichtenaustausch zwischen dem Benutzer-Client und dem Benutzer-Manager wurde ein eigenes Protokoll entwickelt, welches einem erlaubt, Benutzer hinzuzufügen, zur Dateifreigabe auszuwählen bzw. wieder abzuwählen, die Listen der Benutzer und ihre öffentlichen Schlüssel abzufragen. Anschließend wurden Tests mit zwei Clients durchgeführt und dokumentiert.

A. BAclientManager-Quellcode

A.1. SSL-Listener.java

```
1  package de.sembo.manager;

3  import java.io.DataInputStream;
4  import java.io.DataOutputStream;
5  import java.io.IOException;
6  import java.net.ServerSocket;
7  import java.net.Socket;
8  import java.security.Key;

10 public class SSL_Listener {

12    private int port;
13    private UserManager um;

15    public SSL_Listener(UserManager um) {
16       this.port = 32000;
17       this.um = um;
18    }

20    public void startServer() {
21       ServerSocket server = null;
22       Socket client = null;
23       try {
24          // listen on port
25          server = new ServerSocket(port);

27          while (true) {
28             System.out.println("Listening...");
29             client = server.accept();
30             /* start a new thread to handle this client */
31             Thread t = new Thread(new ClientThread(client));
32             t.start();
33          }
34       } catch (IOException e) {
35          e.printStackTrace();
36       }
37    }

39       class ClientThread implements Runnable {
40           private Socket client;
41           public ClientThread(Socket client) {
42            System.out.println("Init ClientThread");
43               this.client = client;
44           }

46           public void run() {;

48               DataInputStream dis = null;
49               DataOutputStream dos = null;
50               try {
51                   dis = new DataInputStream(client.getInputStream());
```

```
52                  dos = new DataOutputStream(client.getOutputStream());
53              } catch (IOException e) {
54                  System.err.println(e);
55                  return;
56              }

58              boolean run = true;
59              int size;
60              byte[] encodedAesKey;
61              try {
62                 //init interaction Protocoll
63                 size = dis.readInt();
64                 encodedAesKey = new byte[size];
65                 dis.read(encodedAesKey, 0, size);
66                 System.out.println("request aes key: " + encodedAesKey);
67 //              Key decodedAesKey = decodeWith_RSA(encodedAesKey);
68                 String answer1 = um.getAvailableCommands();
69 //              encodeWith_AES(decodedAesKey, answer1);
70                 dos.writeInt(answer1.getBytes().length);
71                 dos.write(answer1.getBytes(), 0, answer1.getBytes().length);


74                 byte[] array;
75                 while(run){
76                    System.out.println("Interaction is running...");
77                    size = dis.readInt();
78                    System.out.println("bla");
79                    array = new byte[size];
80                    dis.read(array, 0, size);
81                    String request = new String(array) ;
82                    System.out.println("request is: " + request);
83                    if(request.equalsIgnoreCase("close")){
84                       run = false;
85                       System.out.println("close command");
86                       continue;
87                    }
88                    if(request.equalsIgnoreCase("addUserToDB")){
89                       //user id einlesen
90                       size = dis.readInt();
91                       array = new byte[size];
92                       dis.read(array, 0, size);
93                       String ID = new String(array) ;
94                       //public key empfangen
95                       size = dis.readInt();
96                       array = new byte[size];
97                       dis.read(array, 0, size);
98                       //zur datenbank hinzufuegen
99                       answer1 = um.addUserToDB(ID, array);
100                      //antwort zurueckschicken
101                      dos.writeInt(answer1.getBytes().length);
102                      dos.write(answer1.getBytes(), 0, answer1.getBytes().length);
103                      continue;
104                   }
105                   if(request.equalsIgnoreCase("getUserList")){
106                      size = dis.readInt();
107                      array = new byte[size];
108                      dis.read(array, 0, size);
109                      String ID = new String(array) ;
110                      answer1 = um.getUserList(ID);
111                      dos.writeInt(answer1.getBytes().length);
112                      dos.write(answer1.getBytes(), 0, answer1.getBytes().length);
113                      continue;
114                   }
115                   if(request.equalsIgnoreCase("addOrRemoveToMySharedUsers")){
116                      size = dis.readInt();
117                      array = new byte[size];
118                      dis.read(array, 0, size);
119                      String myID = new String(array) ;
```

```
120                     size = dis.readInt();
121                     array = new byte[size];
122                     dis.read(array, 0, size);
123                     String guestID = new String(array) ;
124                     answer1 = um.addOrRemoveToMySharedUsers(myID, guestID);
125                     dos.writeInt(answer1.getBytes().length);
126                     dos.write(answer1.getBytes(), 0, answer1.getBytes().length);
127                     continue;
128                  }
129                  if(request.equalsIgnoreCase("getPublicKeyFor")){
130                     size = dis.readInt();
131                     array = new byte[size];
132                     dis.read(array, 0, size);
133                     String guestID = new String(array) ;
134                     byte[] data = um.getPublicKeyFor(guestID);
135                     dos.writeInt(data.length);
136                     dos.write(data, 0, data.length);
137                     continue;
138                  }
139                  if(request.equalsIgnoreCase("getMySharedUsers")){
140                     size = dis.readInt();
141                     array = new byte[size];
142                     dis.read(array, 0, size);
143                     String myID = new String(array);
144                     answer1 = um.getMySharedUsers(myID);
145                     dos.writeInt(answer1.getBytes().length);
146                     dos.write(answer1.getBytes(), 0, answer1.getBytes().length);
147                     continue;
148                  }

150                  answer1 = "unknown command";
151                  dos.writeInt(answer1.getBytes().length);
152                  dos.write(answer1.getBytes(), 0, answer1.getBytes().length);

154               }

156            } catch (IOException e) {
157               System.err.println(e);
158            }
159         }
160      }

162      class AEScodec{

164         byte[] input;
165         byte[] output;
166         Key aesKey;

168         public AEScodec(byte[] keyAsArray, byte[] input, String codecMode){
169            this.input = input;
170         }

172         byte[] code(){
173            output = input;
174         return output;
175         }
176      }

178 }
```

A.2. UserManager.java

```
1 package de.sembo.manager;

3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
```

```
5  import java.io.File;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.FileReader;
10 import java.io.FileWriter;
11 import java.io.FilenameFilter;
12 import java.io.IOException;
13 import java.util.ArrayList;

15 public class UserManager {

17    private String folderName;

19    public UserManager() {
20       this.folderName = "userDB";
21    }

23    public String addUserToDB(String ID, byte[] data) {
24       System.out.println("addUserToDB");
25       // Look If User with this name Already exists
26       String filePrefix = ID + ".publicKey";
27       String answer = "";
28       File folder = new File(folderName);
29       if (folder.exists()) {
30          File[] fileList = folder.listFiles();
31          for (int i = 0; i < fileList.length; i++) {
32             System.out.println("userDBfolder contains: "
33                   + fileList[i].getName());
34          }
35          for (File f : fileList) {
36             if (f.getName().equalsIgnoreCase(filePrefix)) {
37                return "UserAlreadyExists";
38             }
39          }
40          try {
41             FileOutputStream fileOutputStream = new FileOutputStream(
42                   new File(folder + "/" + ID + ".publicKey"));
43             fileOutputStream.write(data);
44             fileOutputStream.close();
45             fileOutputStream = new FileOutputStream(new File(folder + "/"
46                   + ID + ".users"));
47             fileOutputStream.close();
48             return "UserAdded";

50          } catch (FileNotFoundException e) {
51             e.printStackTrace();
52          } catch (IOException e) {
53             e.printStackTrace();
54          }
55       }
56       return answer;
57    }

59    public String getUserList(String myID) {
60       System.out.println("getUserList");
61       String result = "";
62       File folder = new File(folderName);
63       if (folder.exists()) {
64          File[] fileList = folder.listFiles(new publickeyFilenameFilter());
65          for (File f : fileList) {
66             // mich selbst rausfiltern
67             if (!f.getName().equalsIgnoreCase(myID + ".publicKey")) {
68                String ID = f.getName().replaceAll(".publicKey", "");
69                result = result + ID + ";";
70             }
71          }
72       }
```

```
73        return result;
74     }

76     public String getMySharedUsers(String myID) {
77        String result = "";
78        ArrayList<String> users;
79        users = getStringsFromFile(folderName + "/" + myID + ".users");
80        if (users != null) {
81           for (String user : users) {
82              result = result + user + ";";
83           }
84        }
85        return result;
86     }

88     public String addOrRemoveToMySharedUsers(String myID, String guestID) {
89        if(myID.equalsIgnoreCase(guestID)){
90           return "bad name";
91        }
92        File fi = new File(folderName + "/" + myID + ".publicKey");
93        if(!fi.exists()){
94           return "You R Not In DB";
95        }
96        File folder = new File(folderName);
97        String usersSharedFileName = folderName + "/" + myID + ".users";
98        String result = "DB folder does not exist";
99        if (folder.exists()) {
100          File[] fileList = folder.listFiles(new publickeyFilenameFilter());
101          for (File f : fileList) {
102             // ob es den User im Ordner gibt
103             if (f.getName().equalsIgnoreCase(guestID + ".publicKey")) {
104                boolean userInSharedList = false;
105                // System.out.println(userInSharedList);
106                int cnt = -1;
107                ArrayList<String> usersShared = getStringsFromFile(usersSharedFileName);

109                for (String user : usersShared) {
110                   cnt = cnt + 1;
111                   if (user.equalsIgnoreCase(guestID + ";")) {
112                      userInSharedList = true;
113                      break;
114                   }
115                }
116                if (userInSharedList) {
117                   // remove user
118                   usersShared.remove(cnt);
119                   BufferedWriter bw = null;
120                   try {
121                      FileWriter fw = new FileWriter(usersSharedFileName);
122                      bw = new BufferedWriter(fw);
123                      for (String string : usersShared) {
124                         String[] parsed = string.split(";");
125                         bw.write(parsed[0]+ ";");
126                         bw.newLine();
127                      }
128                      bw.close();
129                      return "user removed from shared list";
130                   } catch (IOException e) {
131                      e.printStackTrace();
132                   }
133                } else {
134                   // add user
135                   try {
136                      FileWriter fw = new FileWriter(usersSharedFileName,
137                            true);
138                      BufferedWriter bw = new BufferedWriter(fw);
139                      bw.write(guestID + ";");
140                      bw.newLine();
```

```
141                 bw.close();
142                 return "user added to shared list";
143             } catch (FileNotFoundException e) {
144                 e.printStackTrace();
145             } catch (IOException e) {
146                 e.printStackTrace();
147             }
148         }

150     } else {
151         result = "User does not exist";
152     }
153   }
154 }
155 return result;
156 }

158 public byte[] getPublicKeyFor(String userID){



162   System.out.println("getPublicKeyFor()" + userID);
163   File publicKeyFile = new File(folderName +"/" + userID + ".publicKey");
164   byte[] data = new byte[0];

166   if(publicKeyFile.exists()){

168     System.out.println("Size of key file: "+ publicKeyFile.length());
169     FileInputStream fileInputStream;
170     try {
171       fileInputStream = new FileInputStream(publicKeyFile);
172       data = new byte[(int) publicKeyFile.length()];
173       fileInputStream.read(data);
174       fileInputStream.close();
175     } catch (FileNotFoundException e) {
176       // TODO Auto-generated catch block
177       e.printStackTrace();
178     } catch (IOException e) {
179       // TODO Auto-generated catch block
180       e.printStackTrace();
181     }
182   }
183   return data;
184 }

186 private ArrayList<String> getStringsFromFile(String fileName) {

188   ArrayList<String> strings = new ArrayList<String>();
189   File f = new File(fileName);
190   if(f.exists()){

192     BufferedReader in;
193     try {
194       in = new BufferedReader(new FileReader(fileName));
195       String line;
196       // wenn datei nicht leer
197       if ((line = in.readLine()) != null) {
198         strings = new ArrayList<String>();
199         strings.add(line);
200         while ((line = in.readLine()) != null) {
201           strings.add(line);
202         }
203       }
204       in.close();
205     } catch (FileNotFoundException e) {

207       e.printStackTrace();
208     } catch (IOException e) {
```

```
209         e.printStackTrace();
210       }
211     }
212     return strings;
213   }

215   public String getAvailableCommands(){
216     String commandsList = "addUserToDB;getPublicKeyFor;addOrRemoveToMySharedUsers;getMySharedUsers;getUserList;";
217     return commandsList;
218   }
219 }

221 class publickeyFilenameFilter implements FilenameFilter {
222   public boolean accept(File f, String s) {
223     return s.toLowerCase().endsWith(".publickey");
224   }
225 }
```

A.3. User.java

```
1 package de.sembo.manager;

3 import java.security.Key;

5 public class User {

7   private int userID;
8   private Key publicKey;

10   public User(int userID, Key publicKey){
11     this.userID = userID;
12     this.publicKey = publicKey;
13   }

15   public int getUserID() {
16     return userID;
17   }

19   public Key getPublicKey() {
20     return publicKey;
21   }
22 }
```

A.4. Main.java

```
1 package de.sembo.manager;

3 public class Main {

5   public static void main(String[] args) {

7     UserManager um = new UserManager();
8     SSL_Listener ssll = new SSL_Listener(um);
9     ssll.startServer();
10   }
11 }
```

B. BAclient-Quellcode

B.1. DataBaseClient.java

```java
package de.sembo;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;

public class DataBaseClient {

   static String ipAddress;
   static int ipPort;

   static public ArrayList<String> getUserList() {

      Socket socket = null;
      DataOutputStream dos = null;
      DataInputStream dis = null;
      ArrayList<String> userList = new ArrayList<String>();

      try {
         socket = new Socket(ipAddress, ipPort);
         dos = new DataOutputStream(socket.getOutputStream());
         dis = new DataInputStream(socket.getInputStream());

         // init interaction
         String value = "ssl";
         dos.writeInt(value.getBytes().length);
         dos.write(value.getBytes(), 0, value.getBytes().length);

         int size = dis.readInt();
         byte[] array = new byte[size];
         dis.read(array, 0, size);
         System.out.println(new String(array));

         // all users list holen
         value = "getUserList";
         dos.writeInt(value.getBytes().length);
         dos.write(value.getBytes(), 0, value.getBytes().length);
         dos.writeInt(LogInActivity.myID.getBytes().length);
         dos.write(LogInActivity.myID.getBytes(), 0,
              LogInActivity.myID.getBytes().length);

         size = dis.readInt();
         array = new byte[size];
         dis.read(array, 0, size);

         value = "close";
         dos.writeInt(value.getBytes().length);
         dos.write(value.getBytes(), 0, value.getBytes().length);
```

```
52          dos.close();
53          dis.close();
54          socket.close();

56          String answer = new String(array);
57          String[] parsedAnswer = answer.split(";");

59          for (String userName : parsedAnswer) {
60             userList.add(userName);
61          }

63       } catch (UnknownHostException e) {
64          e.printStackTrace();
65       } catch (IOException e) {
66          e.printStackTrace();
67       }

69       return userList;
70    }

72    static public ArrayList<String> getSharedUserList() {

74       Socket socket = null;
75       DataOutputStream dos = null;
76       DataInputStream dis = null;
77       ArrayList<String> sharedUserList = new ArrayList<String>();

79       try {
80          socket = new Socket(ipAddress, ipPort);
81          dos = new DataOutputStream(socket.getOutputStream());
82          dis = new DataInputStream(socket.getInputStream());

84          String value = "ssl";
85          dos.writeInt(value.getBytes().length);
86          dos.write(value.getBytes(), 0, value.getBytes().length);
87          int size = dis.readInt();
88          byte[] array = new byte[size];
89          dis.read(array, 0, size);
90          System.out.println(new String(array));

92          // shared List holen
93          value = "getMySharedUsers";
94          dos.writeInt(value.getBytes().length);
95          dos.write(value.getBytes(), 0, value.getBytes().length);
96          dos.writeInt(LogInActivity.myID.getBytes().length);
97          dos.write(LogInActivity.myID.getBytes(), 0,
98                LogInActivity.myID.getBytes().length);

100         size = dis.readInt();
101         array = new byte[size];
102         dis.read(array, 0, size);

104         value = "close";
105         dos.writeInt(value.getBytes().length);
106         dos.write(value.getBytes(), 0, value.getBytes().length);
107         dos.close();
108         dis.close();
109         socket.close();

111         String answer = new String(array);
112         String[] parsedAnswer = answer.split(";");

114         for (String userName : parsedAnswer) {
115            if(userName.equalsIgnoreCase("")){

117            }else if(userName.equalsIgnoreCase("\n")){

119            }else if(userName.equalsIgnoreCase("\r\n")){
```

```
121         }else{
122            sharedUserList.add(userName);
123         }
124      }
125   } catch (UnknownHostException e) {
126      e.printStackTrace();
127   } catch (IOException e) {
128      e.printStackTrace();
129   }
130   return sharedUserList;
131 }

133 static public String addOrRemoveToMySharedUsers(String guestID) {
134   String answer = "";
135   Socket socket = null;
136   DataOutputStream dos = null;
137   DataInputStream dis = null;

139   try {
140      socket = new Socket(ipAddress, ipPort);
141      dos = new DataOutputStream(socket.getOutputStream());
142      dis = new DataInputStream(socket.getInputStream());

144      // aes key schicken
145      String value = "ssl";
146      dos.writeInt(value.getBytes().length);
147      dos.write(value.getBytes(), 0, value.getBytes().length);
148      int size = dis.readInt();
149      byte[] array = new byte[size];
150      dis.read(array, 0, size);
151      System.out.println(new String(array));

153      // trigger on shared List
154      value = "addOrRemoveToMySharedUsers";
155      dos.writeInt(value.getBytes().length);
156      dos.write(value.getBytes(), 0, value.getBytes().length);
157      dos.writeInt(LogInActivity.myID.getBytes().length);
158      dos.write(LogInActivity.myID.getBytes(), 0,
159            LogInActivity.myID.getBytes().length);

161      dos.writeInt(guestID.getBytes().length);
162      dos.write(guestID.getBytes(), 0,
163            guestID.getBytes().length);

165      size = dis.readInt();
166      array = new byte[size];
167      dis.read(array, 0, size);

169      answer = new String(array);
170      System.out.println(answer);

172      value = "close";
173      dos.writeInt(value.getBytes().length);
174      dos.write(value.getBytes(), 0, value.getBytes().length);
175      dos.close();
176      dis.close();
177      socket.close();
178   } catch (UnknownHostException e) {
179      e.printStackTrace();
180   } catch (IOException e) {
181      e.printStackTrace();
182   }
183   return answer;
184 }

186 static public byte[] getPublicKeyFor(String guestID){
```

```
188     byte[] publicKey = new byte[0];

190     Socket socket = null;
191     DataOutputStream dos = null;
192     DataInputStream dis = null;

194     try {
195       socket = new Socket(ipAddress, ipPort);
196       dos = new DataOutputStream(socket.getOutputStream());
197       dis = new DataInputStream(socket.getInputStream());

199       String value = "ssl";
200       dos.writeInt(value.getBytes().length);
201       dos.write(value.getBytes(), 0, value.getBytes().length);
202       int size = dis.readInt();
203       byte[] array = new byte[size];
204       dis.read(array, 0, size);
205       System.out.println(new String(array));

207       value = "getPublicKeyFor";
208       dos.writeInt(value.getBytes().length);
209       dos.write(value.getBytes(), 0, value.getBytes().length);
210       dos.writeInt(guestID.getBytes().length);
211       dos.write(guestID.getBytes(), 0, guestID.getBytes().length);

213       // public key des user holen
214       size = dis.readInt();
215       array = new byte[size];
216       dis.read(array, 0, size);
217       publicKey = array;

219       String answer = new String(array);
220       System.out.println(answer);

222       value = "close";
223       dos.writeInt(value.getBytes().length);
224       dos.write(value.getBytes(), 0, value.getBytes().length);
225       dos.close();
226       dis.close();
227       socket.close();
228     } catch (UnknownHostException e) {
229       e.printStackTrace();
230     } catch (IOException e) {
231       e.printStackTrace();
232     }
233     return publicKey;
234   }

236   static public String addUserToDB(String ID, byte[] publicKey){

238     Socket socket = null;
239     DataOutputStream dos = null;
240     DataInputStream dis = null;
241     String answer = "";

243     try {
244       socket = new Socket(ipAddress, ipPort);
245       dos = new DataOutputStream(socket.getOutputStream());
246       dis = new DataInputStream(socket.getInputStream());

248       String value = "ssl";
249       dos.writeInt(value.getBytes().length);
250       dos.write(value.getBytes(), 0, value.getBytes().length);
251       int size = dis.readInt();
252       byte[] array = new byte[size];
253       dis.read(array, 0, size);
254       System.out.println(new String(array));
```

```
256         // meine Kennung und public key schicken
257         value = "addUserToDB";
258         dos.writeInt(value.getBytes().length);
259         dos.write(value.getBytes(), 0, value.getBytes().length);
260         dos.writeInt(ID.getBytes().length);
261         dos.write(ID.getBytes(), 0, ID.getBytes().length);

263         dos.writeInt(publicKey.length);
264         dos.write(publicKey, 0,
265                 publicKey.length);

267         size = dis.readInt();
268         array = new byte[size];
269         dis.read(array, 0, size);
270         answer = new String(array);
271         System.out.println(answer);


274         value = "close";
275         dos.writeInt(value.getBytes().length);
276         dos.write(value.getBytes(), 0, value.getBytes().length);
277         dos.close();
278         dis.close();
279         socket.close();
280       } catch (UnknownHostException e) {
281         e.printStackTrace();
282       } catch (IOException e) {
283         e.printStackTrace();
284       }
285       return answer;
286   }
287 }
```

B.2. DropBoxClient.java

```
1  package de.sembo;

3  import java.io.ByteArrayInputStream;
4  import java.io.ByteArrayOutputStream;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.OutputStream;
8  import java.util.ArrayList;
9  import java.util.List;

11 import android.util.Log;

13 import com.dropbox.client2.DropboxAPI;
14 import com.dropbox.client2.DropboxAPI.DropboxFileInfo;
15 import com.dropbox.client2.DropboxAPI.Entry;
16 import com.dropbox.client2.android.AndroidAuthSession;
17 import com.dropbox.client2.exception.DropboxException;
18 import com.dropbox.client2.session.Session.AccessType;

20 public class DropBoxClient {
21   static DropboxAPI<AndroidAuthSession> mApi;
22   final static String APP_KEY = "jb2g1m7vi27rhte";
23   final static String APP_SECRET = "xq6apebciouzfnp";
24   final static AccessType ACCESS_TYPE = AccessType.APP_FOLDER;

26   // You don't need to change these, leave them alone.
27   final static String ACCOUNT_PREFS_NAME = "prefs";
28   final static String ACCESS_KEY_NAME = "ACCESS_KEY";
29   final static String ACCESS_SECRET_NAME = "ACCESS_SECRET";

31   static public ArrayList<String> getRemoteListing() {
```

```
33      ArrayList<String> fileNames = new ArrayList<String>();
34      if (mApi.getSession().isLinked()) {

36        Entry filesListEntry = new Entry();
37        try {
38          filesListEntry = mApi.metadata("", 1000, null,
39              true, null);
40          List<Entry> fileNameEntries = filesListEntry.contents;
41          if (fileNameEntries != null) {
42            for (Entry ent : fileNameEntries) {
43              fileNames.add(ent.fileName());
44            }
45          }
46        } catch (DropboxException e) {
47          e.printStackTrace();
48        }
49      }
50      return fileNames;
51    }

53    static public void uploadInputStream(String fileName, InputStream is, long size) {
54      System.out.println("uploadInputStream:" +fileName);
55      try {
56        Entry newEntry = mApi.putFileOverwrite(fileName, is,
57            size, null);
58        Log.i("DbExampleLog", "The uploaded file's rev is: " + fileName);
59        // + newEntry.rev + " : "
60      } catch (DropboxException e) {
61        e.printStackTrace();
62      }
63    }

65    static public byte[] downloadByteArray(String fileName) {

67      byte[] dl = new byte[0];
68      ByteArrayOutputStream keyByteArrayOutputStream = new ByteArrayOutputStream();

70      DropboxFileInfo infoKey;
71      try {
72        infoKey = mApi.getFile("/" + fileName, null,
73            keyByteArrayOutputStream, null);
74        Log.i("DbExampleLog", "The file's rev is: "
75            + infoKey.getMetadata().rev + " ; name: " + fileName);

77      } catch (DropboxException e) {
78        e.printStackTrace();
79      }finally{
80        if (keyByteArrayOutputStream != null) {
81          try {
82            keyByteArrayOutputStream.close();
83          } catch (IOException e) {
84            e.printStackTrace();
85          }
86        }
87      }

89      dl = keyByteArrayOutputStream.toByteArray();
90      System.out.println("downloadByteArray(): " + fileName + dl.length);
91      return dl;
92    }

94    static public boolean downloadOutputStream(String fileName, OutputStream os) {
95      System.out.println("downloadOutputStream(): " + fileName);
96  //    OutputStream cipherOutputStream = null;
97      try {
98        DropboxFileInfo info = mApi.getFile(
99            "/" + fileName, null, os, null);
```

```
100         Log.i("DbExampleLog", "The file's rev is: "
101               + info.getMetadata().rev + " ; name: " + fileName);
102         return true;

104      } catch (DropboxException e) {
105         e.printStackTrace();
106         return false;
107      } finally {
108      }
109   }

111   static public void uploadByteArray(byte[] bytearray, String fileName) {

113      ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(
114            bytearray);

116      try {
117         Entry newEntry = mApi.putFileOverwrite(fileName,
118               byteArrayInputStream, bytearray.length, null);
119         Log.i("DbExampleLog", "The uploaded file's rev is: " + newEntry.rev);

121      } catch (DropboxException e) {
122         e.printStackTrace();
123      } finally {
124         try {
125            byteArrayInputStream.close();
126         } catch (IOException e) {
127            System.out.println("Stream already closed");
128         }
129      }
130   }

132   static public void deleteFile(String fileName){
133      try {
134         mApi.delete(fileName);
135      } catch (DropboxException e) {
136         e.printStackTrace();
137      }
138   }

140 }
```

B.3. LocalFileListActivity.java

```
1  package de.sembo;

3  import java.io.File;
4  import java.util.ArrayList;

6  import android.app.Activity;
7  import android.content.Intent;
8  import android.graphics.Color;
9  import android.os.Bundle;
10 import android.view.View;
11 import android.view.View.OnClickListener;
12 import android.widget.Button;
13 import android.widget.LinearLayout;

15 public class LocalFileListActivity extends Activity {

17    Button localFileListButton;
18    LinearLayout localFileListLayout;

20    @Override
21    public void onCreate(Bundle savedInstanceState) {
22       super.onCreate(savedInstanceState);
```

```
24      setContentView(R.layout.localfilelist);
25      localFileListButton = (Button) findViewById(R.id.refreshLocalFiles_button);
26      localFileListLayout = (LinearLayout) findViewById(R.id.linearLayoutLocalFiles);
27      localFileListButton.setOnClickListener(new OnClickListener() {
28         public void onClick(View v) {
29            getFilesList();
30         }
31      });
32   }

34   public void getFilesList(){
35      localFileListLayout.removeAllViews();
36      ArrayList<String> remoteFileNames = DropBoxClient.getRemoteListing();

38      String place = LogInActivity.filePlaceFolder;
39      File dir = new File(place);
40      if (dir.exists()) {
41         System.out.println(dir.getAbsolutePath());
42         File[] fileList = dir.listFiles();
43         for (File file : fileList) {
44            boolean fileOnDropBox = false;
45            for (String name : remoteFileNames) {
46               if(name.contains(LogInActivity.myFilesPrefix+file.getName()) && !name.endsWith(".aes")){
47                  fileOnDropBox = true;
48                  break;
49               }
50            }

52            Button fileNameButton = new Button(getBaseContext());
53            fileNameButton.setText(file.getAbsolutePath());
54            fileNameButton.setTextColor(Color.WHITE);
55            fileNameButton.setBackgroundColor(Color.RED);

57            if(!fileOnDropBox){
58               fileNameButton.setBackgroundColor(Color.TRANSPARENT);
59               fileNameButton
60               .setOnClickListener(new OnClickListener4Upload(
61                     file.getName(), this));
62            }
63            localFileListLayout.addView(fileNameButton);
64         }
65      }
66   }

68   class OnClickListener4Upload implements OnClickListener {

70      String fileName;
71      LocalFileListActivity a;

73      public OnClickListener4Upload(String fileName, LocalFileListActivity a) {
74         this.fileName = fileName;
75         this.a = a;
76      }

78      @Override
79      public void onClick(View v) {
80         System.out.println("file is:" + fileName);
81         System.out.println("runUpload");
82         Intent intent = new Intent(a, Service.class);
83         intent.putExtra("FILENAME", fileName);
84         intent.putExtra("DIRECTION", "Upload");
85         a.startService(intent);
86         a.finish();
87      }
88   }

90 }
```

B.4. RemoteFileListActivity.java

```
1  package de.sembo;

3  import java.io.File;
4  import java.util.ArrayList;

6  import android.app.Activity;
7  import android.content.Intent;
8  import android.graphics.Color;
9  import android.os.Bundle;
10 import android.view.View;
11 import android.view.View.OnClickListener;
12 import android.widget.Button;
13 import android.widget.LinearLayout;

15 public class RemoteFileListActivity extends Activity{

17   Button remoteFileListButton;
18   LinearLayout remoteFileListLayout;

20   @Override
21   public void onCreate(Bundle savedInstanceState) {
22     super.onCreate(savedInstanceState);

24     setContentView(R.layout.remotefilelist);
25     remoteFileListButton = (Button) findViewById(R.id.refreshRemoteFiles_button);
26     remoteFileListLayout = (LinearLayout) findViewById(R.id.linearLayoutRemoteFiles);

28     remoteFileListButton.setOnClickListener(new OnClickListener() {
29       public void onClick(View v) {
30         getFilesList();
31       }
32     });
33   }

35   private void getFilesList() {

37     remoteFileListLayout.removeAllViews();
38     ArrayList<String> fileNames = DropBoxClient.getRemoteListing();

40     for (String string : fileNames) {
41       if(string.endsWith(LogInActivity.rsaSuffix)){
42         String nameToShow = "";

44         Button fileNameButton = new Button(getBaseContext());
45         String[] parsed = string.split("\\.");
46         if(parsed[1] != null && parsed[1].equalsIgnoreCase(LogInActivity.myID)){

48           //Endung und prefixe abschneiden
49           for (int i = 2; i < parsed.length - 1; i++) {
50             if (i == parsed.length - 2) {
51               nameToShow += parsed[i];
52             } else {
53               nameToShow += (parsed[i] + ".");
54             }
55           }
56           //gucken, ob es die Datei schon bei mir gibt
57           //wenn ja, listener(Button) abschalten
58           fileNameButton.setText(nameToShow);
59           fileNameButton.setBackgroundColor(Color.RED);
60           fileNameButton.setTextColor(Color.WHITE);

62           File f = new File(LogInActivity.filePlace + nameToShow);
63           if(!f.exists()){
64             fileNameButton.setBackgroundColor(Color.TRANSPARENT);
65             fileNameButton
```

```
66              .setOnClickListener(new OnClickListener4Download(
67                   string, this));
68           }
69           remoteFileListLayout.addView(fileNameButton);
70         }
71       }
72     }
73   }

75   class OnClickListener4Download implements OnClickListener {

77     String fileName;
78     RemoteFileListActivity a;

80     public OnClickListener4Download(String fileName, RemoteFileListActivity a) {
81       this.fileName = fileName;
82       this.a = a;
83     }

85     @Override
86     public void onClick(View v) {
87       System.out.println("file is:" + fileName);
88       System.out.println("runDownload");
89       Intent intent = new Intent(a, Service.class);
90       intent.putExtra("FILENAME", fileName);
91       intent.putExtra("DIRECTION", "Download");
92       a.startService(intent);
93       a.finish();
94     }
95   }
96 }
```

B.5. UserListActivity.java

```
1  package de.sembo;

3  import java.util.ArrayList;

5  import android.app.Activity;
6  import android.content.Intent;
7  import android.graphics.Color;
8  import android.os.Bundle;
9  import android.view.View;
10 import android.view.View.OnClickListener;
11 import android.widget.Button;
12 import android.widget.LinearLayout;

14 public class UserListActivity extends Activity {

16   private Button refreshUserListButton;
17   private LinearLayout userListLayout;
18   UserListActivity ac = this;

20   @Override
21   public void onCreate(Bundle savedInstanceState) {
22     super.onCreate(savedInstanceState);

24     setContentView(R.layout.userlist);
25     refreshUserListButton = (Button) findViewById(R.id.refresh_button);
26     userListLayout = (LinearLayout) findViewById(R.id.linearLayoutUsers);

28     refreshUserListButton.setOnClickListener(new OnClickListener() {
29       public void onClick(View v) {

31         userListLayout.removeAllViews();
```

```
33            // all users list holen
34            ArrayList<String> allUsers = DataBaseClient.getUserList();
35            //buttons aufbauen
36            for (String string : allUsers) {
37               Button userNameButton = new Button(getBaseContext());
38               userNameButton.setText(string);
39               userNameButton.setTextColor(Color.WHITE);
40               userNameButton.setBackgroundColor(Color.DKGRAY);
41               userNameButton
42                     .setOnClickListener(new OnClickListener4Share(ac));
43               userListLayout.addView(userNameButton);
44            }

46            // shared List holen
47            ArrayList<String> sharedUsers = DataBaseClient.getSharedUserList();
48            //farbe fuer erlaubte User gelb setzen
49            for (int i = 0; i < userListLayout.getChildCount(); i++) {
50               Button button = (Button) userListLayout.getChildAt(i);
51               for (String string : sharedUsers) {
52                  if (string.equalsIgnoreCase((String) button.getText())) {
53                     button.setTextColor(Color.YELLOW);
54                  }
55               }
56            }
57         }
58      });
59   }

61 }

63 class OnClickListener4Share implements OnClickListener {

65   UserListActivity a;

67   public OnClickListener4Share(UserListActivity a) {
68      this.a = a;
69   }

71   @Override
72   public void onClick(View v) {
73      Button b = (Button) v;
74      String answer;
75      String guestID = (String) b.getText();

77      answer = DataBaseClient.addOrRemoveToMySharedUsers(guestID);

79         if (answer.equalsIgnoreCase("user added to shared list")) {
80            //Service starten mit ANweisung aes schluessel fuer den
81            // gerade hinzugefuegten User zur verfuegung zu stellen
82            Intent intent = new Intent(a, Service.class);
83            intent.putExtra("SHARE", "Yes");
84            intent.putExtra("USER", guestID);
85            a.startService(intent);
86            b.setTextColor(Color.YELLOW);
87         } else if (answer.equalsIgnoreCase("user removed from shared list")) {
88            //Service starten mit ANweisung aes schluessel
89            // zu loeschen
90            Intent intent = new Intent(a, Service.class);
91            intent.putExtra("SHARE", "No");
92            intent.putExtra("USER", guestID);
93            a.startService(intent);
94            b.setTextColor(Color.WHITE);
95         } else {
96            b.setTextColor(Color.RED);
97         }
98   }
99 }
```

B.6. EncodedInput.java

```java
package de.sembo;

import java.io.InputStream;

public class EncodedInput {

  byte[] keyAsByteArray;
  InputStream inputStream;

  public EncodedInput(byte[] keyAsByteArray, InputStream inputStream){
    this.keyAsByteArray = keyAsByteArray;
    this.inputStream = inputStream;
  }

}
```

B.7. Crypter.java

```java
package de.sembo;

import java.io.InputStream;
import java.io.OutputStream;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Security;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

public class Crypter {

  static {
    Security.addProvider(new org.spongycastle.jce.provider.BouncyCastleProvider());
  }
  static String algorithmName = "AES/ECB/PKCS5Padding";
  static String provider = "SC";
  static String codeTypeSymm = "AES";
  static String codeTypeAsymm = "RSA";
  static PublicKey publicKey;
  static PrivateKey privateKey;

  private Crypter() {

  }

  public static EncodedInput getEncodedInput(InputStream inputStream) {
    System.out.println("Set AES Cipher");
```

```
46      EncodedInput encodedInput = null;
47      try {
48         Cipher cipher = Cipher.getInstance(algorithmName, provider);
49         KeyGenerator keygen = KeyGenerator.getInstance(codeTypeSymm,
50               provider);

52         System.out.println("Generate AES Key");
53         // SecureRandom random = new SecureRandom();
54         // keygen.init(random);
55         keygen.init(128);
56         Key key = keygen.generateKey();
57         System.out.println("algorithm: " + key.getAlgorithm());
58         cipher.init(Cipher.ENCRYPT_MODE, key);
59         System.out.println("OK AES Cipher");
60         encodedInput = new EncodedInput(key.getEncoded(),
61               new CipherInputStream(inputStream, cipher));

63      } catch (NoSuchAlgorithmException e) {
64         e.printStackTrace();
65      } catch (NoSuchPaddingException e) {
66         e.printStackTrace();
67      } catch (InvalidKeyException e) {
68         e.printStackTrace();
69      } catch (NoSuchProviderException e) {
70         e.printStackTrace();
71      }

73      return encodedInput;
74   }

76   static public OutputStream getDecodedOutputStream(byte[] aeskey,
77         OutputStream os) {
78      System.out.println("getDecodedOutputStream()");

80      Key key = new SecretKeySpec(aeskey, codeTypeSymm);
81      CipherOutputStream cipherOutputStream = null;

83      try {
84         Cipher cipher;
85         cipher = Cipher.getInstance(algorithmName, provider);
86         cipher.init(Cipher.DECRYPT_MODE, key);
87         cipherOutputStream = new CipherOutputStream(os, cipher);
88      } catch (NoSuchAlgorithmException e) {
89         e.printStackTrace();
90      } catch (NoSuchProviderException e) {
91         e.printStackTrace();
92      } catch (NoSuchPaddingException e) {
93         e.printStackTrace();
94      } catch (InvalidKeyException e) {
95         e.printStackTrace();
96      }

98      return cipherOutputStream;
99   }

101  static public byte[] cryptAsymmetric(byte[] data,
102        byte[] rsaKey, String publicORprivate) {
103     System.out
104           .println("Tryin Asymetric crypt: " + publicORprivate);

106     byte[] bytearray = new byte[0];
107     try {
108        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

110        if (publicORprivate.equalsIgnoreCase("public")) {
111           Key key = initRSAPublicFromBytes(rsaKey);
112           cipher.init(Cipher.ENCRYPT_MODE, key);
113           byte[] encryptedData = cipher.doFinal(data);
```

```
114             System.out.println("aes key encrypted with public rsa key");
115             return encryptedData;

117          } else if (publicORprivate.equalsIgnoreCase("private")) {
118             Key key = initRSAPrivateFromBytes(rsaKey);
119             cipher.init(Cipher.DECRYPT_MODE, key);
120             byte[] decryptedData = cipher.doFinal(data);
121             System.out.println("aes key decrypted with private rsa key");
122             return decryptedData;

124          } else {
125             System.out.println("publicORprivate : nicht festgelegt");
126          }

128       } catch (NoSuchAlgorithmException e) {
129          e.printStackTrace();
130       } catch (NoSuchPaddingException e) {
131          e.printStackTrace();
132       } catch (InvalidKeyException e) {
133          e.printStackTrace();
134       } catch (IllegalBlockSizeException e) {
135          e.printStackTrace();
136       } catch (BadPaddingException e) {
137          e.printStackTrace();
138       }
139       return bytearray;

141    }

143    static public void createRSAKeys() {

145       System.out.println("create rsa keys");
146       KeyPairGenerator keyPairGen;
147       KeyPair keyPair;
148       try {
149          keyPairGen = KeyPairGenerator.getInstance(codeTypeAsymm, provider);
150          keyPairGen.initialize(1024);
151          keyPair = keyPairGen.generateKeyPair();
152          privateKey = keyPair.getPrivate();
153          publicKey = keyPair.getPublic();

155       } catch (NoSuchAlgorithmException e) {
156          e.printStackTrace();
157       } catch (NoSuchProviderException e) {
158          e.printStackTrace();
159       }
160    }

162    static private PublicKey initRSAPublicFromBytes(byte[] data){
163       System.out.println("initRSAPublicFromBytes()");
164       PublicKey key = null;
165       try {
166          key = KeyFactory.getInstance("RSA").generatePublic(
167                new X509EncodedKeySpec(data));
168       } catch (InvalidKeySpecException e) {
169          e.printStackTrace();
170       } catch (NoSuchAlgorithmException e) {
171          e.printStackTrace();
172       }
173       return key;
174    }

176    static private PrivateKey initRSAPrivateFromBytes(byte[] data){
177       System.out.println("initRSAPrivateFromBytes()");
178       KeyFactory kf;
179       PrivateKey key = null;
180       try {
181          kf = KeyFactory.getInstance("RSA");
```

```
182         PKCS8EncodedKeySpec ks = new PKCS8EncodedKeySpec(data);
183         key = kf.generatePrivate(ks);
184      } catch (NoSuchAlgorithmException e) {
185         e.printStackTrace();
186      } catch (InvalidKeySpecException e) {
187         e.printStackTrace();
188      }
189      return key;
190   }

192   static public void setMyPrivateKeyFromBytes(byte[] data){
193      privateKey = initRSAPrivateFromBytes(data);
194   }

196   static public void setMyPublicKeyFromBytes(byte[] data){
197      publicKey = initRSAPublicFromBytes(data);
198   }

200 }
```

B.8. Service.java

```
1  package de.sembo;

3  import java.io.BufferedOutputStream;
4  import java.io.ByteArrayInputStream;
5  import java.io.File;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.util.ArrayList;

14 import android.app.IntentService;
15 import android.app.Notification;
16 import android.app.NotificationManager;
17 import android.app.PendingIntent;
18 import android.content.Intent;
19 import android.os.Bundle;
20 import android.os.Environment;
21 import android.util.Log;

23 import com.dropbox.client2.exception.DropboxException;

25 public class Service extends IntentService {

27    public Service() {
28       super("Service");
29       System.out.println("myID in Service: " + LogInActivity.myID);
30    }

32    @Override
33    protected void onHandleIntent(Intent intent) {
34       System.out.println("Service Started");
35       System.out.println("myID in Service: " + LogInActivity.myID);

37       final Intent intentLocal = intent;
38       final Bundle extras = intentLocal.getExtras();

40       if (extras != null) {
41          String fileName = extras.getString("FILENAME");
42          String direction = extras.getString("DIRECTION");
43          String share = extras.getString("SHARE");
44          String userName = extras.getString("USER");
```

```
46        if (direction != null && fileName != null) {

48           if (direction.equalsIgnoreCase("Download")) {
49              downloadFile(fileName);
50           } else if (direction.equalsIgnoreCase("Upload")) {
51              uploadFile(fileName);
52           } else {
53              Log.e("Service Intent", "Unknown Command");
54           }
55           createNotification(direction + ": " + fileName);
56        }

58        if (share != null && userName != null) {
59           String user = userName;
60           if (share.equalsIgnoreCase("Yes")) {
61              addShares4User(user);
62           } else if (share.equalsIgnoreCase("No")) {
63              deleteShares4User(user);
64           }
65           createNotification(share + ": " + user);
66        }
67     }
68  }

70  private void createNotification(String msg) {

72     Notification notification = new Notification(R.drawable.ic_launcher, "",
73           System.currentTimeMillis());
74     NotificationManager mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
75     String tickerText = "Service: finished";

77     notification.flags |= Notification.FLAG_AUTO_CANCEL | Notification.FLAG_SHOW_LIGHTS;
78     // Notification notification = new Notification(R.drawable.ic_launcher,
79     // tickerText, System.currentTimeMillis());
80     // notification.flags |= Notification.FLAG_AUTO_CANCEL;
81     // notification.number += 1;
82     notification.tickerText = tickerText;
83     // Context context = getApplicationContext();

85     Intent notificationIntent = new Intent(this, this.getClass());
86     PendingIntent contentIntent = PendingIntent.getActivity(this,
87           (int) System.currentTimeMillis(), notificationIntent, 0);

89     notification.setLatestEventInfo(this, "BAcleintService", msg,
90           contentIntent);
91     mNotificationManager.notify(0, notification);
92  }

94  private void addShares4User(String guestID) {
95     System.out.println("addShares4User");

97     String userFilesPrefix = guestID + ".";
98     byte[] publicKeyAsBytes = new byte[0];

100    // public key des user holen
101    publicKeyAsBytes = DataBaseClient.getPublicKeyFor(guestID);

103    // alle AES keys von meinen dateien holen
104    ArrayList<String> fileNames = DropBoxClient.getRemoteListing();
105    ArrayList<String> aesKeysFileNames = new ArrayList<String>();
106    for (String string : fileNames) {
107       if (string.startsWith("" + LogInActivity.myFilesPrefix
108             + LogInActivity.myFilesPrefix)
109             && string.endsWith(LogInActivity.rsaSuffix)) {
110          aesKeysFileNames.add(string);
111       }
112    }
```

```
114       for (String string : aesKeysFileNames) {
115          //key runterladen
116          byte[] data = DropBoxClient.downloadByteArray(string);
117          //Dateinamen UserUnabhaengig machen
118          String[] tempStrings = string.split("\\.");
119          String fileName = "";
120          // prefixe und suffix erstma abschneiden
121          for (int i = 2; i < tempStrings.length - 1; i++) {
122             if (i == tempStrings.length - 2) {
123                fileName += tempStrings[i];
124             } else {
125                fileName += (tempStrings[i] + ".");
126             }
127          }

129          // aes schluessel mir rsa entschluesseln
130          data = Crypter.cryptAsymmetric(data,
131                Crypter.privateKey.getEncoded(), "private");

133          // aes schluessel mit public key des users verschluesseln
134          data = Crypter.cryptAsymmetric(data,
135                publicKeyAsBytes, "public");
136          if (data.length != 0) {
137             String fileNameOnDBox = LogInActivity.myFilesPrefix
138                   + userFilesPrefix + fileName + LogInActivity.rsaSuffix;

140 //          Utils.uploadByteArray(data, fileNameOnDBox);
141             InputStream is = new ByteArrayInputStream(data);
142             DropBoxClient.uploadInputStream(fileNameOnDBox, is, data.length);

144             System.out.println("RSA hochgeladen: "+ guestID);
145          } else {
146             System.out.println
147                   .println("RSA Fehler: " + guestID);
148          }
149       }
150    }

152    private void deleteShares4User(String guestID) {

154       String userFilesPrefix = guestID + ".";
155       ArrayList<String> fileNames = DropBoxClient.getRemoteListing();
156       for (String string : fileNames) {
157          if (string.startsWith("" + LogInActivity.myFilesPrefix
158                + userFilesPrefix)
159                && string.endsWith(LogInActivity.rsaSuffix)) {
160             System.out.println("trying to delete: " + string);
161             DropBoxClient.deleteFile(string);
162          }
163       }
164    }

166    private void uploadFile(String fileName) {
167       // Uploading content.
168       FileInputStream fileInputStream = null;
169       EncodedInput encodedInput = null;
170       ByteArrayInputStream keyInputStream = null;

172       String state = Environment.getExternalStorageState();
173       if (Environment.MEDIA_MOUNTED.equals(state)) {
174          // SDcard is available

176          File f = new File(LogInActivity.filePlace + fileName);
177          System.out.println("Trying to Upload: " + f.getAbsolutePath());
178          try {

180             fileInputStream = new FileInputStream(f);
```

```
181 //          MessageDigest md = MessageDigest.getInstance("SHA");
182 //          DigestInputStream digestInputStream = new DigestInputStream(
183 //                fileInputStream, md);
184             encodedInput = Crypter.getEncodedInput(
185                   fileInputStream);

187             // Blockgroesse fuer AES beruecksichtigen
188             long filePaddedSize = (16 - (f.length() % 16)) + f.length();
189             String fileNameOnDBox = LogInActivity.myFilesPrefix
190                   + LogInActivity.myFilesPrefix + fileName
191                   + LogInActivity.aesSuffix;
192             DropBoxClient.uploadInputStream(fileNameOnDBox, encodedInput.inputStream, filePaddedSize);
193             System.out.println("rsa encrypt for me...");
194             byte[] encryptedData = Crypter.cryptAsymmetric(encodedInput.keyAsByteArray, Crypter.publicKey.getEncoded(), "public");

196             if (encryptedData.length != 0) {
197                fileNameOnDBox = LogInActivity.myFilesPrefix
198                      + LogInActivity.myFilesPrefix + fileName
199                      + LogInActivity.rsaSuffix;
200                InputStream is = new ByteArrayInputStream(encryptedData);
201                DropBoxClient.uploadInputStream(fileNameOnDBox, is, encryptedData.length);
202                try {
203                   is.close();
204                } catch (IOException e) {
205                   e.printStackTrace();
206                }
207             } else {
208                System.out
209                      .println("bei Asymetric was schiefgelaufen");
210             }

212             //fuer andere shared user freigeben
213             ArrayList<String> sharedUsers = new ArrayList<String>();
214             System.out.println("arrayList groesse: "+ sharedUsers.size());
215             sharedUsers = DataBaseClient.getSharedUserList();
216             for (String userID : sharedUsers) {
217                //public key des Users holen
218                byte[] publicKey = DataBaseClient.getPublicKeyFor(userID);
219                System.out.println("rsa encrypt for: "+userID+ "...");
220                encryptedData = Crypter.cryptAsymmetric(encodedInput.keyAsByteArray, publicKey, "public");
221                if (encryptedData.length != 0) {
222                   fileNameOnDBox = LogInActivity.myFilesPrefix
223                         + userID+"." + fileName
224                         + LogInActivity.rsaSuffix;
225                   InputStream is = new ByteArrayInputStream(encryptedData);
226                   DropBoxClient.uploadInputStream(fileNameOnDBox, is, encryptedData.length);
227                   try {
228                      is.close();
229                   } catch (IOException e) {
230                      e.printStackTrace();
231                   }
232                } else {
233                   System.out
234                         .println("bei Asymetric was schiefgelaufen");
235                }
236             }
237          } catch (FileNotFoundException e) {
238             Log.e("DbExampleLog", "File not found.");
239          } finally {
240             try {
241                if (keyInputStream != null) {
242                   keyInputStream.close();
243                }
244                if (encodedInput.inputStream != null) {
245                   encodedInput.inputStream.close();
246                }

248             } catch (IOException e) {
```

```
249             Log.e("Service", "stream close error");
250          }
251       }
252    } else {
253       Log.e("Error", "No SD-CARD");
254    }
255 }

257 private void downloadFile(String fileName) {

259    System.out.println("downloadFile()" + fileName);
260    String state = Environment.getExternalStorageState();
261    String[] fileNameReal = fileName.split("\\.");

263    String fileName2Save = "";
264    String fileName2Dl = "";
265    // Endung und prefixe abschneiden
266    for (int i = 2; i < fileNameReal.length - 1; i++) {
267       if (i == fileNameReal.length - 2) {
268          fileName2Save += fileNameReal[i];
269       } else {
270          fileName2Save += (fileNameReal[i] + ".");
271       }
272    }
273    String aesFileName = fileNameReal[0] +"."+ fileNameReal[0]+"." + fileName2Save + LogInActivity.aesSuffix;
274    System.out.println("save file to: " + fileName2Save);

276    fileNameReal = fileName.split(LogInActivity.rsaSuffix);
277    fileName2Dl = fileNameReal[0];
278    System.out.println("get file: " + fileName2Dl);

280    if (Environment.MEDIA_MOUNTED.equals(state)) {
281       // SDcard is available
282       String localFileName = LogInActivity.filePlace + fileName2Save;
283 //    aesFileName = fileName2Dl + LogInActivity.aesSuffix;
284       String keyFileName = fileName2Dl + LogInActivity.rsaSuffix;

286       // mit rsa verschluesseltes aes schluessel runterladen
287       byte[] data = DropBoxClient.downloadByteArray(keyFileName);

289       // aes schluessel mir rsa entschluesseln
290       data = Crypter.cryptAsymmetric(data, Crypter.privateKey.getEncoded(), "private");

292       File f = new File(localFileName);
293       try {
294          OutputStream outputStream = new BufferedOutputStream(
295                new FileOutputStream(f));
296          OutputStream cipherOutputStream = Crypter
297                .getDecodedOutputStream(data, outputStream);
298          boolean ok = false;
299          ok = DropBoxClient
300                .downloadOutputStream(aesFileName, cipherOutputStream);
301          if (!ok) {
302             f.delete();
303          }

305       } catch (FileNotFoundException e) {
306          e.printStackTrace();
307       }
308    } else {
309       Log.e("Error", "No SD-CARD");
310    }
311 }

313 }
```

B.9. LogInActivity.java

```
package de.sembo;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.text.Editable;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;
import com.dropbox.client2.android.AndroidAuthSession;
import com.dropbox.client2.android.AuthActivity;
import com.dropbox.client2.session.AccessTokenPair;
import com.dropbox.client2.session.AppKeyPair;
import com.dropbox.client2.session.Session.AccessType;
import com.dropbox.client2.session.TokenPair;


public class LogInActivity extends Activity {

  private static final String TAG = "LogInActivity";

  private boolean mLoggedIn;
  private Button mSubmit;

  static String aesSuffix = ".aes";
  static String rsaSuffix = ".rsa";

  private Button localFileListButton;
  private Button remoteFileListButton;

  private LinearLayout layoutID;
  private LinearLayout layoutButtons;
  private Button userListButton;
  static String fileFolderName = "Files";
  static String appFolder = "BAclient";
  static String filePlaceFolder = "/sdcard/" + appFolder + "/" + fileFolderName;
  static String filePlace = filePlaceFolder + "/";
  static String IDfileName = "BAclientMyIdentity.txt";
  static String myID;
```

```
66 	static String myFilesPrefix = myID + ".";

68 	@Override
69 	public void onCreate(Bundle savedInstanceState) {
70 		super.onCreate(savedInstanceState);
71 		System.out.println("suffix:" + aesSuffix);

73 		File dir = new File("/sdcard/" + appFolder);
74 		if (!dir.exists()) {
75 			dir.mkdir();
76 			System.out.println("create dir: " + dir.getAbsolutePath());
77 		}
78 		File dir2 = new File(filePlaceFolder);
79 		if (!dir2.exists() && dir.exists()) {
80 			dir2.mkdir();
81 			System.out.println("create dir: " + dir2.getAbsolutePath());
82 		}

84 		File fi = new File("/sdcard/" + appFolder + "/"+ "tcp.txt");
85 		BufferedReader inn;
86 		try {
87 			inn = new BufferedReader(new FileReader(fi));
88 			String line;
89 			// wenn datei nicht leer
90 			while ((line = inn.readLine()) != null) {
91 				String[] temp = line.split(":");
92 				if(temp[0].equalsIgnoreCase("ipAddress")){
93 					DataBaseClient.ipAddress = temp[1];
94 				}
95 				if(temp[0].equalsIgnoreCase("ipPort")){
96 					DataBaseClient.ipPort = Integer.parseInt(temp[1]);
97 				}
98 			}
99 			inn.close();
100 		} catch (FileNotFoundException e) {
101 			createNotification(fi.getAbsolutePath() + " not found", "error");
102 			finish();

104 			e.printStackTrace();
105 		} catch (IOException e) {
106 			e.printStackTrace();
107 		}

109 		// We create a new AuthSession so that we can use the Dropbox API.
110 		AndroidAuthSession session = buildSession();
111 		DropBoxClient.mApi = new DropboxAPI<AndroidAuthSession>(session);

113 		setContentView(R.layout.main);
114 		layoutID = (LinearLayout) findViewById(R.id.linearLayoutID);
115 		layoutButtons = (LinearLayout) findViewById(R.id.linearLayoutButtons);

117 		// Gucken, ob mit diesem Cleint schon ein user registriert ist
118 		File file = new File("/sdcard/" + appFolder + "/" + IDfileName);
119 		if (!file.exists()) {
120 			System.out.println("IDDatei existiert nicht");

122 			OnKeyListener l = new OnKeyListener() {
123 				@Override
124 				public boolean onKey(View v, int keyCode, KeyEvent event) {
125 					if ((event.getAction() == KeyEvent.ACTION_DOWN)
126 							&& (keyCode == KeyEvent.KEYCODE_ENTER)) {

128 						EditText et = (EditText) v;
129 						Editable ed = et.getText();
130 						LogInActivity.myID = ed.toString();
131 						LogInActivity.myFilesPrefix = LogInActivity.myID + ".";
132 						System.out.println("myID is:" + LogInActivity.myID);
```

```
134                 BufferedWriter bw = null;
135                 File fil = new File("/sdcard/" + appFolder + "/"
136                       + IDfileName);
137                 FileWriter fw;
138                 try {
139                    fw = new FileWriter(fil);
140                    bw = new BufferedWriter(fw);
141                    bw.write(LogInActivity.myID);
142                    bw.close();
143                    LinearLayout lo = (LinearLayout) v.getParent();
144                    lo.removeAllViews();
145                    continueInit();
146                 } catch (IOException e) {
147                    // TODO Auto-generated catch block
148                    e.printStackTrace();
149                 }
150                 return true;
151              }
152              return false;
153           }
154        };

156        EditText box = new EditText(getBaseContext());
157        box.setOnKeyListener(l);
158        layoutID.addView(box);

160     } else {
161        System.out.println("IDDatei existiert");

163        BufferedReader in;
164        try {
165           in = new BufferedReader(new FileReader(file));
166           String line;
167           // wenn datei nicht leer
168           if ((line = in.readLine()) != null) {
169              LogInActivity.myID = line;
170              LogInActivity.myFilesPrefix = LogInActivity.myID + ".";
171              System.out.println("myID:" + LogInActivity.myID);
172           }
173           in.close();
174        } catch (FileNotFoundException e) {

176           e.printStackTrace();
177        } catch (IOException e) {
178           e.printStackTrace();
179        }

181        continueInit();
182     }

184  }

186  @Override
187  protected void onResume() {
188     super.onResume();
189     AndroidAuthSession session = DropBoxClient.mApi.getSession();

191     // The next part must be inserted in the onResume() method of the
192     // activity from which session.startAuthentication() was called, so
193     // that Dropbox authentication completes properly.
194     if (session.authenticationSuccessful()) {
195        try {
196           // Mandatory call to complete the auth
197           session.finishAuthentication();

199           // Store it locally in our app for later use
200           TokenPair tokens = session.getAccessTokenPair();
201           storeKeys(tokens.key, tokens.secret);
```

```
202            setLoggedIn(true);
203         } catch (IllegalStateException e) {
204            showToast("Couldn't authenticate with Dropbox:"
205                  + e.getLocalizedMessage());
206            Log.i(TAG, "Error authenticating", e);
207         }
208      }
209   }

211   // ########################################################################

213   // ########################################################################

215   private AndroidAuthSession buildSession() {
216      AppKeyPair appKeyPair = new AppKeyPair(DropBoxClient.APP_KEY, DropBoxClient.APP_SECRET);
217      AndroidAuthSession session;

219      String[] stored = getKeys();
220      if (stored != null) {
221         AccessTokenPair accessToken = new AccessTokenPair(stored[0],
222               stored[1]);
223         session = new AndroidAuthSession(appKeyPair, DropBoxClient.ACCESS_TYPE,
224               accessToken);
225      } else {
226         session = new AndroidAuthSession(appKeyPair, DropBoxClient.ACCESS_TYPE);
227      }
228      return session;
229   }

231   /**
232    * Shows keeping the access keys returned from Trusted Authenticator in a
233    * local store, rather than storing user name & password, and
234    * re-authenticating each time (which is not to be done, ever).
235    *
236    * @return Array of [access_key, access_secret], or null if none stored
237    */
238   private String[] getKeys() {
239      SharedPreferences prefs = getSharedPreferences(DropBoxClient.ACCOUNT_PREFS_NAME,
240            MODE_WORLD_READABLE);
241      String key = prefs.getString(DropBoxClient.ACCESS_KEY_NAME, null);
242      String secret = prefs.getString(DropBoxClient.ACCESS_SECRET_NAME, null);
243      if (key != null && secret != null) {
244         String[] ret = new String[2];
245         ret[0] = key;
246         ret[1] = secret;
247         return ret;
248      } else {
249         return null;
250      }
251   }

253   /**
254    * Shows keeping the access keys returned from Trusted Authenticator in a
255    * local store, rather than storing user name & password, and
256    * re-authenticating each time (which is not to be done, ever).
257    */
258   private void storeKeys(String key, String secret) {
259      // Save the access key for later
260      SharedPreferences prefs = getApplicationContext().getSharedPreferences(
261            DropBoxClient.ACCOUNT_PREFS_NAME, MODE_WORLD_READABLE);
262      Editor edit = prefs.edit();
263      edit.putString(DropBoxClient.ACCESS_KEY_NAME, key);
264      edit.putString(DropBoxClient.ACCESS_SECRET_NAME, secret);
265      edit.commit();
266   }

268   private void clearKeys() {
269      SharedPreferences prefs = getSharedPreferences(DropBoxClient.ACCOUNT_PREFS_NAME,
```

```
270            MODE_WORLD_READABLE);
271      Editor edit = prefs.edit();
272      edit.clear();
273      edit.commit();
274   }

276   private void checkAppKeySetup() {
277      // Check to make sure that we have a valid app key
278      if (DropBoxClient.APP_KEY.startsWith("CHANGE") || DropBoxClient.APP_SECRET.startsWith("CHANGE")) {
279         showToast("You must apply for an app key and secret from developers.dropbox.com, and add them to the DBRoulette ap before
                 trying it.");
280         finish();
281         return;
282      }

284      // Check if the app has set up its manifest properly.
285      Intent testIntent = new Intent(Intent.ACTION_VIEW);
286      String scheme = "db-" + DropBoxClient.APP_KEY;
287      String uri = scheme + "://" + AuthActivity.AUTH_VERSION + "/test";
288      testIntent.setData(Uri.parse(uri));
289      PackageManager pm = getPackageManager();
290      if (0 == pm.queryIntentActivities(testIntent, 0).size()) {
291         showToast("URL scheme in your app's "
292               + "manifest is not set up correctly. You should have a "
293               + "com.dropbox.client2.android.AuthActivity with the "
294               + "scheme: " + scheme);
295         finish();
296      }
297   }

299   private void showToast(String msg) {
300      Toast error = Toast.makeText(this, msg, Toast.LENGTH_LONG);
301      error.show();
302   }

304   private void logOut() {
305      // Remove credentials from the session
306      DropBoxClient.mApi.getSession().unlink();

308      // Clear our stored keys
309      clearKeys();
310      // Change UI state to display logged out version
311      setLoggedIn(false);
312   }

314   /**
315    * Convenience function to change UI state based on being logged in
316    */
317   private void setLoggedIn(boolean loggedIn) {
318      mLoggedIn = loggedIn;
319      if (loggedIn) {
320         mSubmit.setBackgroundColor(Color.GREEN);
321         mSubmit.setText("Unlink from Dropbox");
322         // mDisplay.setVisibility(View.VISIBLE);
323      } else {
324         mSubmit.setBackgroundColor(Color.RED);
325         mSubmit.setText("Link with Dropbox");
326         // mDisplay.setVisibility(View.GONE);
327         // mImage.setImageDrawable(null);
328      }
329   }

331   private void continueInit() {
332      checkAppKeySetup();
333      String place = "/sdcard/" + LogInActivity.appFolder + "/" + LogInActivity.myID;
334      File privateKeyFile = new File(place + ".privateKey");
335      File publicKeyFile = new File(place + ".publicKey");
336      if (!privateKeyFile.exists() || !publicKeyFile.exists()) {
```

```
337         Crypter.createRSAKeys();
338         // waehrend der Anmeldung am Server wird public key
339         // dahingeschikt
340         String answer = DataBaseClient.addUserToDB(LogInActivity.myID,
341               Crypter.publicKey.getEncoded());

343         if (answer.equalsIgnoreCase("UserAdded")) {
344            // wenn user erfolgreich in db gespeichert, wird auch local
345            // gespeichert
346            FileOutputStream fileOutputStream;
347            try {
348               fileOutputStream = new FileOutputStream(
349                     privateKeyFile);
350               fileOutputStream.write(Crypter.privateKey.getEncoded());
351               System.out.println("privateKey written");
352               fileOutputStream.close();
353               fileOutputStream = new FileOutputStream(publicKeyFile);
354               fileOutputStream.write(Crypter.publicKey.getEncoded());
355               System.out.println("publicKey written");
356               fileOutputStream.close();
357            } catch (FileNotFoundException e) {
358               e.printStackTrace();
359            } catch (IOException e) {
360               e.printStackTrace();
361            }
362         }

364      }else{
365         System.out.println("PrivateKey and PublicKey exist already!");
366         FileInputStream fileInputStream;
367         try {
368            fileInputStream = new FileInputStream(publicKeyFile);
369            byte[] data = new byte[(int) publicKeyFile.length()];
370            fileInputStream.read(data);
371            fileInputStream.close();
372            Crypter.setMyPublicKeyFromBytes(data);
373            System.out.println("public Key read");

375            fileInputStream = new FileInputStream(privateKeyFile);
376            data = new byte[(int) privateKeyFile.length()];
377            fileInputStream.read(data);
378            fileInputStream.close();
379            Crypter.setMyPrivateKeyFromBytes(data);
380            System.out.println("private Key read");

382         } catch (FileNotFoundException e) {
383            e.printStackTrace();
384         } catch (IOException e) {
385            e.printStackTrace();
386         }
387      }

389      mSubmit = new Button(getBaseContext());
390      layoutButtons.addView(mSubmit);
391      remoteFileListButton = new Button(getBaseContext());
392      remoteFileListButton.setText("Remote File List");
393      localFileListButton = new Button(getBaseContext());
394      localFileListButton.setText("Local File List");
395      layoutButtons.addView(remoteFileListButton);
396      layoutButtons.addView(localFileListButton);
397      userListButton = new Button(getBaseContext());
398      userListButton.setText("Remote User List");
399      layoutButtons.addView(userListButton);

401      final Intent userListActivityIntent = new Intent(this,
402            UserListActivity.class);
403      final Intent remoteFileListActivityIntent = new Intent(this,
404            RemoteFileListActivity.class);
```

```
405       final Intent localFileListActivityIntent = new Intent(this,
406             LocalFileListActivity.class);

408       mSubmit.setOnClickListener(new OnClickListener() {
409          public void onClick(View v) {
410             // This logs you out if you're logged in, or vice versa
411             if (mLoggedIn) {
412                logOut();
413             } else {
414                // Start the remote authentication
415                DropBoxClient.mApi.getSession().startAuthentication(LogInActivity.this);
416             }
417          }
418       });

420       // Display the proper UI state if logged in or not
421       setLoggedIn(DropBoxClient.mApi.getSession().isLinked());

423       userListButton.setOnClickListener(new OnClickListener() {
424          @Override
425          public void onClick(View v) {
426             startActivity(userListActivityIntent);
427          }
428       });

430       remoteFileListButton.setOnClickListener(new OnClickListener() {
431          @Override
432          public void onClick(View v) {
433             startActivity(remoteFileListActivityIntent);
434          }
435       });

437       localFileListButton.setOnClickListener(new OnClickListener() {
438          @Override
439          public void onClick(View v) {
440             startActivity(localFileListActivityIntent);
441          }
442       });
443    }

445    private void createNotification(String msg, String ticker) {

447       Notification notification = new Notification(R.drawable.ic_launcher, "",
448             System.currentTimeMillis());
449       NotificationManager mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
450       notification.flags |= Notification.FLAG_AUTO_CANCEL | Notification.FLAG_SHOW_LIGHTS;
451       notification.tickerText = ticker;
452       Intent notificationIntent = new Intent(this, this.getClass());
453       PendingIntent contentIntent = PendingIntent.getActivity(this,
454             (int) System.currentTimeMillis(), notificationIntent, 0);

456       notification.setLatestEventInfo(this, this.getTitle(), msg,
457             contentIntent);
458       mNotificationManager.notify(0, notification);
459    }
460 }
```

Literaturverzeichnis

[1] Acdx: *Digitale Signatur*. http://de.wikipedia.org/wiki/Digitale_Signatur, 2012. – Online Ressource, Abruf: 29.Juli 2012

[2] atvirtual.net: *Das Aussehen eines Zertifikates?* http://www.atvirtual.net/addon/ssl/faq.html, . – Online Ressource, Abruf: 29.Juli 2012

[3] Backes, Michael ; Cachin, Christian ; Oprea, Alina: Lazy Revocation in Cryptographic File Systems. In: *Proceedings of 3rd International IEEE Security in Storage Workshop (SISW)*, 2005, S. 1–11

[4] Borgmann, Moritz ; Hahn, Tobias ; Herfert, Michael ; Kunz, Thomas ; Richter, Marcel ; Viebeg, Ursula ; Vowé, Sven: On the Security of Cloud Storage Services / Fraunhofer Institute for Secure Information Technology SIT. Version: März 2012. http://www.sit.fraunhofer.de/content/dam/sit/en/studies/Cloud-Storage-Security_a4.pdf. Rheinstraße 75, 64295 Darmstadt, Germany, März 2012. (SIT Technical Reports). – Forschungsbericht

[5] bouncycastle.org: *The Legion of the Bouncy Castle*. http://www.bouncycastle.org/, . – Online Ressource, Abruf: 03.August 2012

[6] Eckert, C.: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg Wissenschaftsverlag, 2011 http://books.google.de/books?id=x5Y8psQsdnIC. – ISBN 9783486706871

[7] Elektronik-Kompendium.de: *Hybride Verschlüsselungsverfahren*. http://www.elektronik-kompendium.de/sites/net/0908071.htm, . – Online Ressource, Abruf: 30.Juli 2012

[8] Grupp, A.: *Einführung in Verschlüsselung*. https://www.elektronikschule.de/~grupp/verschluesselung/intro.html, . – Online Ressource, Abruf: 30.Juli 2012

[9] Hook, D.: *Beginning Cryptography with Java*. Wiley, 2005 (Wrox Beginning guides). http://books.google.de/books?id=WLLAD2FKH3IC. – ISBN 9780764596339

[10] Horster, P.: *Systemsicherheit: Grundlagen, Konzepte, Realisierungen, Anwendungen.* Vieweg, 2000 (DuD-Fachbeiträge). http://books.google.de/books?id=g7mrAAAACAAJ. – ISBN 9783528057459

[11] Inc., Dropbox: *Dropbox for Developers.* https://www.dropbox.com/developers, . – Online Ressource, Abruf: 05.August 2012

[12] rtyley: *Spongy Castle.* http://rtyley.github.com/spongycastle/, . – Online Ressource, Abruf: 03.August 2012

[13] Stolfi, J.: *Hash function.* http://en.wikipedia.org/wiki/Hash_function, . – Online Ressource, Abruf: 30.Juli 2012

[14] www.android.com: *Android Developers.* http://developer.android.com/index.html, . – Online Ressource, Abruf: 08.August 2012

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. August 2012 Ernst Mattern