Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andreas Rebri

Konzept zur sicheren Verarbeitung von Unternehmensdaten mit privaten Android-Smartphones

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Rebri

Konzept zur sicheren Verarbeitung von Unternehmensdaten mit privaten Android-Smartphones

Bachelorarbeit eingereicht im Rahmen 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. Hübner
Zweitgutachter : Prof. Dr. Sarstedt

Abgegeben am 27.06.2014

Andreas Rebri

Thema der Bachelorarbeit
Konzept zur sicheren Verarbeitung von Unternehmensdaten mit privaten Android-Smartphones

Stichworte
BYOD, Bring Your Own Device, Android, Sicherheit

Kurzzusammenfassung
Diese Arbeit handelt von einem Konzept und Entwurf zur Umsetzung einer Androidanwendung zur sicheren Datenverarbeitung. Die Smartphones sind hierbei private Geräte auf denen Unternehmensdaten sicher und vertraulich gespeichert und verarbeitet werden können.

Andreas Rebri

Title of the paper
Concept for secure data processing of company data with a non-corporate Android-Smartphone

Keywords
BYOD, Bring Your Own Device, Android, Security

Abstract
This work is about the concept and design of an application for secure data processing on an Android Smartphone. The smartphone is a non-corporate device where corporate data is securely saved and processed.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Smartphones sind ein wichtiger Bestandteil unseres Lebens. Sie sind heutzutage weitaus mehr als nur ein Telefon. Man kann sie als Navigation, zum Surfen im Internet, als Organisator/Kalender nutzen und vieles mehr. Es gibt für fast alles Apps(Anwendungen), wodurch das Smartphone immer mehr zu einem Ersatz für einen vollwertigen Rechner wird. 2010 überstiegen die verkauften Smartphones die Anzahl der verkauften PCs. 2015 soll laut Schätzungen der IDC die eine Milliarde Marke an verkauften Smartphones überschritten werden. [WAN2012]

Diese Entwicklung ist den Unternehmen natürlich nicht entgangen, deshalb wollen viele Unternehmen Smartphones als PC-Ersatz nutzen und ihren Mitarbeitern es ermöglichen von überall aus arbeiten zu können. Hierfür gibt es z.B. das Konzept Bring Your Own Device(BYOD). BYOD bezeichnet das benutzen eines privaten Smartphones zu Unternehmenszwecken.

Da Smartphones über eine hohe Rechenleistung, hohe Konnektivität zu Netzen und Internet und meist über eine große Ansammlung von persönlichen Daten verfügen, werden sie aber auch immer mehr zu beliebten Angriffszielen.

Außerdem bieten gehackte Smartphones das perfekte „Überwachungsgerät“. Man könnte die GPS Koordinaten abfragen, um herauszufinden, wo sich die Person gerade aufhält. Abhören starten, da das Smartphone über ein Mikrofon verfügt. Sogar Bilder/Videos machen, da die meisten Smartphones über mindestens eine Kamera verfügen. Ebenso sind die persönlichen Daten ein beliebtes Angriffsziel, da man an Kontakte, Mails, SMS/MMS etc. herankommt und man den Nutzer somit komplett ausspionieren kann.

Leider ist es nicht so einfach, ein hohes Maß an Sicherheit für ein Smartphone zu garantieren, da man nicht wie auf einen Heimrechner einen dauerhaften Echtzeitscanner laufen lassen kann. Der Scanner könnte nicht dauerhaft Zugriffe überwachen, da ein Smartphone nicht über eine dauerhafte Stromversorgung verfügt und so ein Echtzeitscan sehr schnell den die Batterie entladen würde. Dies hat zur Folge, dass die Unternehmen sich Sicherheitskonzepte ausdenken müssen, um die Sicherheit des Firmennetzwerk und der Firmen-

daten zu gewährleisten, wenn sie ihren Angestellten erlauben wollen private Smartphones als Arbeitsgeräte zu verwenden.
Eine weit verbreitete Methode ist es den Angestellten ein Firmenhandy zu kaufen und es zu verbieten, Software bzw. Apps zu installieren, die nicht von den Administratoren des Unternehmens erlaubt sind.
Eine weitere Methode erlaubt es den Angestellten ihre eigenen Smartphones mit Einschränkungen zu nutzen. Bei Verlust oder z.B. Virenbefall des Smartphones hat das Unternehmen das Recht den kompletten Smartphonespeicher zu löschen.
In beiden Fällen hat der Angestellte alle Rechte über das Smartphone abgetreten. Das bedeutet, er kann sein Smartphone nicht komplett frei nutzen bzw. er wird dabei überwacht.
Letzteres ist eine abgewandelte Form des Bring Your Own Device(BYOD) Konzeptes.
Diese Bachelorarbeit beschäftigt sich mit diesen Konzept und wird ein Sicherheitskonzept für BYOD für Androidsmartphones präsentieren, ohne dass die kompletten Rechte für das Smartphone abgetreten werden müssen und ohne dass der Smartphonespeicher bei Befall/Verlust komplett gelöscht werden muss.

1.2 Ziel dieser Arbeit

Das Ziel der Arbeit ist es ein Konzept zur sicheren Nutzung eines privaten Smartphones für die Verarbeitung von Unternehmensdaten und einen Entwurf für dieses Konzept zu erstellen.
Eines der wichtigsten Ziele des Konzepts soll es sein, dass die Angestellten ihr privates Smartphone uneingeschränkt nutzen können. Dies bedeutet, dass das Unternehmen weder das Recht hat den kompletten Smartphonespeicher zu löschen noch es zu überwachen. Dennoch soll die Integrität und Vertraulichkeit der Unternehmensdaten gewährleistet werden. Außerdem sollen die Angestellten auf ihre Daten von überall aus zugreifen können.

1.3 Gliederung der Arbeit

Das erste Kapitel besteht aus der Motivation und dem Ziel dieser Arbeit.
Im zweiten Kapitel werden die Grundlagen, die für diese Arbeit nötig sind, erklärt.
Diese bestehen aus Unterschieden zwischen Mobile- und Desktopsecurity, einigen der bereits vorhandenen BYOD Ansätzen, Verschlüsselungsalgorithmen und VPN.
Das dritte Kapitel befasst sich dann mit den Anforderungen an das BYOD Konzept und die Anwendung.
Im vierten Kapitel folgt dann der Entwurf der Architektur. Es ist in zwei Unterkapitel unterteilt: Der erste Teil beschäftigt sich mit dem Sicherheitskonzept selbst und der zweite Teil besteht aus einem Entwurf zur Umsetzung des Konzepts.
Im fünften Kapitel befindet sich eine Beschreibung der Implementierung eines Prototyps.
Das sechste und letzte Kapitel fasst nochmal die Arbeit zusammen.

2 Grundlagen

Das Grundlagenkapitel erläutert Themen die für das Gesamtverständnis der Arbeit wichtig sind.

2.1 Android Security

2.1.1 Unterschiede zwischen Mobile- und Desktopsecurity

Eines der großen Probleme der Mobilesecurity ist, dass man die bisher bekannten Sicherheitskonzepte nicht ohne weiteres alle auf die mobilen Geräte übertragen kann. Smartphones sind nicht dauerhaft an eine Stromquelle angeschlossen. Deshalb ist es nicht möglich einen Echtzeitscanner sämtliche laufende Prozesse überprüfen zu lassen, denn das würde die Batterie sehr schnell entladen. [CHU2012]
Hierbei ist ein geläufiger Ansatz nach der Installation von Software, diese zu scannen und auf Malware zu untersuchen. Diese Methode stellt aber nur sicher, dass die Software bei der Installation keine Malware beinhaltet hatte/keine gefunden wurde. Es kann also durchaus sein, dass die Software erst zur Laufzeit anfängt sich schädlich zu verhalten. Bei einem normalen Desktop-PC würde der aktive Schutz die Prozesse überwachen und dieses evtl. feststellen.
Des Weiteren kommen bei Smartphones weitere Angriffsmöglichkeiten hinzu. Eine davon ist es durch Malware Premium SMS zu versenden. Dadurch entstehen immense Kosten für den Besitzer des Geräts. Außerdem liegen auf dem Smartphone viele persönliche Daten wie: Kontakte, Bilder, Videos, etc. die durch Malware ausgespäht werden könnten. Eine weitere Angriffsmöglichkeit ist das Smartphone als Ortungs- und Überwachungsgerät zu verwenden, hierbei zeichnet die Malware die GPS Position und Mittschnitte von Gesprächen durch das Mikrofon oder Kamera auf und sendet diese an den Angreifer. [TIT2013]
Bei Androidgeräten muss man außerdem bei der Installation der Software bereits alle Rechte vergeben, die die Software benötigt. Ungeschulte und unaufmerksame Nutzer könnten damit Malware installieren, weil sie sich diese Berechtigungen bei der Abfrage nicht durchlesen oder damit nichts anfangen können.
Androidgeräte haben aber nicht nur Nachteile gegenüber den Desktop-PCs in Bezug auf Sicherheit. Einen der größten Vorteile bietet hier die Dalvik Virtual Maschine des Android.
Die Dalvik VM behandelt jeden Prozess komplett getrennt nach dem Sandboxprinzip und jeder Prozess hat nur bestimmte Zugriffrechte, was eingriffe in den Prozess von anderen Prozessen sehr schwer macht. [KHA2012]

Alle diese Aspekte müssen bei dem Bring your own Device Konzept beachtet werden, da sensible Unternehmensdaten auf den Geräten liegen könnten.
Folgend befinden sich einige Ansätze um die oben genannten Sicherheitsprobleme zu bewältigen.

2.1.2 Ansätze für das BYOD Problem

Folgend nun einige Ansätze für BYOD:

Abtretung der Rechte des Smartphones

Eines der z.Z. gängigen Konzepte ist die Abtretung der Rechte am Smartphone. Das bedeutet, dass das Unternehmen das Gerät komplett überwachen oder sogar auch sperren/löschen kann. Hierbei ist es meist üblich, dass der Angestellte sich am Einkauf des Smartphones beteiligt oder es den Angestellten kostenlos zur Verfügung gestellt wird. Das Unternehmen hat dann die volle Kontrolle über das Gerät. Der Angestellte kann das Gerät zwar für seine eigene Zwecke nutzen aber nur mit Software, die das Unternehmen genehmigt hat. Außerdem bleibt immer noch der Aspekt, dass das Unternehmen die Kommunikation überwachen darf. [ACK2013]

Nutzung von Cloudservices

Ein anderer Ansatz ist es, die Unternehmensdaten in einer Cloud zu speichern und diese mit den Smartphones abzurufen. Hier gibt es bereits einige gute Ansätze wie z.B. das Nutzen von Profilen, indem man die Smartphones so einstellt, dass diese keine unternehmensrelevanten Daten außerhalb der Arbeitszeiten abrufen können. Außerdem können Angestellte nicht innerhalb der Arbeitszeiten frei im Internet surfen. Damit ist gemeint, dass Angestellte keine Spiele oder anstößige Seiten etc. während der Arbeitszeit aufrufen können. Außerhalb der Arbeitszeiten können sie aber das Smartphone für ihre eigenen Zwecke verwenden. Für zusätzliche Sicherheit wird ein Antivirus Programm auf dem Smartphone installiert. Dieses scannt Daten die von oder zu der Cloud geschickt werden. In der Cloud werden die Daten dann nochmal wesentlich gründlicher gescannt, da es sich hier um einen Server handelt. Der Server hat die Ressourcen dafür und kann somit das Verhalten der Daten genau beobachten. [CHU2012]

Verschlüsselung des Speichermediums

Bei diesem Ansatz wird die Speicherkarte des Geräts verschlüsselt um die Daten nicht zugänglich zu machen. Somit würden bei Diebstahl oder Verlust des Geräts die Daten ohne einen Schlüssel unbrauchbar sein. [WAN2012]

Virtualisierung
Bei diesem Ansatz muss man zwischen zwei Arten der Virtualisierung unterscheiden:
Die erste Art ist die VM auf einem Server laufen zu lassen und per Remote-Verbindung die VM vom Smartphone aus zu steuern. Dabei werden keine Daten auf den Smartphone geladen, es steuert lediglich die VM. [SCA2012]
Die zweite Art ist es die VM auf dem Smartphone parallel laufen zu lassen. Hierbei wird strickt zwischen persönlichen und geschäftlichen Modus unterschieden. Der geschäftliche Modus ist die VM, zu der man wechseln kann. Das hier laufende Betriebssystem ist eingeschränkt für den Benutzer, d.h. er kann viele Einstellungen nicht ändern oder Apps installieren. Im persönlichen Modus kann der Benutzer Apps installieren und Einstellungen ändern wie es ihm beliebt. Hierbei ist der persönliche Modus das eigentliche Betriebssystem und der Geschäftsmodus ist eine darauf laufende VM. Hierdurch sind Daten des Unternehmens von den privaten Daten gesperrt. Diese lassen sich natürlich auch verschlüsseln. Bisher funktionierende Implementierungen sind Samsung SAFE und BlackBerry Balance. [SAM2014 und BLA2014]

2.2 Android Programmierung

Ein weiterer wichtiger Aspekt der Arbeit ist die Umsetzung des Clients auf einem Android Smartphone. Die Android Programmierung unterscheidet sich ebenfalls etwas von der normalen Programmierung von Desktopanwendungen. Da wie bereits 2.1.1 erwähnt die Ressourcen des Smartphones beschränkt sind, muss die Anwendung einige Methoden implementieren und einige Einschränkungen beachten.
Bei einer Android Anwendung gibt es einen Programm Lifecycle. Dieser besteht aus:
Create: Erstellung des Prozesses, Start: Starten der Anwendung, Resume: Anwendung in Ausführung, Pause: Anwendung wird von einem anderen Prozess Teilweise überdeckt und keine Interaktion ist möglich, Stopp: Anwendung ist komplett im Hintergrund, Destroy: Anwendung wird beendet und die Ressourcen werden freigegeben.
Beim Programmieren von Android Anwendungen sind besonders die Pause und Stopp Abschnitte wichtig. Hierbei kann es sein, dass das Betriebssystem aufgrund von Speichermangel die Instanzen der Anwendung löscht, d.h. jede nicht persistent gespeicherte Änderung kann verloren gehen. Hierfür muss dann im Resume Abschnitt beachtet werden, dass Daten gelöscht werden könnten und es muss für die Wiederherstellung der Daten gesorgt werden.
Außerdem kann man zwar bei der Android Programmierung die Programmiersprache Java verwenden, muss aber zusätzlich bestimmte Paradigmen einhalten. Die wichtigsten sind die Activities und die Intends. Eine Activity ist eine Klasse die eine Interaktion vom Benutzer erlaubt. Activities besitzen die oben genannten Lifecycles und sind Klassen, die das Android Programm verwenden kann. Die einzige Methode, die eine Activity voraussetzt ist die onCreate Methode für die Erstellung des Programms. Die anderen Lifecycle Methoden können bei Bedarf überschrieben werden.

Intends sind “Absichten“ von einer Activity zu einer anderen zu wechseln. Ein Intend kann auch weitere Informationen in einem “Extras“ Feld von einer Activity zur anderen übertragen. Wenn von einer Activity zu einer anderen Activity durch einen Intend gewechselt wurde, wird die vorher aktive Activity auf einen Stack abgelegt. Sobald die derzeitig aktive Activity fertig ist, wird die vorherige Activity von Stack genommen und weiterverwendet.
Man kann größere Programme durch mehre Activities lösen. Hierbei wird für jede Activity eine eigene GUI erstellt. Des Weiteren gibt es für dynamische GUIs die Fragments. Mit diesen kann man in einer Activity mehre GUIs implementieren. Die Fragments werden dynamisch von einer Activity geladen oder die Activity wechselt zwischen unterschiedlichen Fragments. [AND2014]
Man kann sich ein Fragment als ein Tab einer GUI vorstellen, wenn also eine Activity mehrere GUIs braucht wie z.B. eine GUI für das Einloggen und eine weitere für das Verarbeiten von Daten nach dem Login, kann man mit der Activity diese GUIs austauschen. Die Activity selbst braucht hierfür nur eine „leere“ GUI um als Container für die Fragments zu dienen.

2.3 Verschlüsselungsalgorithmen

Bei der Verschlüsselung von Dateien gibt es 2 wichtige Arten von Algorithmen:
symmetrische und asymmetrische Verschlüsselung:

2.3.1 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsslung basieren auf mathematischen Problemen und wird sowohl für Verschlüsslung als auch zum Siegnieren von Daten genutzt. Es wird auf einem mathematischen Problem basierend ein Public-Key und ein Private-Key erzeugt. Der Public-Key wird veröffentlicht und jeder kann diesen nutzen, der Private-Key hingegen ist nur dem Besitzer bekannt, der die Keys generiert hat und nutzen möchte. Der Besitzer kann nun mit dem Private-Key seine Daten verschlüsseln und somit eine Signatur erstellen. Meist wird nur ein Hash über die Daten gebildet und dieser signiert, da große Daten eine genau so große Signatur benötigen. Durch die Signatur kann überprüft werden, ob die Daten nicht bei der Kommunikation von 3ten verändert wurden. Möchte nun ein Benutzer eine Nachricht an den Besitzer schicken, verschlüsselt er die Daten mit dem Public-Key und sendet diese an den Besitzer des Private-Keys. Der Besitzer kann nun mit seinen Private-Key die Daten entschlüsseln. [FAN2012]

2.3.2 Symmetrische Verschlüsselung

Anders als bei der asymmetrischen Verschlüsselung, gibt es bei der symmetrischen Verschlüsselung nur einen gemeinsamen Schlüssel. Mit diesem Schlüssel wird sowohl verschlüsselt als auch entschlüsselt. Dieser Schlüssel muss beiden Seiten bekannt sein, wenn diese Daten verschicken möchten.
Zu den bekanntesten Algorithmen zählen hier: AES, Blowfisch und 3DES. Alle 3 Algorithmen verschlüsseln die Daten blockweise. Die Blöcke werden je nach Algorithmus dann permutiert und substituiert. Anschließend wird der Vorgang mehrfach wiederholt.
Da der DES Algorithmus aufgrund der 56 Bit Schlüsselläge heutzutage als nicht mehr sicher gilt, wurde 3DES entwickelt. Dieses Verfahren wendet den DES Algorithmus 3-mal hintereinander auf den Klartext an. Dadurch werden Schlüssellängen von bis zu 168 Bit möglich. Der neuere AES Algorithmus hat Schlüssellängen von 128, 192 und 256 Bit.
Der AES Algorithmus wurde von der NIST als Nachfolger des DES ausgewählt und ist mittlerweile der Standard Verschlüsselungsalgorithmus. [NAD2005]

2.4 VPN

Das Prinzip hinter VPN ist es ein Gerät oder Netzwerk an ein bestehendes z.B. Unternehmensnetzwerk sicher über das Internet(unsichere Leitungen) anzubinden. Das daraus entstandene Netzwerk nennt man auch Virtual Private Network(VPN).
Es gibt 2 Unterschiedliche Arten von VPN:
Remote Access VPN: Es verbindet sich ein Gerät zu dem Netzwerk.
Site-to-Site VPN: Mehrere Netzwerke mit einander virtuell verbinden.
In dieser Arbeit wird nur auf Remote Access VPN eingegangen, auf Site-to-Site VPN wird nicht näher eingegangen. Um VPN nutzen zu können, muss das Unternehmensnetz einen Gateway bereitstellen zu dem sich VPN-Teilnehmer verbinden können. Sobald diese verbunden sind, sind sie in das Netzwerk eingebunden und sind in ihrem ursprünglichen Netzwerk nicht mehr ohne weiteres erreichbar. Die sichere Kommunikation wird durch verschlüsseln der Datenpakete erreicht. Hier einige Beispiele wie die sichere Kommunikation erreicht werden kann:
Eine der Möglichkeiten ist es, IPsec im ESP(Encapsuling Security Payload) zu verwenden. Hierbei werden die ursprünglichen IP-Pakete verschlüsselt und in neue IP-Pakete gepackt(auch Tunneling genannt). Diese werden dann versendet und an der anderen Seite entpackt und entschlüsselt. Dieses Verfahren hat aber den Nachteil, dass es bei NAT einen höheren Konfigurationsaufwand hat.

Außerdem gibt es noch das SSL VPN, bei dem zwischen der Anwendungsschicht und der Transportschicht die Daten verschlüsselt werden. Diese Art von VPN wird meistens nicht dazu genutzt Netzwerke mit einander zu verbinden, obwohl dies möglich wäre, sondern es wird meist zum Einbinden von einzelnen Geräten oder nutzen bestimmter Dienste benutzt. Hierbei findet die Kommunikation meistens über den Port 443(Https) statt. Da dieser Port in den meisten Netzwerken freigegeben ist, ist diese Art des VPN sehr für mobile Nutzer geeignet.
Alle oben genannten Möglichkeiten können so konfiguriert werden, dass sie die Vertraulichkeit, die Integrität und die Authentifikation sicherstellen. [ABD2012]

3 Anforderungsanalyse

Um die Zielsetzung dieser Arbeit zu erreichen, muss nun eine Anforderungsanalyse durchgeführt werden um die Anforderungen an die App auszuarbeiten. Die Anwendung hat hierbei nicht nur Anforderungen als App, sondern auch Sicherheitsanforderungen, da Unternehmensdaten sehr sensibel sind. Außerdem werden hier die wichtigsten Anwendungsfälle für die App beschrieben, die Einsicht in den Ablauf des Konzeptes geben.

Anforderungen an die Anwendung:
Das Unternehmen möchte, dass Angestellte ihre Daten die auf den Unternehmensserver liegen von überall aus lesen/bearbeiten können. Hierbei können sie ihr eigenes privates Smartphone verwenden. Da dies mit Smartphones möglich sein soll, wird für die Lösung eine Android App verwendet. Der Angestellte muss hierzu die App installieren, die sich dann mit dem Unternehmensnetz-werk verbindet. Außerdem sollen die Daten auf dem Smartphone ebenfalls sicher und verschlüsselt gelagert werden, so dass sogar bei Verlust des Smartphones die Daten nur in verschlüsselter Form auf dem Gerät sind und nicht mit vertretbaren Aufwand eingesehen werden können. Der Benutzer der App muss sich mit seinen Zugangsdaten über die App zum Server verbinden und kann somit Daten einsehen oder bearbeiten. Das Bearbeiten der Daten selbst wird von der App an die dafür zuständigen Programme weitergeleitet. Bei Verlust oder Malwarefund muss das Smartphone gesperrt werden können, sodass es keinen Zugang zum Unternehmensnetzwerk mehr hat.

Sicherheitsanforderungen:
Die Vertraulichkeit und Integrität der Daten muss von der Anwendung sichergestellt werden. Hierbei ist es wichtig, dass Daten sobald diese nicht benutzt werden nur auf dem Server oder File Server unverschlüsselt liegen. Auf dem Smartphone sollen die Daten nur zur Verarbeitung entschlüsselt und temporär abgelegt werden. Sobald diese nicht mehr in Bearbeitung sind, müssen sie wieder verschlüsselt werden. Die Kommunikation zwischen Server und Anwendung soll sicher sein d.h. die versendeten Datenpakete sind immer verschlüsselt, so dass diese nicht mitgelesen werden können. Des Weiteren soll die Anwendung sicherstellen dass men-in-the-middle Angriffe nicht ohne weiteres möglich sind. Der Server darf nur autorisierte Benutzer die Dateien lesen und bearbeiten lassen. Die von der Anwendung verwendete Software(z.B. Betriebssystem und Virenscanner) müssen stets auf den aktuellsten stand sein um mögliche Sicherheitslücken zu schließen. Der Server sollte so wenig freie Ports ins Internet wie möglich haben um mögliche Angriffe zu minimieren.
Die Daten eines Angestellten dürfen auch nur von diesem eingesehen werden, sowohl wenn diese noch auf dem Server als auch auf dem Smartphone liegen.

Daraus ergeben sich nun folgende funktionale und nichtfunktionale Anforderungen:

3.1 Funktionale und nicht funktionale Anforderungen

3.1.1 Funktionale Anforderungen

Die Anforderung unterteilen sich in Client/Anwendung und Server Anforderungen.

Anforderungen an den Client/Anwendung

A1.1 Dem Benutzer soll es mögliche sein durch das Installieren und Benutzen einer Android Anwendung Zugriff auf seine Unternehmensdaten zu erhalten.

A1.2 Beim Starten der Anwendung soll diese über das Internet bei dem Server anfragen ob das Gerät gesperrt ist um sicherzustellen, dass keine befallenen oder vermissten Smartphones ins Netzwerk gelassen werden.

A1.3 Die Anwendung muss vor dem Verbinden zum VPN-Gateway einen Virenscanner anstoßen und gemäß dessen Rückmeldung die VPN Verbindung aufbauen oder dem Server melden, dass das Gerät befallen ist.

A1.4 Die Anwendung soll bei Virenbefall oder mehrfacher falscher Passworteingabe über das Internet eine Sperrmeldung verschicken, um den Server mitzuteilen, dass es befallen ist oder jemand zu oft das Passwort falsch eingegeben hat.

A1.5 Bei Sperrung des Geräts für VPN durch Virenbefall oder falsche Passworteingabe, soll die Anwendung alle relevanten Unternehmensdaten löschen und sich beenden, damit die Unternehmensdaten nicht mehr einsehbar sind.

A1.6 Die Anwendung soll eine VPN Verbindung zu einem Server aufbauen und diese Nutzen können. Es dürfen insgesamt 5 Anmeldeversuche stattfinden. Danach wird das Gerät für VPN Zugang gesperrt. Normalerweise werden max. 3 Anmeldeversuche erlaubt, da es hier aber zu höheren Konfigurationsaufwand führt, wenn man die Geräte wieder entsperren möchte und da man sich auf dem Smartphone relativ einfach vertippen kann, sind hier 5 Anmeldeversuche erlaubt. Die Login Daten sehen wie folgt aus: Name: Username, Passwort: Benutzererstelltes Passwort, dass mindestens 8 Zeichen lang ist, 1 Großbuchstaben und 1 Zahl enthält. [GEH2002]

A1.7 Bei einem Verbindungsabbruch soll die Anwendung 10-mal in 3 Sekunden Abständen versuchen neu zu verbinden, sollte dies nicht erfolgreich sein, muss die Anwendung sich neu anmelden. Diese Anforderung soll das Arbeiten mit der Anwendung für den Benutzer erleichtern.

A1.8 Die Anwendung muss alle Unternehmensdaten, die es vom Server bekommt und auf den Smartphone speichern soll, verschlüsselt auf dem Smartphone ablegen. Dies dient zur Sicherstellung der Vertraulichkeit der Daten. Hierdurch können die Daten nicht mit vertretbarem Aufwand von Dritten gelesen werden.

A1.9 Die Anwendung muss die Aktualität der Daten feststellen können um Konflikte bei der Versionierung zu vermeiden

A1.10 Die Anwendung soll dem Benutzer die Möglichkeit bieten alle nicht aktuellen Daten zu aktualisieren.

A1.11 Die Anwendung ermöglicht es dem Benutzer seine Daten zu bearbeiten, indem es die Daten vom Server empfängt und an das dafür vorgesehene Programm weitergibt. Die Dateien werden an externe Programme weitergegeben, da es sonst nötig wäre für jede Datei Art eine eigene Implementierung zum Lesen/Bearbeiten zu entwickeln.

A1.12 Die Anwendung muss den Benutzer warnen, wenn dieser Versucht auf einer nicht aktuellen Kopie der Datei zu arbeiten.

A1.13 Die Anwendung muss verschlüsselt abgelegte Dateien temporär entschlüsseln und speichern können. Dadurch können die Dateien gelesen und bearbeitet werden.

A1.14 Nach Bearbeitung einer Datei soll die Anwendung einen Zeitstempel bei der Datei setzen. Dies dient zur Feststellung der Aktualität der Datei bei der Synchronisierung zum Server.

A1.15 Die Anwendung soll es dem Benutzer erlauben sein Passwort bei fehlerhafter Entschlüsselung erneut eingeben zu lassen. Diese Anforderung ist nötig, da das Verschlüsselungspasswort der Dateien sich ändern kann. Dadurch werden die Dateien mit dem alten Passwort entschlüsselt und mit dem neuen wieder verschlüsselt. Hierdurch wird ein neues Anfordern aller Dateien vom Server vermieden, wodurch auch Änderungen an Dateien nicht überschrieben werden.

A1.16 Die Anwendung muss beim Beenden alle noch temporär entschlüsselten Dateien verschlüsseln und an den Server senden. Sobald diese beim Server aktualisiert sind, muss die Anwendung die entschlüsselten Dateien löschen und die VPN Verbindung beenden. Dies hat zur Folge, dass sich keine unverschlüsselten und somit von Dritten lesbaren Dateien mehr auf dem Smartphone befinden.

A1.17 Die Anwendung muss Erfolgs- und Fehlermeldungen an den Benutzer weitergeben können.

A1.18 Die Anwendung muss die entschlüsselten Dateien, nach dem Bearbeiten wieder verschlüsseln und auf dem Smartphone speichern können. Nach dem Verschlüsseln wird die entschlüsselte Datei gelöscht.

Anforderungen an den Server

A2.1 Der Server muss auf Anfragen der Anwendung antworten können.

A2.2 Der Server muss über das Internet Anfragen zu Sperrungen beantworten bzw. durchführen können.

A2.3 Der Server verwaltet eine Abbildung von GeräteIDs auf Benutzernamen des VPN Zugangs. Da das versenden von Benutzernamen über das Internet ohne VPN ein unnötiges Sicherheitsrisiko darstellen würde, senden die Smartphones außerhalb des VPN nur ihre IDs. Diese IDs können nur für die Sperrungen genutzt werden und dienen keinen anderen Zweck. Der Server kann mit den IDs den Benutzernamen des VPN Zugangs feststellen und diesen Sperren, falls nötig.

A2.4 Die Sperrliste des Servers muss manuell durch Administratoren konfigurierbar sein, um Geräte zu entsperren und neue Geräte einzutragen.

A2.5 Der Server meldet gesperrte Benutzer an den VPN Gateway.

A2.6 Der Server ist an einen File Server angebunden. Da der Server eine Art Firewall darstellt, ist es sicherer die Dateien auf einen anderen Rechner zu lagern.

A2.7 Der Server muss bei dem File Server Dateien abrufen, speichern und löschen können.

A2.8 Der Server soll das letzte Bearbeitungsdatum der Daten vom File Server überprüfen können.

A2.9 Der Server muss Dateien, die er vom File Server abgerufen hat, an die Anwendung senden können.

A2.10 Der Server muss Dateien, die er von der Anwendung erhält, auf den File Server aktualisieren/löschen können.

A2.11 Der Server leitet Fehlermeldungen transparent an die Anwendung weiter.

A2.12 Der Server scannt die empfangenen Dateien mit 2 unterschiedlichen Scannern. Es werden 2 Scanner benutzt um die Wahrscheinlichkeit auf einen Fund zu erhöhen.

A2.13 Der Server sendet Sperrnachrichten an die Anwendung bei Virenfund aus Punkt A2.12.

3.1.2 Nicht funktionale Anforderungen

A3.1. Die Kommunikation zwischen Anwendung und Server muss sicher sein, sodass die Kommunikation nicht mit vertretbaren Aufwand abgehört/entschlüsselt werden kann.

A3.2. Die Kommunikation zwischen Server und Anwendung soll nicht für den Benutzer sichtbar sein.

A3.3. Die Anwendung gibt dem Benutzer aussagekräftige/transparente Fehlermeldungen.

A3.4. Die Architektur soll flexibel und erweiterbar sein und so wenige Einschränkungen wie möglich beinhalten.

3.1.3 Abgrenzungen

A4.1. In dieser Arbeit werden nur Android-Smartphones betrachtet, Apple und Windowsgeräte sind ausgeschlossen.

A4.2. Der File Server ist bereits vorhanden und muss nicht implementiert/konfiguriert werden.

A4.3. In dieser Arbeit wird mit einer generischen Antivirus- und VPN API gearbeitet.

A4.4. Die Sicherheit des internen Unternehmensnetzwerks wird in dieser Arbeit vorausgesetzt.

3.2 Anwendungsfälle

Anwendungsfall AF1:	**Benutzer verbindet sich über VPN zum Server**
Akteur:	Angestellter
Auslöser:	Benutzer möchte Daten bearbeiten
Vorbedingungen:	Anwendung ist installiert
Nachbedingungen:	Anwendung ist gestartet und bereit zur Benutzung

Erfolgsszenario:

1. Benutzer startet die Anwendung und möchte Daten bearbeiten.
2. Die Anwendung fragt beim Server an, ob das Gerät gesperrt wurde.
3. Die Anwendung lässt durch den installierten Virenscanner das Smartphone scannen.
4. Der Virenscanner meldet keinen Virenfund.
5. Die Anwendung fordert den Benutzer auf, seinen Loginnamen und Passwort in die Felder Login und Passwort einzugeben.
6. Der Benutzer gibt die oben genannten Daten ein.
7. Die Anwendung baut eine VPN Verbindung zum Unternehmensserver auf.
8. Die Anwendung überprüft, ob die Daten auf dem Smartphone aktuell sind.
9. Die VPN Verbindung steht und der Benutzer kann nun Daten abrufen/verändern.

Fehlerfälle:

a. Zu 2. Gerät ist gesperrt -> Siehe AF7 Punkt 8.
b. Zu 4. Virenscanner hat einen Virenbefall entdeckt -> Siehe AF7 ab Punkt 3.
c. Zu 6. Benutzerdaten wurden falsch eingegeben -> 4 weitere Versuche die Daten einzugeben, danach wird das Smartphone gesperrt. Siehe AF7 ab Punkt 4.
d. Zu 8. Daten sind nicht aktuell -> Die Anwendung fragt den Benutzer, ob alle Daten aktualisiert werden sollen. Wenn der Benutzer nein auswählt werden nur die Daten aktualisiert, die er auch bearbeiten/öffnen möchte.

Anforderungen: A1.1, A1.2, A1.3, A1.4, A1.6, A1.9, A1.10

Anwendungsfall AF2:	**Benutzer fordert eine Datei über die Anwendung an**
Akteur:	Angestellter
Auslöser:	Benutzer möchte die Daten bearbeiten.
Vorbedingungen:	Anwendung ist gestartet und Anwendungsfall: AF1 ist erfolgreich beendet Die angeforderte Datei befindet sich noch nicht auf dem Smartphone
Nachbedingungen:	Die zu bearbeitenden Daten befinden sich auf dem Smartphone und können bearbeitet werden.

Erfolgsszenario

1. Benutzer möchte eine Datei X bearbeiten.
2. Benutzer wählt die Datei X in der Anwendung aus und klickt auf bearbeiten/öffnen.
3. Die Anwendung fordert vom Server die Datei X an.
4. Der Server fordert die Datei bei dem Fileserver an.
5. Der Fileserver schickt die Datei an den Server.
6. Der Server schickt die Datei über VPN an die Anwendung.
7. Die Anwendung empfängt die Datei.
8. Die Anwendung legt die Datei temporär entschlüsselt an.
9. Der Benutzer kann diese Datei nun bearbeiten.

Fehlerfälle:

a. Zu 1-8 Die Verbindung bricht ab -> Die Anwendung versucht 10-mal neu alle 3 Sekunden neu zu Verbinden. Sollte keine Verbindung möglich sein, muss sich neu eingeloggt werden. Siehe AF1 ab Punkt 1.
b. Zu 4. Die Datei X existiert nicht auf dem Fileserver -> Server gibt Meldung an die Anwendung weiter. Die Anwendung gibt die Meldung an den Benutzer aus. Zurück zu Punkt 2.

Anforderungen: A1.7, A1.13, A1.17, A2.1, A2.6, A2.7, A2.11

Anwendungsfall AF3:	**Benutzer fordert eine Datei über die Anwendung an**
Akteur:	Angestellter
Auslöser:	Benutzer möchte die Daten bearbeiten.
Vorbedingungen:	Anwendung ist gestartet und Anwendungsfall: AF1 ist erfolgreich beendet Die angeforderte Datei befindet sich bereits auf dem Smartphone
Nachbedingungen:	Die Datei ist auf dem neusten Stand und kann bearbeitet werden.

Erfolgsszenario:

1. Benutzer möchte Datei X öffnen/bearbeiten.
2. Benutzer wählt Datei X aus und klickt auf öffnen/bearbeiten.
3. Die Anwendung prüft, ob die Datei aktuell ist.
4. Die Anwendung entschlüsselt die Datei.
5. Die Anwendung legt eine temporäre Datei an, die entschlüsselt ist.
6. Der Benutzer kann nun die Datei bearbeiten/öffnen.

Fehlerfälle:

a. Zu 3. Datei ist nicht aktuell -> Anwendung fordert die Datei X beim Server an. Siehe AF2 ab 3. Weiter bei 5.
b. Zu 4. Schlüssel stimmt nicht überein -> Fehler an Benutzer weitergeben. Die Anwendung fordert eine Erneute Passworteingabe an.
c. Zu 5. Nicht genügend Speicherplatz für die Datei -> Fehlermeldung an den Benutzer weitergeben.

Anforderungen: A1.6, A1.8, A1.9, A1.13, A1.17

Anwendungsfall AF4:	**Verlust des Smartphones**
Akteur:	Angestellter, Administrator
Auslöser:	Der Benutzer hat sein Smartphone verloren.
Vorbedingungen:	Anwendung ist installiert.

1. Benutzer meldet Verlust des Smartphones.
2. Administrator fügt den Benutzer zur Sperrliste hinzu.
3. Anwendung wird von einem unbekannten Benutzer gestartet.
4. Anwendung fragt beim Server an ob das Gerät gesperrt ist.
5. Server antwortet, dass das Gerät gesperrt ist.
6. Anwendung löscht alle Unternehmensrelevanten Daten.

Anforderungen: A1.5, A2.1, A2.2, A2.4

Anwendungsfall AF5:	**Benutzer bearbeitet Datei X und speichert diese.**
Akteur:	Angestellter
Auslöser:	Benutzer hat eine Datei bearbeitet und möchte diese speichern
Vorbedingungen:	Anwendung ist gestartet und Anwendungsfall: AF1 ist erfolgreich beendet. Anwendungsfall AF2 oder AF3 ist erfolgreich abgeschlossen.
Nachbedingungen:	Die vom Benutzer bearbeitete Datei ist auf dem File Server gespeichert und wurde auf dem Smartphone verschlüsselt gespeichert.

Erfolgsszenario:

1. Benutzer bearbeitet die Datei.
2. Benutzer speichert die Änderungen an der Datei X.
3. Die Anwendung speichert die Änderungen in die temporär angelegte Datei ab. (siehe AF2 Punkt 8. oder AF3 Punkt 5.)
4. Die Anwendung schickt die temporär entschlüsselte Datei an den Server.
5. Der Server empfängt die Datei.
6. Der Server scannt die Datei mit 2 unterschiedlichen Virenscannern.
7. Der Server aktualisiert/überschreibt die alte Datei mit der neuen Datei X auf dem File Server.
8. Der Server meldet Erfolg an die Anwendung.
9. Die Anwendung verschlüsselt die temporäre Datei X.
10. Die Anwendung setzt bei der verschlüsselten Datei X den Änderungszeitstempel auf das aktuelle Datum und die aktuelle Uhrzeit.
11. Die Anwendung speichert die verschlüsselte Datei X.
12. Die Anwendung meldet Erfolg an den Benutzer.

Erweiterungen:

12.a. Der Benutzer möchte die nicht mehr weiter bearbeiten: Die Anwendung löscht nach Punkt 9.-11. Die entschlüsselte Datei.

Fehlerfälle:

a. Zu 6. Malware wird entdeckt -> Server antwortet mit Sperrung des Geräts siehe AF8
b. Zu 4. Fehler bei der Übertragung -> Datei wird erneut gesendet.
c. Zu 7. Fehler beim überschrieben/aktualisieren -> 3-mal jede Sekunde Erneut Versuchen. Danach an Anwendung Fehler melden. Anwendung meldet Fehler an Benutzer weiter.

Anforderungen: A1.8, A1.13, A1.14, A1.18, A2.10, A2.12, A2.13

Anwendungsfall AF6:	**Benutzer öffnet Datei X um diese zu Lesen.**
Akteur:	Angestellter
Auslöser:	Der Benutzer möchte eine Datei öffnen um diese zu Lesen.
Vorbedingungen:	Anwendung ist gestartet und Anwendungsfall: AF1 ist erfolgreich beendet. Anwendungsfall AF2 oder AF3 ist erfolgreich abgeschlossen.
Nachbedingungen:	Die Datei ist geöffnet.

Erfolgsszenario:

1. Benutzer klickt auf Öffnen in der Anwendung.
2. Die Anwendung überprüft, ob die Datei aktuell ist.
3. Die Anwendung entschlüsselt die Datei.
4. Die Anwendung legt die entschlüsselte Datei temporär auf dem Smartphone an.
5. Die Anwendung öffnet die Datei mit dem dafür vom Benutzer ausgewählten Programm
6. Der Benutzer kann die Datei nun lesen.

Fehlerfälle:

a. Zu 2. Datei nicht aktuell -> siehe AF2 ab Punkt 3. bis Punkt 7. Weiter bei Punkt 3.
b. Zu 3. Fehler beim Entschlüsseln -> Siehe AF3 Fehlerfall b.
c. Zu 4. Nicht genügend Speicherplatz -> Siehe AF3 Fehlerfall c.
d. Zu 5. Kein Programm zum öffnen gefunden -> Fehler an Benutzer weitergeben.

Anforderungen: A1.9, A1.11, A1.13, A1.17

Anwendungsfall AF7:	**Sperren eines Smartphones für VPN**
Akteur:	Angestellter
Auslöser:	Der Benutzer möchte die Daten bearbeiten.
Vorbedingungen:	Anwendung ist installiert.
Nachbedingungen:	Der Benutzer hat keinen VPN Zugang zu den Unternehmensdaten mehr und die Anwendung hat alle Unternehmensrelevanten Daten gelöscht.

Erfolgsszenario

1. Benutzer startet die Anwendung.
2. Die Anwendung lässt durch einen Virenscanner das Smartphone scannen
3. Der Virenscanner meldet einen oder mehrere Funde
4. Die Anwendung sendet eine Sperraufforderung an den Server über das Internet.
5. Der Server empfängt die Aufforderung.
6. Der Server ordnet die mitgesendete ID dem Benutzer zu.
7. Der Server löscht den Benutzer für VPN Zugang und fügt ihn der Sperrliste hinzu.
8. Die Anwendung löscht alle Unternehmensrelevanten Daten in seinem Ordner

Fehlerfälle:

a. Zu. 4. Kein Internet verfügbar -> Weiter ab 8.

Anforderungen: A1.3, A1.4, A1.5, A2.2, A2.3, A2.5

Anwendungsfall AF8:	**Serverseitiger Virenfund**
Akteur:	Angestellter
Auslöser:	Der Benutzer hat eine Datei zum speichern an den Server geschickt.
Vorbedingungen:	Anwendung ist installiert. Anwendung ist gestartet und Anwendungsfall AF1 ist erfolgreich abgeschlossen. Anwendung hat eine Datei an den Server geschickt.
Nachbedingungen:	Der Benutzer hat keinen VPN Zugang zu den Unternehmensdaten mehr und die Anwendung hat alle Unternehmensrelevanten Daten gelöscht.

Erfolgsszenario:

1. Server empfängt die Datei, die die Anwendung gesendet hat
2. Einer der Virenscanner des Servers meldet einen Fund.
3. Server sendet eine Sperraufforderung an die Anwendung
4. Server fügt den Benutzer zur Sperrliste für VPN ein
5. Anwendung löscht alle Unternehmensdaten und schließt sich

Fehlerfälle:

a. Zu 3. Nachricht kommt nicht an -> weiter ab Punkt 4. Der 5. Punkt wird nicht direkt ausgeführt. Spätestens beim Neustart der Anwendung wird dann aber eine Sperraufforderung empfangen und Punkt 5 wird ausgeführt.

Anforderungen: A1.5, A2.5, A2.12, A2.13

Anwendungsfall AF9:	**Beenden der Anwendung**
Akteur:	Angestellter
Auslöser:	Der Benutzer hat die Daten bearbeitet und möchte die Anwendung jetzt schließen
Vorbedingungen:	Anwendung ist installiert. Anwendung ist gestartet und Anwendungsfall AF1 ist erfolgreich abgeschlossen.
Nachbedingungen:	Die bearbeiteten Daten sind auf dem File Server aktualisiert worden. Die entschlüsselten Dateien sind vom Smartphone gelöscht worden. Die VPN-Sitzung ist beendet. Die Anwendung ist geschlossen.

Erfolgsszenario

1. Benutzer möchte die Anwendung beenden
2. Benutzer klickt beenden in der Anwendung an
3. Die Anwendung sendet alle geänderten Daten an den Server.
4. Die Anwendung verschlüsselt alle noch offenen temporär angelegten Dateien.
5. Die Anwendung speichert die verschlüsselten Dateien.
6. Die Anwendung löscht alle temporär entschlüsselten Dateien.
7. Die Anwendung beendet die VPN Sitzung

Anforderungen: A1.16

4 Architektur

Nach der Anforderungsanalyse folgen nun die Ausarbeitung des Sicherheitskonzepts und die hier vorgeschlagene Umsetzung des Konzepts. Dieses Kapitel ist in 2 Unterkapitel eingeteilt. In 4.1 wird das Sicherheitskonzept beschrieben und analysiert. In Kapitel 4.2 befindet sich der vorgeschlagene Entwurf zur Umsetzung des Konzepts

4.1 Entwurf der Sicherheitsarchitektur

Das hier vorgestellte Konzept ist ein Sicherheitskonzept zur sicheren Datenverarbeitung. Damit ist gemeint, dass nicht die Sicherheit der laufenden Applikationen gegen Angriffe geschützt wird, sondern dass die Unternehmensdaten vor einem Angreifer geschützt werden. Es basiert auf der Ver- und Entschlüsselung der Daten sowie aus einem oder mehreren Clients und einer serverseitigen Anwendung.
Für die Kommunikation zwischen Client und Server wird, bis auf die Sperranfragen und die Sperraufforderungen, VPN benutzt. Die Clients lassen sich durch den Server bei Bedarf sperren, wodurch sie keinen Zugang mehr zum VPN haben.
Die wichtigsten Aspekte des Konzepts sind: Kommunikation über VPN, Verschlüsselung der Daten und Antivirenschutz für Datenintegrität.

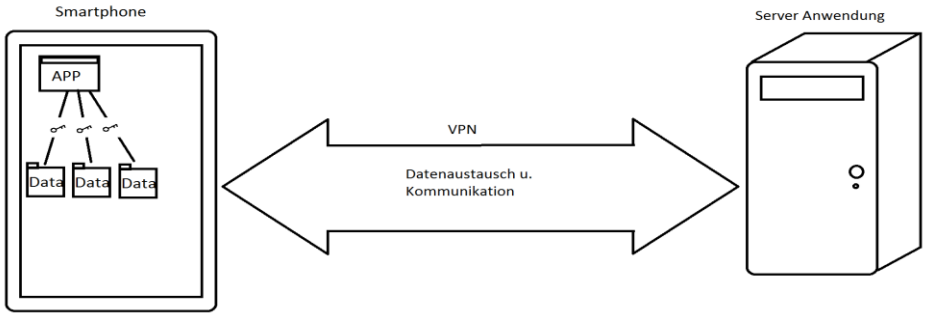

Abbildung 1: Konzeptskizze

Kommunikation über das Internet:
Bei diesem Konzept ist es nötig, dass bevor die VPN Verbindung hergestellt wird, Nachrichten mit dem Server ausgetauscht werden. Die Kommunikation außerhalb des VPN beschränkt sich lediglich auf Sperranfragen und Sperraufforderungen. Eine Sperranfrage ist eine vom Client gestellte Anfrage ob das Gerät gesperrt ist und somit keinen Zugang zum VPN hat. Eine Sperraufforderung hingegen ist eine Aufforderung an den Server das gerät zu Sperren oder eine Aufforderung des Servers den Client zu sperren. Diese werden gesendet

wenn das Gerät entweder durch Malware befallen ist oder wiederholt die Anmeldung aufgrund von falschen Anmeldedaten fehlgeschlagen ist.
Der Server besitzt hierfür ein RSA Schlüsselpaar, wodurch man Nachrichten zum Server schicken kann, ohne dass diese entschlüsselt werden können. Der Public-Key ist in der Clientanwendung enthalten, dadurch ist es dem Client möglich eine Randomzahl zu generieren, diese mit dem Public-Key zu verschlüsseln und an den Server zu senden. Der Server entschlüsselt dann die Randomzahl mit dem Private-Key. Außerdem ist die Randomzahl auch noch mit einem aus dem Clientpasswort erstellten Hash verschlüsselt. Dies dient zur Authentifizierung des Clients. Hierfür hat der Server alle Hashes der Clientpasswörter. Diese Randomzahl wird dann als gemeinsamer geheimer Schlüssel zum Austausch von Sperranforderungen und Sperranfragen benutzt.

Kommunikation über VPN:
Die restliche Kommunikation findet über VPN statt. Da es sich um Anbindungen von einzelnen Geräten an ein vorhandenes Netzwerk handelt, bietet sich SSL-VPN an. Außerdem sollen die Angestellten auch aus z.B. Hotels etc. ihre Daten einsehen können. Da wie in 2.3 beschrieben bei SSL-VPN der Port 443 benutzt wird und der Port in den meisten Netzwerken freigegeben ist, ist dies ein weiterer Grund SSL-VPN zu verwenden.

IPSec VPN könnte man zwar auch nutzen, aber es hat den Nachteil, dass es bei NAT(Network Address Translation) zu Problemen führen kann, da NAT die IP des Datenpakets manipuliert und es dadurch als ein verfälschtes Paket erkannt werden würde. Deshalb wäre hier ein höherer Konfigurationsaufwand auf der Serverseite nötig.

Eine alternative zu VPN wäre es, sämtliche Kommunikation zwischen Server und Client mit einem eigenen verschlüsselten Kommunikationsprotokoll zu implementieren. Da es aber bereits fertige VPN-Lösungen gibt, wäre das ein unnötiger Mehraufwand.

Außerdem bietet SSL-VPN weitere Möglichkeiten wie z.B. die Verwaltung von Zugriffsrechten und Authentifizierung von Client/Server.
SSL-VPN stellt, wie die anderen VPN-Arten auch, die Integrität und die Vertraulichkeit der Daten sicher.

Verschlüsselung der Daten:
Ein weiterer Aspekt des Konzepts ist die Verschlüsselung. Hierbei kommen die in 2.2 genannten Verschlüsselungsarten in Frage.
Die Vorteile der asymmetrischen Verschlüsselung sind, dass man eine Signatur und eine Verschlüsselung bekommt. Diese Verfahren stellen die Integrität, die Authentifikation und die Vertraulichkeit der Daten sicher. Außerdem werden weniger Schlüssel benötigt, da jeder nur seinen Public-Key veröffentlichen muss und damit ihn jeder nutzen kann.
Eines der Nachteile ist die Länge des benötigten Schlüssels. Da diese Verschlüsselung auf mathematischen Problemen basiert, müssen möglichst große Zahlen als Schlüssel genommen werden, damit man diese auch mit genügend Rechenkraft nicht knacken kann.
Einer der großen Vorteile der symmetrischen Verschlüsslung ist die kurze Schlüssellänge. Hierbei reichen meist Schlüssel von 64-256 Bit, im Vergleich dazu das asymmetrische Verfahren RSA braucht Schlüssellängen von Minimum 1024 Bit.
Der größte Nachteil hingegen ist die Anzahl der benötigten Schlüssel.
Bei einer Kommunikation zwischen 2 Nutzer braucht man 1 Schlüssel den sich diese beiden teilen. Bei 3 Nutzern die unter einander kommunizieren möchten, braucht man schon 3 Schlüssel, bei 6 Nutzern sind es bereits 15 Schlüssel, wenn jeder vertraulich mit einem anderen Nutzer kommunizieren möchte.
Ein weiterer Nachteil ist hierbei auch der Schlüsselaustausch, da dieser nicht ohne weiteres über eine unverschlüsselte Leitung ausgetauscht werden kann.

Da bei diesem Konzept die Kommunikation durch VPN abgedeckt wird und die Daten, die verschlüsselt werden sollen immer nur Lokal auf dem Smartphone vorliegen, überwiegen die Vorteile der symmetrischen Verschlüsselung. Lediglich für die Sperranfragen und Sperraufforderungen wird RSA verwendet.
Als Verschlüsselungsalgorithmus könnte man AES oder Tripple DES verwenden.
Der DES Algorithmus wird hier nicht verwendet, da er nicht mehr als sicher gilt.
Da 3DES den DES Algorithmus 3-mal durchlaufen muss, ist es zwar sicher jedoch nicht performant.
Der AES Algorithmus ist wesentlich performanter als der 3DES und außerdem verfügt der AES Algorithmus über Schlüssellängen von bis zu 256 Bit. [NAD2005]
Deshalb wird in dieser Arbeit für das Verschlüsseln von Dateien der AES Algorithmus verwendet. Die Daten auf den Smartphone werden mit einem von dem Login-Passwort des VPN Zugangs abgeleiteten Schlüssels verschlüsselt. Die Verschlüsselung der Daten soll die Vertraulichkeit der Daten sicherstellen, auch wenn das Smartphone verloren/entwendet wurde.

Antivirenschutz:
Bei dieser Arbeit wird von einer generischen Antivirussoftware API ausgegangen.
Hierbei ist es wichtig, dass die Software stets auf den neusten stand ist. Die Antivirensoftware muss Daten vor dem versenden an den Server scannen können. Hier wird zwischen Serverseitigen Scannen mit 2 unterschiedlichen Virenscannern und dem Clientseiten Smartphone Antivirusscanner unterschieden. Die Antivirensoftware ist hierbei ein externes Tool, das sowohl bei dem Client als auch bei dem Server durch Aufrufe eines Externen Dienstes angestoßen wird. Die Aufrufe können entweder über eine spezielle API oder durch Kommandozeilen Befehle abgesetzt werden. Der Server besitzt 2 Virenscanner von unterschiedlichen Herstellern um die Wahrscheinlichkeit eines Fundes zu erhöhen. Die Clientseite hat einen Virenscanner für Smartphones. Dieser soll lediglich die wichtigsten Komponenten bei dem kompletten Scan des Smartphones scannen. Das bedeutet, dass er nicht alle Dateien scannt, sondern nur den Kernel und einige andere wichtige Systemrelevante Dateien.

4.1.1 Schlüsselaustausch

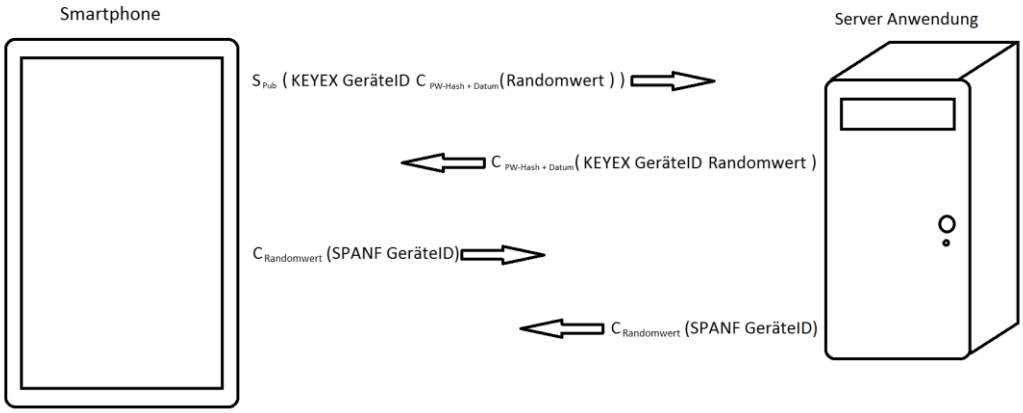

Abbildung 2: Schlüsselaustausch inkl. Sperranfrage

Da die Sperranfragen und Sperraufforderungen über das Internet versendet werden, wird die Kommunikation, die nicht über VPN geschieht, verschlüsselt. Hierfür ist ein Schlüsselaustausch zwischen Client und Server nötig.
Der Schlüsselaustausch wird vom Client initialisiert. Dieser sendet eine KEYEX Nachricht mit der GeräteID und dem random generierten Schlüssel an den Server. Die gesamte Nachricht ist mit dem Public-Key des Servers verschlüsselt. Außerdem wird der Randomwert gesondert nochmal mit einem Hashwert, der aus dem Userpasswort abgeleitet wurde und dem derzeitigen Datum(ohne Uhrzeit) besteht, verschlüsselt. Der Server empfängt die Nachricht, entschlüsselt diese mit dem Private-Key und entschlüsselt den Randomwert. Hierfür hat der Server eine Abbildung von GeräteID auf die abgeleiteten Hashwerte der Passwörter. Zum Schluss sendet der Server die gleiche Nachricht, die er bekommen hat, an den Client. Diese Nachricht ist aber mit dem Userpasswort Hashwert + Datum verschlüsselt. Der Client empfängt diese, entschlüsselt die Nachricht und überprüft ob der Randomwert korrekt ist. Hierdurch haben sich sowohl der Server(durch den Public-Key) als auch der Client(durch das User-Passwort) authentifiziert.

4.1.2 Sicherheit von Server und Client und deren Zusammenarbeit

In diesem Unterkapitel wird der Ablauf der Zusammenarbeit zwischen Client und Server mit Blick auf die Sicherheit erklärt.

Starten der Anwendung

Wenn der Benutzer Dateien bearbeiten oder öffnen möchte, muss er die Clientanwendung auf den Smartphone starten. Dies ist ein Aspekt dieser Arbeit. Es dient zur Sicherstellung der Vertraulichkeit. Unternehmensdaten sind dadurch außerhalb vom Unternehmensnetzwerk nur über die Anwendung lesbar/bearbeitbar. Dateien liegen nur in verschlüsselter Form auf dem Smartphone und sind dadurch bei Verlust des Smartphones oder bei einem Angriff nicht ohne weiteres lesbar.

Schlüsselaustausch, Sperranfragen und Sperraufforderungen

Die Anwendung baut beim Login eine Verbindung zum Server auf. Diese Verbindung geschieht über das Internet und ohne VPN d.h. es ist eine unsichere Leitung. Dies ist nötig da die Einwahl über VPN erst geschehen darf, wenn das Smartphone gescannt und als nicht gefährlich eingestuft wurde. Um sicherzustellen, dass auch die versendeten Nachrichten über die unsichere Leitung nicht mitgelesen oder manipuliert werden, ist eine Verschlüsselung der Nachrichten nötig. Hierfür wird der in 4.1.1 erwähnte Schlüsselaustausch verwendet. Über die unsichere Leitung werden nur Sperranfragen/Sperraufforderungen und KEYEX Nachrichten versendet. Die Sperranfragen werden vom Client gesendet und vom Server beantwortet. Der Client fragt damit nach, ob das Gerät gesperrt wurde, da es evtl. gestohlen wurde oder es wegen Malwarebefall gesperrt ist. Der Server antwortet mit einer Sperranfrage, wenn das Smartphone sich über VPN einwählen darf. Ansonsten wird eine Sperraufforderung versendet. Dies bedeutet, dass das Smartphone alle unternehmensrelevanten Daten löscht und die Anwendung beendet. Sperraufforderungen können außerdem von der Anwendung an den Server versendet werden. Dies geschieht im Fall von mehrfacher falscher Passworteingabe oder einen Fund von Malware auf dem Smartphone.
Bei einer Sperraufforderung löscht der Server den Benutzer den VPN Zugang des Gerätes.

Server- und Clientseitige Malwarescans

Ein weiterer Aspekt dieser Arbeit sind die Malwarescans. Es gibt mehrere Stellen an dem die Malwarescans durchgeführt werden. Eine davon ist der komplette Smartphonemalwarescan bei dem Verbindungsaufbau. Dieser wird durchgeführt, wenn der Client eine Sperranfrage vom Server zurückerhalten hat und sich somit zum Server über VPN verbinden darf. Außerdem werden vom Client verschickte Dateien gescannt. Der Client scannt diese vor dem Verschicken an den Server. Der Server scannt die empfangene Datei mit den zwei unterschiedlichen Malwarescannern. Die Malwarescans sollen sicherstellen, dass der Server so sicher wie möglich vor Malware ist, da dieser sich im Unternehmensnetz befindet.

Betriebt über VPN
Nachdem der Verbindungsaufbau mit Sperranfragen und Malwarescans abgeschlossen ist, wird die VPN Verbindung aufgebaut. Ab diesen Punkt geschieht die gesamte Kommunikation über VPN. Da die Sperranfragen über einen separaten Port versendet wurden, muss sich der Client nun nach dem VPN Login identifizieren. Dies geschieht über eine HALLO Identifizierungsnachricht, die die ID des Smartphones enthält. Nach der HALLO Nachricht kann der Benutzer nun Dateien vom Server anfordern oder auf dem Server speichern. Da hier die Kommunikation über VPN geschieht, ist ein Verschlüsseln der Nachrichten/Daten nicht mehr nötig. Auf dem Server müssen außerdem die Zugriffsberechtigungen richtig konfiguriert werden, damit Benutzer nur auf ihre eigenen Dateien zugreifen können.

4.1.3 Vergleich der BYOD Ansätze

In diesem Abschnitt werden die in 2.1.2 beschriebenen Ansätze sowie das in dieser Arbeit vorgeschlagene Konzept verglichen und bewertet.

Die Rechteabtretung wiederspricht in einigen Aspekten den Bring your own device Konzept, wird aber dennoch oft angewendet, da es z.Z. an sicheren Lösungen für dieses Problem mangelt. Das Problem hierbei ist ebenfalls eine Sicherheit zu gewährleisten, dass das Smartphone nicht von Schadsoftware befallen ist, wenn es sich zum Unternehmensnetzwerk verbindet. Es muss also auch hier ein Sicherheitskonzept her, dass sicherstellt, dass die Smartphones nicht das Unternehmen Netzwerk schädigen können. Außerdem muss es einen Mechanismus zur Überprüfung der Nutzung des Smartphones geben, da es zum Teil oder sogar ganz dem Unternehmen gehört.

Die Cloudservices haben den Vorteil, dass die Angestellten nur bestimmte Software auf ihr eigenes Smartphone installieren müssen und es im Unternehmen dann verwenden können. Es erfordert ein hohes Maß an Verwaltung, da die Profile für jeden Angestellten einzeln konfiguriert werden müssen. Ein weiteres Problem der Profile ist auch dass Überstunden berücksichtigt werden müssen. Nach dem Ende der regulären Arbeitszeit würde das Smartphone es hier nicht erlauben die Unternehmensdaten einzusehen bzw. die Unternehmensservices aus der Cloud anzusprechen. Also müsste es hierfür eine Möglichkeit geben die Arbeitszeit zu erweitern. Dies hätte aber wiederrum zur Folge, dass im Endeffekt die Profile variabel gehalten werden müssten oder z.B. Überstunden genehmigt und auf den Smartphone von einem Admin freigeschaltet werden müssten. Ein weiterer Nachteil ist hierbei auch, dass man einen Cloudservice entweder selber zu Verfügung stellen oder ihn anmieten muss. Die Konfiguration des Services ist ebenfalls aufwendig.

Bei der kompletten Verschlüsselung des Speichermediums ist das Problem, dass jeder Zugriff auf Daten mehr Rechenzeit benötigt, da erst verschlüsselt oder entschlüsselt werden muss. Dies hat wiederrum zur Folge, dass sich die Batterie schneller entlädt, da mehr gerechnet werden muss. Außerdem wird das gesamte Speichermedium verschlüsselt also auch private Daten, was eigentlich unnötig wäre und im Endeffekt wieder die Batterie des

Smartphones unnötig entlädt. Außerdem muss hier ebenfalls ein Sicherheitskonzept her, da mit der kompletten Verschlüsselung der Daten nur sichergestellt ist, dass diese bei Verlust/Diebstahl nicht lesbar sind. Es ist durchaus denkbar, dass Schadsoftware zur Laufzeit Dateien auslesen oder sogar manipulieren könnte. Des Weiteren müsste man auch hier ein VPN-Protokoll zusätzlich nutzen damit die Kommunikation zwischen Smartphone und Server nicht mitlesbar bleibt.

Die erste Art der Virtualisierung aus Kapitel 2.1.2 hat natürlich den großen Vorteil, dass es keinerlei Daten auf den Smartphone gibt, da alles auf den Server bedient wird. Das ist aber auch ein Nachteil, denn bei einer schlechten Internetverbindung wird die Bedienung unzumutbar, da einfache Interaktionen wie eine Datei öffnen oder einen Text schreiben erst an den Server gesendet und dort ausgeführt werden müssen. Außerdem muss man für jeden Benutzer eine eigene VM auf dem Server eingerichtet werden. Diese VMs müsste speziell für den Smartphonegebrauch angepasst werden.
Die zweite Art der Virtualisierung hat den Vorteil, dass hier klar zwischen persönlichen und geschäftlichen Daten getrennt wird. Der Benutzer hat kaum Einschränkungen im persönlichen Modus. Das Problem dieses Konzepts ist, dass viele Smartphones interne VMs nicht ohne weiteres unterstützen. Das bedeutet, dass eine veränderte Firmware auf den Smartphone installiert werden müsste.
Insgesamt gesehen ist VM ein gutes und sicheres Konzept, aber mit sehr hohen Konfigurationsaufwand und Kompatibilitätsproblemen.

Das Konzept dieser Arbeit erlaubt es den Benutzer sein eigenes Smartphone zu benutzen ohne die kompletten Rechte dafür abtretten zu müssen. Das Unternehmen kann zwar bei Verlust oder Diebstahl des Smartphones Daten löschen lassen, aber nur die Unternehmensdaten, d.h. die privaten Daten auf dem Smartphone bleiben erhalten, falls es wiedergefunden wird. Auch bei Schadsoftwarebefall werden nur die Unternehmensdaten gelöscht und das Smartphone bleibt weiterhin für den Privatgebrauch benutzbar. Des Weiteren erlaubt dieses Konzept dem Benutzer Einsicht in die Unternehmensdaten von überall aus wo der Benutzer Internetzugang hat. Dadurch dass hier VPN genutzt wird, ist die Datenübertragung sicher.

4.1.4 Risiko- und Bedrohungsanalyse

In diesem Abschnitt wird eine Risiko- und Bedrohungsanalyse zum Sicherheitskonzept erstellt und ausgewertet.

Als erstes müssen die zu schützenden Daten und Komponenten identifiziert werden. In diesem Fall sind es die Unternehmensdaten, die die Benutzer zum Bearbeiten und Lesen anfordern. Außerdem zählen hier auch die Zugangsdaten der Benutzer für VPN dazu. Als nächstes muss eine Übersicht über die Sicherheitsarchitektur erstellt werden. Diese ist bereits ab Anfang des Kapitels 4.1 beschrieben.

Darauf folgend werden mit Hilfe der STRIDE(Spoofing Tampering Repudiation Infomation disclosure Denial of service Elevation of privilage) Bedrohungsanalyse die 3 Komponenten Netzwerk, Host(Server) und Anwendung(Client) auf die wichtigsten Bedrohungen untersucht. Diese Bedrohungen sind die obengenannten Schlagwörter:

Spoofing: Sich als jemand anders ausgeben, um an Informationen zu kommen.

Tampering with Data: Daten zerstören oder manipulieren.

Repudiation: Nachweis, dass z.B. jemand eine Datei bearbeitet hat.

Information disclosure: Zugangsberechtigungen, sodass unbefugte keinen Zugang zu Dateien haben.

Denial of service: Service unzugänglich machen durch z.B. zu viele offene Verbindungen.

Elevation of privilege: Zugangsberechtigungen durch Angriffe erhöhen. Wenn zum Beispiel ein normaler User sich durch einen Angriff Admin Rechte auf dem System verschafft.

Nach der STRIDE Analyse folgt im Anschluss die DREAD (Damage Reproducibility Exploitability Affectability Discoverbility) Analyse. Die einzelnen Schlagworte bedeuten:

Damage: Wie schlimm wird der entstandene Schaden sein?

Reproducibility: Wie einfach ist es den Angriff zu reproduzieren?

Exploitability: Wie einfach ist dieser Angriff? (z.B. kann man ihn als Programmieranfänger ausführen oder benötigt man tiefe Kenntnisse in bestimmten Systemen)

Affectability: Wie viele Benutzer sind betroffen?

Discoverbility: Wie einfach ist es diesen Bug/Exploit zu finden?

Bei dieser Analyse wird jedes der Schlagworte mit einer Bewertung zwischen 1 bis 3 bewertet, wobei 1 das niedrigste Risiko ist und 3 das höchste. Anschließend wird die Summe daraus gebildet. Eine Bewertung von 5-7 ist ein geringes Risiko, 8-11 mittleres Risiko und 12-15 hohes Risiko. [MS2014]

Bedrohungen für das Netzwerk:
Das Sammeln von Informationen wird bei diesem Konzept eingeschränkt indem nur 1 Port außerhalb des VPNs offen ist. Dieser dient für Sperrungsanfragen und Sperraufforderungen. Das bedeutet es sollten alle anderen Ports und nicht benötigten Protokolle gesperrt bzw. abgeschaltet werden, damit man Angreifern so wenig Angriffsfläche wie möglich bietet. Außerdem sollte man, so weit möglich, den Server so konfigurieren, dass dieser so wenige Informationen wie nur möglich über das Betriebssystem und verwendete Software preisgibt.
Das Abfangen und Abhören der Kommunikation für einen Angreifer ist wegen VPN nur sehr schwer möglich. Da VPN sämtlichen Netzwerkverkehr verschlüsselt, müsste der Angreifer den Schlüssel der VPN Sitzung kennen um an die Daten zu kommen.
Das Spoofing von IP-Adressen wird hier nicht direkt betrachtet, da das Konzept nicht die Sicherheit des internen Netzes betrachtet, sondern nur den Zugang der Unternehmensdaten durch VPN.
Man-in-the-middle Angriffe wären hierbei möglich, wenn der Angreifer Zertifikate fälschen kann, so dass der Benutzer denkt, er verbindet sich mit dem Server.
Hierbei müssen die Benutzer geschult werden, nicht jedes Zertifikat zu akzeptieren, sondern darauf zu achten, dass es das richtige Zertifikat ist.
DDoS Angriffe werden hier nicht explizit verhindert. Hierfür müsste der Server über Intrusion Detection Software verfügen um solche Angriffe zu erkennen und sich zu schützen.

Bedrohungen für den Host:
Der Schutz gegen das eindringen von Trojanern, Würmern und Viren ist eine der Prioritäten dieses Konzepts. Hier wird auf Serverseite mit 2 unterschiedlichen Antivirenscannern eingehende Dateien gescannt. Außerdem müssen hierfür Betriebssystem, Antivirensoftware und sämtliche andere verwendete Software stets auf dem neuesten Stand sein, damit Sicherheitslücken schnellstmöglich geschlossen werden.
Wie bei Bedrohungen für das Netzwerk beschrieben, sollte das Sammeln von Informationen, so weit es geht, unterbunden werden und nur die benötigten Ports und Protokolle verwendet werden.
Die Benutzerkonten für VPN Zugänge müssen sicher sein. Das Passwort soll wie in Anforderung 1.6 beschrieben erstellt werden. Nach 5 fehlgeschlagenen Anmeldeversuchen soll die ID/Benutzername für VPN gesperrt werden, um das Entschlüsseln des Passwortes zu verhindern.
Für DDoS Angriffe gilt hier das gleiche wie bei Bedrohungen für Netzwerk beschrieben.
Das Ausführen von „eingeschleusten“ Codes ist hier nicht möglich, da weder Server noch Anwendung Code ausführt, der versendet wird. Es werden lediglich Dateien und Befehle, die dem Kommunikationsprotokoll entsprechen, verschickt.
Für die Zugriffsrechte auf dem Fileserver ist der File Server zuständig. Der Server selbst fragt beim File Server an, ob der Benutzer die Berechtigungen hat, sollte dies der Fall sein, fordert der Server die Datei beim File Server an.

Bedrohungen für die Anwendung:
Der Client verwendet ebenfalls VPN, wodurch Abhören und Abfangen von Daten nicht ohne weiteres möglich sind.
Bruteforce und Wörterbuchangriffe zur Entschlüsselung des Passwortes werden nach 5 Anmeldeversuchen unterbunten, sowie durch die Passwortrichtlinien aus Anforderung A1.6 erschwert. Die Passwörter müssen als Hash gespeichert werden. Dieser sollte einen „salt“ enthalten und nicht wieder rückgängig errechenbar sein.
Replayangriffe sind ebenfalls durch VPN nur sehr schwer erreichbar und selbst wenn diese stattfinden, würden nur Anfragen oder Dateien neu gesendet werden, da der Angreifer die Daten nicht entschlüsselt hat, sondern einfach nur erneut an den Server sendet. Die Berechtigungen für Benutzung von Ressourcen sind durch Androids Dalvik Engine bereits gegeben. Hierbei darf die Anwendung nur die Ressourcen anfordern, die es bei der Installation angegeben hat. Außerdem können andere Prozesse nicht ohne weiteres von der Anwendung abgefangen oder manipuliert werden.
Der Zugriff auf die Daten, die die Anwendung verwendet, ist durch einen vom VPN-Zugangspasswort abgeleiteten Schlüssel verschlüsselt. Hierfür wird eine SHA-256 Hashfunktion verwendet, wodurch gegeben ist, dass selbst wenn man an den Schlüssel kommt, das Passwort nicht rückgängig errechenbar ist. Die verschlüsselten Daten werden mit dem AES Algorithmus verschlüsselt und können nicht mit vertretbarem Aufwand ohne den Schlüssel wieder entschlüsselt werden.
Zugriffe auf Dateien werden protokoliert, um bei Missbrauch der Daten den Verursacher feststellen zu können. Außerdem ist ein Malwarebefall des Smartphones ebenfalls denkbar.
Als nächstes folgt nun die Risikoanalyse nach der DREAD Methode.

Bewertung der Bedrohungen für das Netzwerk:

Bedrohung	D	R	E	A	D	Gesamt	Bewertung
Sammeln von Daten bzw. Footprinting	2	3	3	3	3	14	Hoch
Abfangen und Abhören der Kommunikation	3	1	1	3	1	9	Mittel
Man-in-the-middle Angriff	3	3	1	3	3	13	Hoch
DDoS	2	3	3	2	3	13	Hoch
Replay Angriff	1	1	1	1	2	6	Niedrig

Tabelle 1: Bewertung der Bedrohungen für das Netzwerk

Ziel der Bedrohung:	Sammeln von Daten bzw. Footprinting
Risikobewertung:	Hoch
Angriffstechniken:	Sammeln von Daten über Serversoftware(Betriebssysteme etc.)
Gegenmaßnahme:	Server so konfigurieren, dass dieser nur die Informationen rausgibt die auch wirklich benötigt werden. Außerdem sollte man nicht verwendete Protokolle und Ports sperren.

Ziel der Bedrohung:	Abfangen und Abhören der Kommunikation
Risikobewertung:	Mittel
Angriffstechniken:	Abhören des Netzwerkverkehrs durch Tools wie Wireshark etc.
Gegenmaßnahme:	Da hier bereits VPN benutzt wird, ist es für einen Angreifer nur sehr schwer die verschlüsselte Kommunikation abzuhören und zu entschlüsseln.

Ziel der Bedrohung:	Man-in-the-middle Angriff
Risikobewertung:	Hoch
Angriffstechniken:	Fälschen von Zertifikaten um den Benutzer vorzutäuschen, dass der Angreifer der Server sei.
Gegenmaßnahme:	Schulung der Benutzer, dass diese nicht jedes Zertifikat akzeptieren, sondern nur die Unternehmenszertifikate annehmen.

Ziel der Bedrohung:	DDoS Angriff
Risikobewertung:	Hoch
Angriffstechniken:	Bot Netzwerk bzw. viele Rechner dafür nutzen Anfragen an den Server zu senden und diese nicht mehr zu beantworten. Meist wird dafür eine TCP Anfrage gestellt und diese dann nicht mehr beantwortet damit die TCP-Verbindung offen bleibt und Ressourcen belegt.

Gegenmaßnahme:	IDS auf den Server installieren und das Timeout für TCP Verbindungen niedriger einstellen.
Ziel der Bedrohung:	Replay Angriff
Risikobewertung:	Niedrig
Angriffstechniken:	Unter bestimmten Voraussetzungen lässt sich ein Replay Angriff auf VPN ausführen. Hierbei muss der Server dieselbe Nounce und dieselbe SessionID für die Verbindung haben. In diesem Fall kann der Angreifer die gleichen Datenpakete nochmal schicken, die der Benutzer vorher geschickt hat.
Gegenmaßnahme:	Keine notwendig, da das Ausmaß des Schadens relativ gering ist und die Wahrscheinlichkeit dieses Angriffs sehr gering ist.

Bewertung der Bedrohungen für den Host:

Bedrohung	D	R	E	A	D	Gesamt	Bewertung
Eindringen von Schadsoftware	3	3	2	3	2	13	Hoch
Entschlüsseln der Passwörter	3	1	1	3	1	9	Mittel
Code Injection	3	1	1	3	1	9	Mittel
Zugriffsrechte	2	3	1	2	2	10	Mittel
Unbefugte Sperranfragen/Sperraufforderungen	2	2	2	3	1	10	Mittel

Tabelle 2: Bewertung der Bedrohungen für den Host

Ziel der Bedrohung:	Eindringen von Schadsoftware
Risikobewertung:	Hoch
Angriffstechniken:	Einschleusen von Schadsoftware durch Dateien von Benutzer
Gegenmaßnahme:	Wie im Konzept beschrieben, wird jede eingehende Datei mit 2 Virenscannern gescannt. Außerdem muss das Betriebssystem und die verwendete Software stets auf dem neuesten Stand sein um Sicherheitslücken zu schließen.

Ziel der Bedrohung:	Entschlüsseln der Passwörter
Risikobewertung:	Mittel
Angriffstechniken:	Bruteforce, Wörterbuchangriffe oder Social Engineering
Gegenmaßnahme:	Die im Konzept vorgeschlagenen Passwortrichtlinien bieten bereits ein hohes Maß an Sicherheit. Nichtsdestotrotz sollte man die Benutzer schulen welche Art von Passwörter sicher sind um auch Social Engineering zu unterbinden.

Ziel der Bedrohung:	Code Injection
Risikobewertung:	Mittel
Angriffstechniken:	Schädlichen Code in die Anwendung einschleusen.
Gegenmaßnahme:	In diesem Konzept gibt es keine direkte Stelle wo man Code Injection betreiben könnte. Der Client und der Server kommunizieren über ein eigenes Protokoll, das nur bestimmte Befehle akzeptiert und diese ausführt. Die Dateien die vom und zum Server gehen werden gescannt und verschickt, es folgt hier auch keine Ausführung von Code der nicht zur Anwendung gehört

Ziel der Bedrohung:	Zugriffsrechte
Risikobewertung:	Mittel
Angriffstechniken:	Benutzer versuchen auf Dateien zu zugreifen die nicht für diese gedacht sind.
Gegenmaßnahme:	Ausführliche Rechteverwaltung auf dem Fileserver. Der Server fragt beim Fileserver nur an ob der Benutzer die Rechte für den Zugriff auf die Datei hat. Der Fileserver muss die Rechte prüfen.

Ziel der Bedrohung:	Unbefugte Sperranfragen/Sperraufforderungen
Risikobewertung:	Mittel
Angriffstechniken:	Angreifer versucht das Sperren eines Smartphones zu erreichen oder die GeräteID rauszufinden
Gegenmaßnahme:	Als erster Schritt der Kommunikation zwischen Client und Server wird ein Schlüssel vom Client generiert und mit dem Public-Key des Servers verschlüsselt. Mit dem generierten Schlüssel wird dann sämtliche Kommunikation zwischen Client und Server verschlüsselt.

Bewertung der Bedrohungen für die Anwendung:

Bedrohung	D	R	E	A	D	Gesamt	Bewertung
Zugriff auf Ressourcen	2	1	1	1	1	6	Niedrig
Fremdzugriff auf lokale Daten des Smartphones	3	3	1	3	3	13	Hoch
Interner Angriff	3	3	3	3	3	15	Hoch
Malwarebefall des Smartphones	2	3	1	1	2	9	Mittel
Wörterbuchangriff/Bruteforce auf Dateien	3	2	2	1	3	11	Mittel
Falsche Sperraufforderung	1	3	1	1	1	7	Niedrig

Tabelle 3: Bewertung der Bedrohungen für die Anwendung

Ziel der Bedrohung:	Zugriff auf Ressourcen des Smartphones
Risikobewertung:	Niedrig
Angriffstechniken:	Versuchen die Anwendung dazu zu bringen Ressourcen des Smartphones zu benutzen, die nicht dafür gedacht sind.
Gegenmaßnahme:	Keine, da die Dalvik Engine es bereits unterbindet.

Ziel der Bedrohung:	Fremdzugriff auf lokale Daten des Smartphones
Risikobewertung:	Hoch
Angriffstechniken:	Diebstahl des Smartphones. Nach der Entwendung des Smartphones wird versucht auf die Unternehmensdaten zuzugreifen.
Gegenmaßnahme:	Die Daten liegen in einer verschlüsselten Form vor. Der Angreifer kann diese ohne den vom Benutzer Passwort abgeleiteten Schlüssel nicht ohne vertretbaren Aufwand einsehen.

Ziel der Bedrohung:	Interner Mitarbeiter versucht dem Unternehmen zu Schaden
Risikobewertung:	Hoch
Angriffstechniken:	Mitarbeiter veröffentlicht oder manipuliert böswillig Dateien
Gegenmaßnahme:	Ausführliche Protokollierung von Schlüsselereignissen wie Login, Zugegriffene Dateien und versuchte aber fehlgeschlagene Zugriffs oder Login Versuche.

Ziel der Bedrohung:	Malwarebefall des Smartphones
Risikobewertung:	Mittel
Angriffstechniken:	Smartphone wird durch Malware befallen
Gegenmaßnahme:	Schulung des Personals und vor jeder Verbindung ein kompletter Malwarescan des Smartphones.

Ziel der Bedrohung:	Wörterbuchangriff/Bruteforce auf Dateien
Risikobewertung:	Mittel
Angriffstechniken:	Anmeldeversuche durch gestohlenes/unbefugtes Smartphone
Gegenmaßnahme:	Maximal 5 Anmeldeversuche erlauben, wodurch dann Wörterbuchangriffe und Bruteforce unterbunden wird.

Ziel der Bedrohung:	Falsche Sperraufforderungen
Risikobewertung:	Niedrig
Angriffstechniken:	Angreifer versucht eine Sperraufforderung an das Smartphone zu senden um es zu sperren.
Gegenmaßnahme:	Die Kommunikation zum Server wird durch einen ausgehandelten Schlüssel verschlüsselt. D.h. dem Angreifer muss dieser Schlüssel bekannt sein damit er die Sperraufforderung verschlüsseln kann und der Client diese Akzeptiert. Außerdem muss dem Angreifer die GeräteID bekannt sein.

4.2 Entwurf der Anwendung

In diesem Kapitelabschnitt wird nun eine Umsetzung des in 4.1 ausgearbeiteten Konzepts vorgeschlagen.

4.2.1 Architektur

Die hier vorgeschlagene Architektur zur Umsetzung des Konzepts ist eine Client-Server Architektur. Der Client ist die App auf dem Smartphone und der Server ist eine Art Applicationfirewall mit zusätzlichen Features wie z.B. dem Datentransfer. Der Client und der Server kommunizieren über VPN miteinander. (Siehe Abbildung 1). Die Kommunikation findet innerhalb des VPN Netzes über TCP statt. Hier wurde TCP gewählt, da hier Dateien übertragen werden und diese fehlerfrei beim Endpunkt ankommen sollen. Außerdem werden Sperranfragen und Sperraufforderungen über einen separaten Port beim Server empfangen. Die Komponenten bieten mindestens ein Interface für die Benutzung der Komponente an. Intern nutzen die Komponenten ebenfalls nur Interfaces z.B. eine Manager Klasse nutzt nicht direkt die Adapter Klasse sondern nur das Interface des Adapters. Alle Methoden der Klassen, die nicht das Interface der Komponente implementieren, sind nur Komponenteninternen benutzbar und nicht nach Außen hin sichtbar.

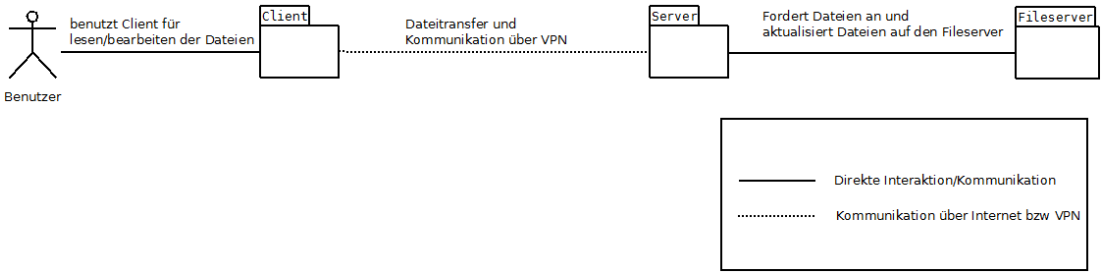

Abbildung 3: Kontextsicht

Der Client und der Server kommunizieren über ein speziell für diesen Fall entworfenes Protokoll. Das Protokoll zur Kommunikation besteht aus 5 ASCII-Zeichen langen Befehlen gefolgt von der angefragten/benötigten Information.
Nun folgt eine Beschreibung des Kommunikationsprotokolls für die Kommunikation zwischen Client und Server:

Befehl: SPANF <GeräteID>
Sender: Client
Empfänger: Server
Mögliche Antworten: SPAUF <GeräteID>, SPANF <GeräteID>
Erklärung:
Eine Sperranfrage vom Smartphone. Sie besteht aus dem Schlüsselwort SPANF und der GeräteID. Die GeräteID wird bei der Einrichtung des Smartphones, wie in 4.1.2 beschrieben, erstellt. Der Server Antwortet mit SPAUF <GeräteID> oder SPANF <GeräteID>. SPAUF bedeutet, dass das Gerät gesperrt ist und SPANF <GeräteID>bedeutet, dass das Gerät verwendet werden kann.

Befehl: SPAUF <GeräteID>
Sender: Client
Empfänger: Server
Mögliche Antworten: SPAUF <GeräteID>
Erklärung:
Eine Sperraufforderung des Smartphones bei Schadsoftwarefund oder mehrfacher falscher Login Dateneingabe. Der Server kann nur mit SPAUF <GeräteID> antworten, auch wenn das Gerät bereits gesperrt ist.

Befehl: HALLO <GeräteID>
Sender: Client
Empfänger: Server
Mögliche Antworten: HALLO <GeräteID>, ERROR <Fehlerbeschreibung>
Erklärung:
Eine Hallo Nachricht zum initialisieren der Kommunikation zwischen Client und Server. Der Server antwortet mit HALLO und der Smartphone GeräteID oder ERROR bei einem Fehler.

Befehl: FILER <Pfad>
Sender: Client
Empfänger: Server
Mögliche Antworten: FILES <Datei>, ERROR <Fehlerbeschreibung>
Erklärung:
Eine Anfrage des Smartphones für eine Datei. Die Pfadangabe ist relativ zum Arbeitsverzeichnis des Benutzers. Der Server schickt den Befehl FILES gefolgt von der Datei oder eine ERROR Meldung mit einem Fehlertext(z.B. nicht genügend Berechtigungen).

Befehl: FILES <Pfad> <Datei>
Sender: Client
Empfänger: Server
Mögliche Antworten: FILER <Pfad>,SPAUF<GeräteID>,
ERROR <Fehlerbeschreibung>
Erklärung:
Das Smartphone sendet eine bearbeitete Datei an den Server um diese dort zu aktualisieren. Der Server antwortet nach Erhalt der Datei mit FILER <Pfad>, wenn er die Datei fehlerfrei bekommen hat und diese unter <Pfad> abgespeichert/aktualisiert hat. Alternativ kann er mit ERROR antworten und die Fehlermeldung weitergeben. Bei fehlerhafter Übertragung, wird ein ERROR ausgegeben und die Datei wird neu vom Smartphone gesendet. Eine SPAUF <GeräteID> wird nur dann gesendet, wenn der Server beim Scannen der erhaltenen Dateien Schadsoftware gefunden hat.

Befehl: FILEL <Pfad>
Sender: Client
Empfänger: Server
Mögliche Antworten: FILEL <Dateiliste>, ERROR <Fehlerbeschreibung>
Erklärung:
Dieser Befehl dient dazu eine Liste der Datei für ein Verzeichnis oder Pfad anzufordern. Der Server schickt eine Liste mit Dateien und Verzeichnissen an den Client zurück.
ERROR kommt zurück wenn z.B. das Verzeichnis nicht existiert oder nicht genügend Berechtigungen für das Verzeichnis vorhanden sind.

Befehl: FILED <Pfad>
Sender: Client
Empfänger: Server
Mögliche Antworten: FILED <Pfad>, ERROR <Fehlerbeschreibung>
Erklärung:
Befehl zum Löschen einer Datei mit dem gegebenen Pfad.

Befehl: CLOSE
Sender: Client/Server
Empfänger: Server/Client
Mögliche Antworten: CLOSE
Erklärung:
Initiiert die Beendigung der Kommunikation.

Befehl: KEYEX <GeräteID><Key>
Sender: Client
Empfänger: Server
Mögliche Antworten: KEYEX <Key>, ERROR <Fehlerbeschreibung>
Erklärung:
Der Client sendet den KEYEX Befehl, da auch die Kommunikation, die nicht über VPN läuft, verschlüsselt sein soll. Dieser Befehl initiiert den Schlüsselaustausch. Der Key ist hierbei ein Randomwert, der für die symmetrische Verschlüsselung der Sperranforderungen benutzt wird. Der Server antwortet mit KEYEX <GeräteID><Key>.

4.2.2 Server

Die Aufgabe des Servers ist es Anfragen der Clients zu empfangen und zu bearbeiten. Der Server hat dafür 2 offene Ports. Einen Port, der über das Internet erreichbar ist, um Sperranfragen und Sperraufforderungen(siehe 4.2.1 SPANF oder SPAUF) zu beantworten. (Anforderung 2.1 und 2.2)
Der andere Port ist nur über VPN, also nur intern im Netzwerk erreichbar. Auf diesem Port bekommt der Server Anfragen für Dateitransfers. Siehe hierzu 4.2.1 FILES und FILER im Kommunikationsprotokoll. Der Server ist außerdem über ein Dateitransferprotokoll wie z.B. FTP an den Fileserver angebunden. (A2.6)

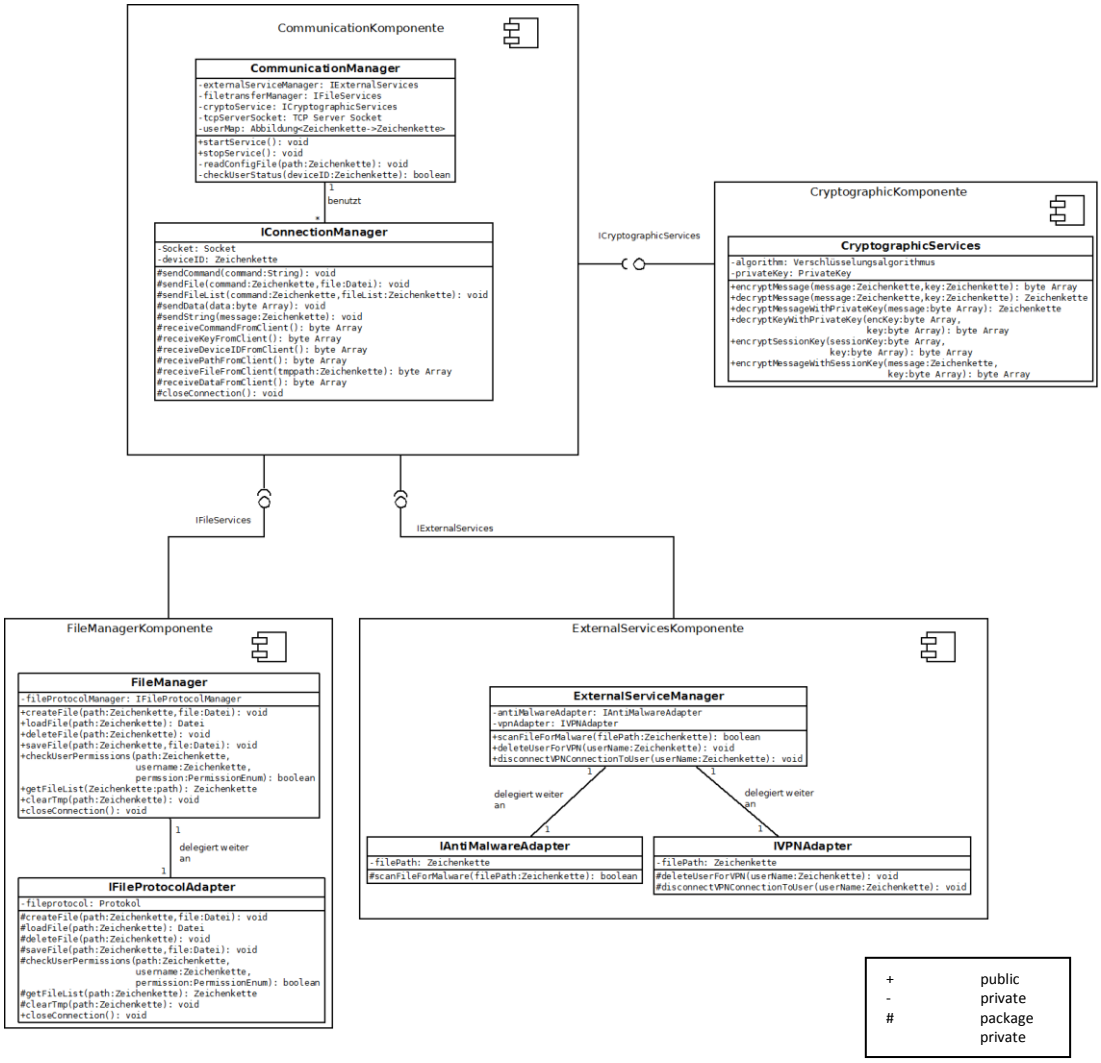

Abbildung 4: Bausteinsicht des Servers

FileManagerKomponente:
Die FileManagerKomponente, die für den Zugriff über das Dateiprotokoll zuständig ist, soll so generisch wie möglich gehalten werden um das Dateitransferprotokoll jederzeit schnell auswechseln zu können. Deshalb gibt es eine FileManager- und FileProtocolAdapterklasse. Der FileManager ist generisch gehalten und bietet die in der Abbildung 4 beschriebenen Operationen. Der FileProtocolAdapter benutzt ein konkretes Dateizugriffsprotokoll zum Fileserver. Die Operationen im FileManager werden an den FileProtocolAdapter weitergeleitet. Dadurch wird von einer konkreten Implementierung eines Dateizugriffsprotokolls abstrahiert und es muss z.B. bei einem Wechsel des Protokolls nur die FileProtocolAdapter Klasse ausgewechselt werden. Die Methode checkUserPermissions überprüft die Benutzerberechtigungen. Dadurch wird sichergestellt, dass Benutzer nur auf Dateien zugreifen, auf die sie auch die Berechtigungen haben. In jeder der vom IFileServices angebotenen Methoden müssen zuerst die Berechtigungen überprüft werden. (A2.7)

ExternalServicesKomponente
Die ExternalServicesKomponente ist für die Aufrufe der Antimalware- und VPN Dienste zuständig. Sie ist in 3 Klassen aufgeteilt:
Die ExternalServiceManager Klasse ist der Verwalter der Komponente. Diese Klasse delegiert die Aufrufe der Komponente an den jeweils dafür zuständigen Dienst.
Einer dieser Dienste ist der AntiMalwareAdapter. Dieser Dienst ist für die Aufrufe des Antimalwaredienstes zuständig. Da die Antimalware Scans über ein externes Programm stattfinden, ruft diese Klasse dieses Programm mit den jeweiligen Parametern auf.
Die andere Klasse dieser Komponente ist der VPNAdapter. Diese Klasse ist für die Verwaltung des VPN Dienstes zuständig. Da der VPN Dienst ebenfalls ein externes Programm ist, ruft diese Klasse die nötigen Operationen auf dem VPN Programm auf.
Beide Klassen lassen sich relativ einfach austauschen, falls ein neues VPN Programm oder eine neue Antimalwaresoftware benutzt werden sollen, da hier nur die IAntiMalwareAdapter und IVPNAdapter Interfaces eingehalten werden sollen.
(A2.5,A2.12 und 2.13)

CryptographicKomponente
Die CryptographicKomponente des Servers wird für die Verschlüsselung der Sperranfragen und Sperraufforderungen benötigt. Sie bietet Methoden zum Ver- und Entschlüsseln von Nachrichten an. Die erste Nachricht für den Schlüsselaustausch wird mit dem Public-Key des Servers verschlüsselt, alle anderen Nachrichten die außerhalb des VPN gesendet werden, werden dann mit den durch den Schlüsselaustausch ausgehandelten Schlüssel verschlüsselt. Die CryptographicKomponente bietet das Interface ICryptographicServices an.

CommunicationKomponente

Die CommunicationKomponente ist die Hauptkomponente des Servers. Hier werden die anderen beiden Komponenten benutzt. Bei der Erstellung der Komponente werden alle nötigen Daten, wie die ID->Benutzername Abbildung, Port-Konfigurationen und alle anderen Einstellungen aus einer Config Datei ausgelesen.

Diese Komponente ist zuständig für die Kommunikation zu den Clients. Sie ist in 2 Klassen aufgeteilt: ConnectionManager und CommunicationManager. Der ConnectionManager ist zuständig für die Kommunikation über TCP. Außerdem bietet die Klasse Methoden zum Senden und Empfangen von Nachrichten des Clients an. Der CommunicationManager besitzt zwei ServerSockets. Sobald ein Client eine Verbindung zum ServerSocket für Sperranfragen aufbaut wird ein ConnectionManager mit dem daraus entstandenen Socket erstellt und ein Thread, der die Kommunikation zum Client abhandelt, wird gestartet. Dieser Thread arbeitet die Sperranfrage des Clients ab. Die direkte Kommunikation geschieht über den ConnectionManager. Der andere ServerSocket ist nur über VPN bzw. aus dem internen Netzwerk erreichbar. Dieser ist für den Datenaustausch zuständig. Das Prinzip hierbei ist dasselbe wie bei dem ersten ServerSocket. Sobald eine Anfrage kommt wird ein Thread gestartet, der die Kommunikation zum Client abhandelt.

Es werden hier auch bei Anfragen für Dateien über den IFileServices Dateien vom FileServer angefordert und an den Client verschickt. Die Berechtigungen werden vor dem Zugriff überprüft. Sollten die Berechtigungen für den Zugriff nicht ausreichend sein, wird eine ERROR <Nachricht> Nachricht an den Client zurück geschickt.

Des Weiteren verwaltet der CommunicationManager eine Abbildung von GeräteID auf VPN-Zugangsnamen. Diese wird bei neuen Geräten manuell hinzugefügt. Bei jedem Hinzufügen wird sofort die Abbildung in eine Configdatei rausgeschrieben und somit aktualisiert. Dadurch wird sichergestellt, dass bei einem Absturz des Servers die Daten nicht verloren gehen.

Außerdem werden dort empfangene Dateien mit dem IExternalServices auf Malware untersucht. Bei Malwarebefall werden auch über IExternalServices die Zugangsdaten des Benutzers gesperrt. Die Klasse CommunicationManager bietet außerdem eine Start und Stop Methode zum Starten und Beenden des Serverdienstes. Die Start Methode startet alle nötigen ServerSockets und Threads und startet somit den Serverdienst.

Die Stop Methode beendet alle laufenden Threads und Verbindungen ordnungsgemäß und der Server Dienst wird somit gestoppt. (A2.3,A2.4,A2.9,A2.10 und A2.11)

4.2.3 Client

Die Clientanwendung initialisiert die Verbindung zum Server. Die Anwendung verwaltet außerdem Dateien, die sich auf den Smartphone befinden. Diese sind verschlüsselt wenn die Datei nicht gerade benötigt wird. Der Client ist in 4 Komponenten aufgeteilt:
Den Dateimanager MobileFileManagerKomponente, der für sämtliche unternehmensrelevanten Datei auf den Smartphone zuständig ist.
Der MobileCryptographicKomponente, die für die Ver- und Entschlüsselung der Dateien zuständig ist.
Der MobileCommunicationKomponente, die die Kommunikation zum Server verwaltet und der MobileExternalServicesKomponente, die hier ebenfalls von den benutzten externen Anwendungen abstrahiert.

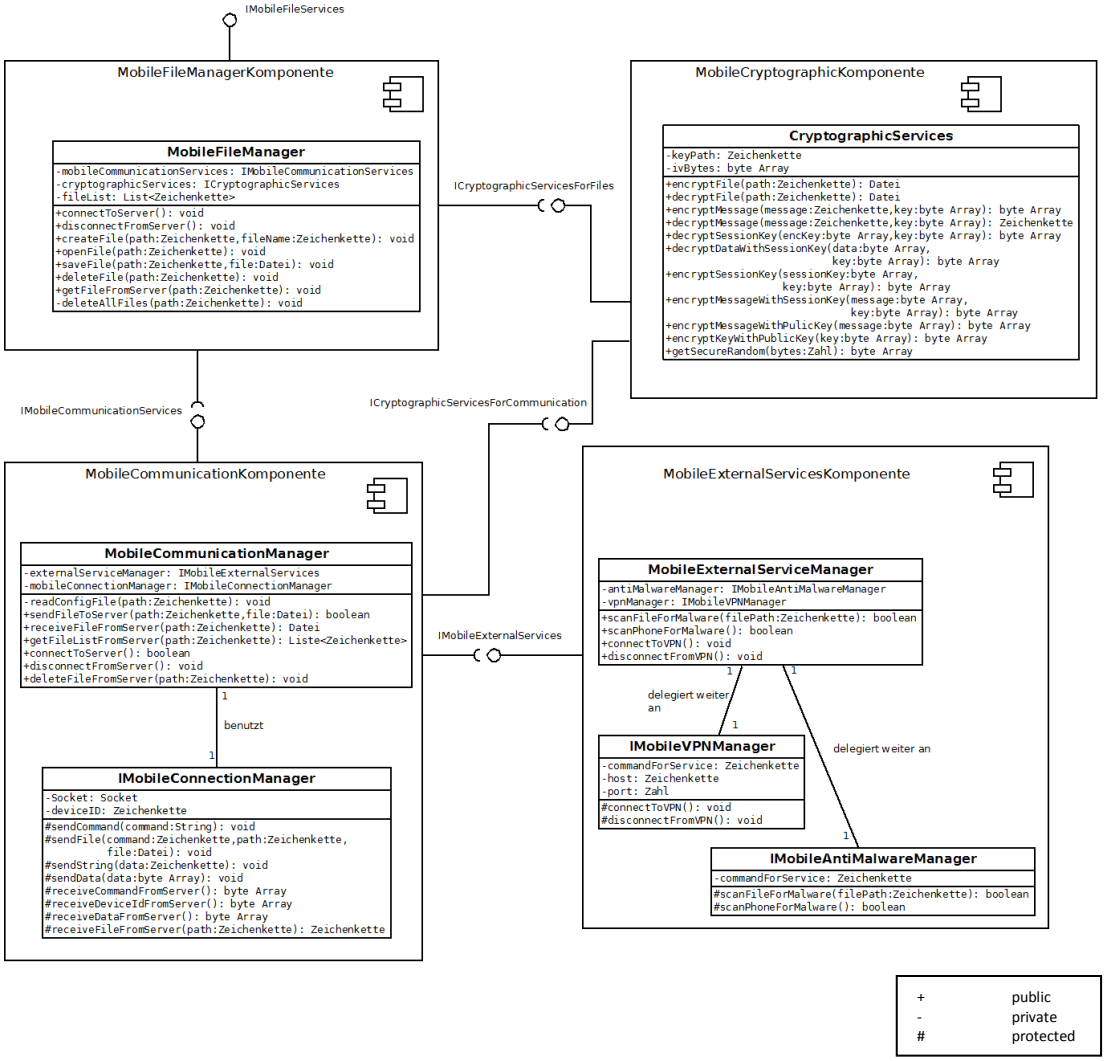

Abbildung 5: Bausteinsicht des Clients

MobileExternalServicesKomponente:
Ähnlich wie die ExternalServicesKomponente des Servers, ist diese Komponente ebenfalls zuständig für Aufrufe von externen Diensten. Die Komponente besteht ebenfalls aus 3 Klassen. Die MobileExternalServiceManager Klasse ist nur ein Verwalter der eigentlichen Dienste und dient zur Abstraktion der einzelnen externen Anwendungen. Auch hier wird für jede externe Anwendung eine Klasse implementiert. Der Verwalter selbst arbeitet nur mit Interfaces, so dass man die Klassen, die für die externen Anwendungen zuständig sind, einfach austauschen kann.
Die Klasse MobileVPNAdapter ist für die Aufrufe zum Aufbau der VPN Verbindung zuständig. Hier wird eine externe Anwendung durch System Calls angesprochen. Anders als beim Server dient der VPN Adapter nur zum Verbinden und Trennen zum VPN. Der MobileAntiMalwareAdapter ist hier ebenfalls für die Aufrufe des AntiMalwaredienstes zuständig. Anders als beim Server müssen nicht nur Dateien sondern auch das komplette Smartphone durch einen Befehl gescannt werden können. (Anforderung 1.3 und 1.6)

Die MobileExternalServicesKomponente ist wie oben beschrieben in 3 Klassen unterteilt.
Die Verwalterklasse muss das Komponenteninterface IMobileExternalServices implementieren. Außerdem muss die Klasse MobileVPNAdapter das Interface IMobileVPNAdapter und die Klasse MobileAntiMalwareAdapter das Interface IMobileAntiMalwareAdapter implementieren.

MobileCommunicationKomponente
Die MobileCommunicationKomponente ist für die Kommunikation zum Server zuständig. Sie initialisiert die Kommunikation und sendet und empfängt Befehle und Dateien. Die Komponente besteht aus 2 Klassen. Die CommunicationManager Klasse abstrahiert von der direkten Kommunikation mit TCP. Sie bietet Methoden zum Senden und Empfangen von Nachrichten. Diese Methoden werden an die ConnectionManager Klasse weiterdelegiert. Außerdem nutzt die CommunicationManager Klasse die IMobileExternalServices Schnittstelle um VPN Verbindungen auf- und abzubauen. Da das Smartphone vor der Verbindung zum VPN gescannt werden soll, stößt sie ebenfalls den Scan an.
Des Weiteren wird jede Datei, die an den Server gesendet werden soll, ebenfalls gescannt. Hierfür nutzt sie ebenfalls die IMobileExternalServices Schnittstelle. DieMobileConnectionManager Klasse ist dem ConnectionManager des Servers ähnlich.
Sie verwendet das TCP Protokoll und kommuniziert direkt mit dem Server. Die Komponente wird über das IMobileCommunicationServices Interface angesprochen. Außerdem nutzt die CommunicationManager Klasse das IMobileConnectionManager Interface. (A1.2 und A1.4)

MobileCryptographicKomponente
Die MobileCrypographicKomponente dient zur Ver- und Entschlüsselung, der sich auf dem Smartphone befindenden Dateien. Sie besteht aus einer Klasse:
CryptographicServices, die für die Ver- und Entschlüsselung der Dateien zuständig ist.
Die Komponente bietet das ICryptographicServicesForFiles und ICryptographicServicesForCommunication Interface an. Das ForFiles Interface bietet Methoden zur Ver- und Entschlüsselung von Dateien an. Es wird von der MobileFileManagerKomponente verwendet. Das ForCommunication Interface dient zum Ver- und Entschlüsseln von Sperranfragen und Sperraufforderungen. Dieses Interface wird von der MobileComminicationKomponente benutzt. Diese müssen verschlüsselt werden, da sie über das Internet ohne VPN versendet werden. Da für die Implementierung des Clients Java benutzt werden muss, werden für sämtliche kryptographischen Anwendungen java.security und java.crypto Libraries verwendet.
(A1.8, A1.13,A1.15,A1.16 und A1.18)

MobileFileManagerKomponente
Die MobileFileManagerKomponente ist zuständig für die Dateien und ist somit die zentrale Komponente des Clients. Sie besteht nur aus der MobileFileManager Klasse. Diese Klasse nutzt die IMobileCommunicationServices Schnittstelle um Dateien, die bearbeitet wurden, an den Server zu senden oder Dateien, die angefordert wurden, zu empfangen. Außerdem ist diese Klasse dafür zuständig sämtliche Dateien zu löschen falls die connectToServer() oder sendFileToServer(Zeichenkette command, Datei file) Methode des IMobileCommunicationServices Interface false zurückgibt. Diese Klasse ist ebenfalls für den Dateizugriff zuständig. Das bedeutet, dass sie ebenfalls die Dateien über das IMobileCryptographic Interface ver- und entschlüsselt.
Die Komponente bietet die CRUD Dateizugriffe an. Außerdem kann sie die Verbindung zum Server auf und abbauen. Sie stellt das IMobileFileServices Interface zur Verfügung, das z.B. von einer GUI genutzt werden kann. (A1.5 und A1.11)

4.2.4 Ablauf des Programms

Erstellung und Zusammenbau der Komponenten
Der Zusammenbau der Komponenten erfolgt über Dependecy Injection.
Beim Server werden die Komponenten, die IExternalServices, IFileServices und ICryptographicServices implementieren, mit den jeweils dafür nötigen Klassen erstellt und dann an den Konstruktor von der Communicationkomponente übergeben.
Bei der Clientanwendung muss man zuerst die MobileExternalServicesKomponente erstellen. Diese wird dann an den Konstruktor von der MobileCommunicationKomponente übergeben damit diese erstellt werden kann.
Als nächstes muss man die MobileCryptographicKomponente erstellen.
Zuletzt übergibt man dann die MobileCommunication und die MobileCryptographicKomponente an den Konstruktor der MobileFileManagerKomponente.

Überprüfung der Aktualität der Daten
Dateien, die zum Server geschickt werden, müssen überprüft werden, ob diese aktueller sind als die bereits auf den Fileserver vorhandenen Dateien. Sollten die Dateien des Fileservers aktueller sein, muss der Benutzer ausdrücklich darauf hingewiesen werden, dass er evtl. eine veraltete Version der Daten bearbeitet hat und diese nun auf den Fileserver überschreiben möchte. Die Aktualität der Daten kann man über den zuletzt bearbeitet Zeitstempel feststellen.

Speichern von bearbeiteten Dateien
Da in diesem Konzept externe Tools zum Öffnen von Dateien verwendet werden, muss separat festgestellt werden wann diese Tools beendet werden. Dies lässt sich über Prozesshandler feststellen. Sobald der, für das Bearbeiten der Datei zuständige, Prozess, beendet ist, bedeutet das, dass der Benutzer mit der Bearbeitung der Datei fertig ist und es kann nun die Datei gespeichert und auf den Server synchronisiert werden. Alternativ kann man den Benutzer darauf hinweisen, dass die Dateien im externen Tool gespeichert werden müssen, bevor er diese mit der App speichern kann. Siehe hierzu Abbildung 11.

Ablauf des Schlüsselaustauschs

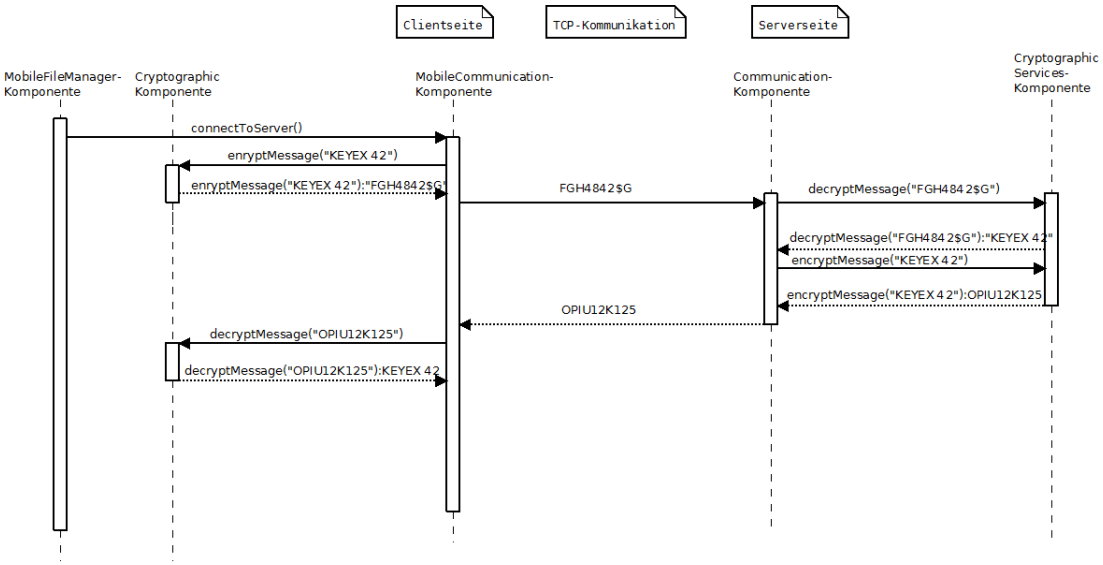

Abbildung 6: Ablauf des Schlüsselaustauschs

Der Benutzer startet die App. Dadurch wird bei der MobileFileManagerKomponente des Clients die Methode connectToServer() aufgerufen. Diese wird an die MobileCommunicationKomponente weitergeleitet. Der Client erzeugt nun eine große Randomzahl und sendet den KEYEX Befehl mit der GeräteID und der Randomzahl an den Server. Dieser Befehl inkl. der Zahl ist mit dem Public-Key des Servers verschlüsselt. Die Zahl ist außerdem noch mit einem Hash aus dem Benutzerpasswort verschlüsselt. Der erste Befehl, den der Server nach einem TCP Verbindungsaufbau erwartet, ist der KEYEX Befehl. Sobald er diese Nachricht erhält, entschlüsselt er die Nachricht und antwortet mit einem KEYEX GeräteID und der Zahl als Bestätigung. Die Antwort ist mit dem Hash des Benutzerpassworts verschlüsselt. (Siehe 4.1.1 Schlüsselaustausch)
Nun kann der Client den Verbindungsaufbau initialisieren.
Die folgenden Ablaufdiagramme zum Verbindungsaufbau führen das Ablaufdiagramm des Schlüsselaustauschs(Abbildung 6) fort. Die connectToServer Methode erstreckt sich hier über mehrere Diagramme. Aufgrund des besseren Verständnisses ist die Kommunikation zwischen Client und Server unverschlüsselt dargestellt. Jeder sendCommand Aufruf innerhalb der connectToServer Methode würde eine verschlüsselte Nachricht versenden, die mit dem ausgehandelten Schlüssel verschlüsselt wurde.

Ablauf des Verbindungsaufbaus

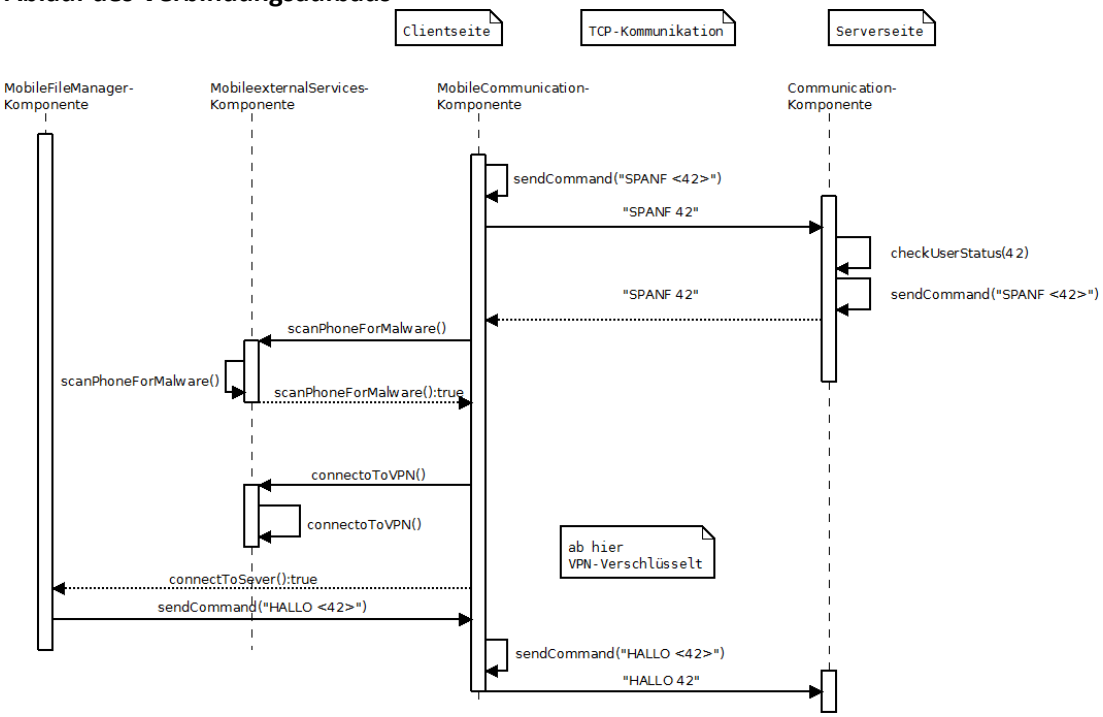

Abbildung 7:Ablauf des erfolgreichen Verbindungsaufbaus

Nach dem Schlüsselaustausch läuft das Programm weiter und die MobileCommunication-Komponente ruft nun die sendCommand Methode mit dem Parameter: „SPANF <GeräteID>“ auf.
Diese Methode schickt dem Server den Befehl SPANF <GeräteID>. Der Server antwortet entweder mit SPANF <GeräteID> oder SPAUF <GeräteID>. Der Server benutzt die checkUserStatus Methode um festzustellen ob der Client gesperrt ist. Wenn der Client nicht gesperrt ist, sendet der Server mit der sendCommand Methode eine SPANF.
Die Verbindung ist hierbei eine synchrone Kommunikation. Nachdem der Client etwas gesendet hat, geht er in den Empfangen Modus und wartet auf eine Antwort vom Server. Bei einem Verbindungsabbruch muss der Client selbständig versuchen die Verbindung wieder aufzubauen.
Nach dem der Client die SPANF des Servers empfangen hat, stößt die MobileCommunicationKomponente die scanPhoneForMalware Methode der MobileExternalServicesKomponente an. Diese scannt dann das Smartphone nach Malware. Wenn keine Malware gefunden wurde, gibt diese Methode true zurück und die MobileCommunicationKomponente ruft über die MobileExternalServicesKomponente connectToVPN auf. Hierdurch wird die VPN Verbindung aufgebaut. Danach gibt dann die connectToServer Methode der MobileCommunicationKomponente true zurück. Als letztes sendet der Client nun ein HALLO <GeräteID> über VPN, damit der Server weiß, dass der Client nun bereit zum Datenaustausch ist.

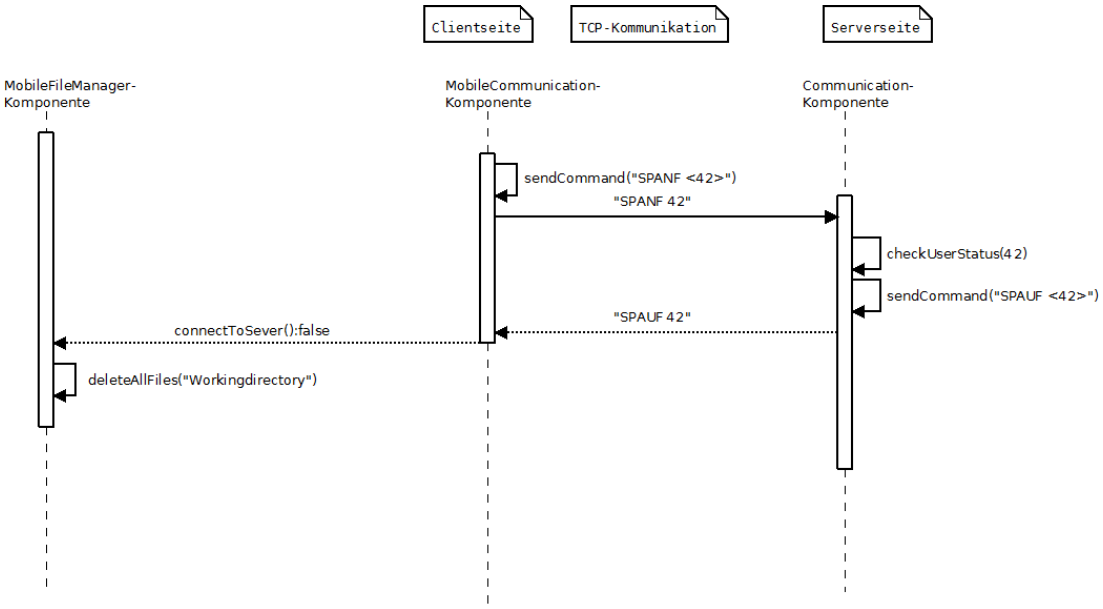

Abbildung 8: Ablauf des Verbindungsaufbaus bei gesperrtem Gerät

In Abbildung 8 ist zu sehen was bei einem gesperrten Client geschieht. Der Ablauf ist bis zur Antwort des Servers identisch zum positiv Fall. Wenn das Gerät gesperrt ist, antwortet der Server mit SPAUF. Sobald der Client eine SPAUF mit seiner ID empfängt, gibt die connectToServer Methode false zurück.
Dadurch weiß die MobileFileManagerKomponente, dass sie nun alle unternehmensrelevanten Daten löschen muss. Gelöscht werden alle Daten, die im Arbeitsverzeichnis der Anwendung liegen. Nachdem die Daten gelöscht wurden, beendet sich die App.

Der letzte mögliche Fall ist, dass das Smartphone beim Scannen Malware findet. Siehe Abbildung 9.

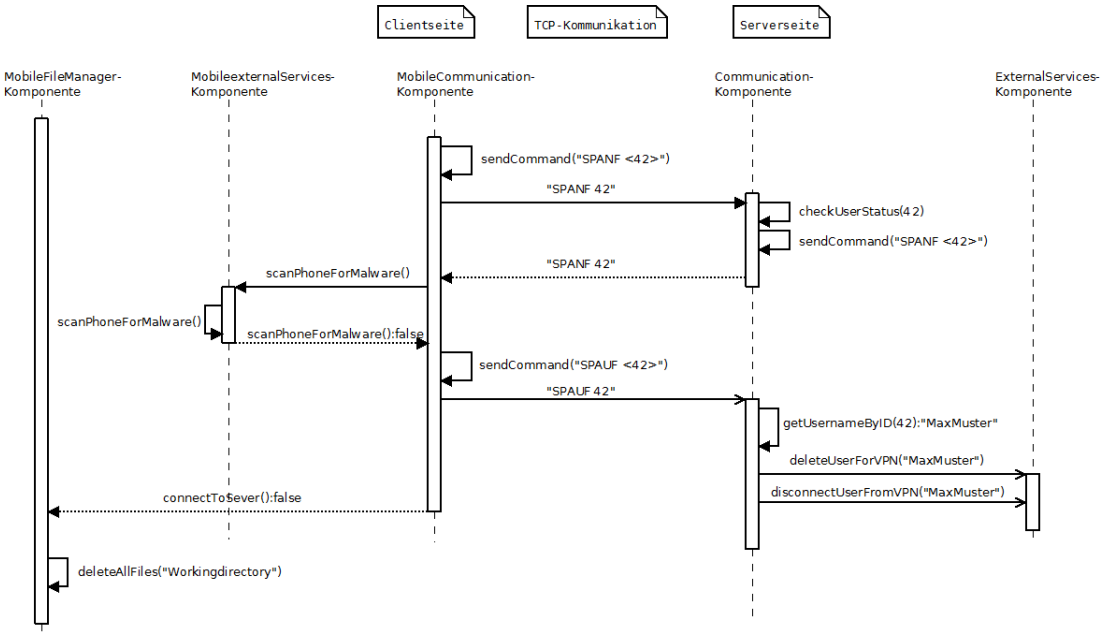

Abbildung 9: Ablauf des Verbindungsaufbaus bei Malwarefund

Am Anfang ist der Ablauf identisch mit den positiv Fall. Wenn jedoch beim Scannen des Smartphones Malware gefunden wird, gibt die scanPhoneForMalware Methode false zurück. Dadurch gibt die connectToServer Methode ebenfalls false zurück. Der Client löscht alle Unternehmensdaten wie im vorherigen Fall. Außerdem sendet die MobileCommunicationKomponente eine SPAUF <GeräteID> Sperraufforderung an den Server. Die CommunicationKomponente des Servers sucht dann mit der getUsernameByID Methode den VPN Zugangsbenutzernamen raus und lässt diesen über die ExternalServicesKomponente löschen. Im Anschluss lässt die CommunicationKomponente ebenfalls die VPN Verbindung über die ExternalServicesKomponente beenden.
Diese Abläufe sind die Umsetzung des Anwendungsfalls „AF1: Benutzer verbindet sich über VPN zum Server“.

Ablauf des Betriebs

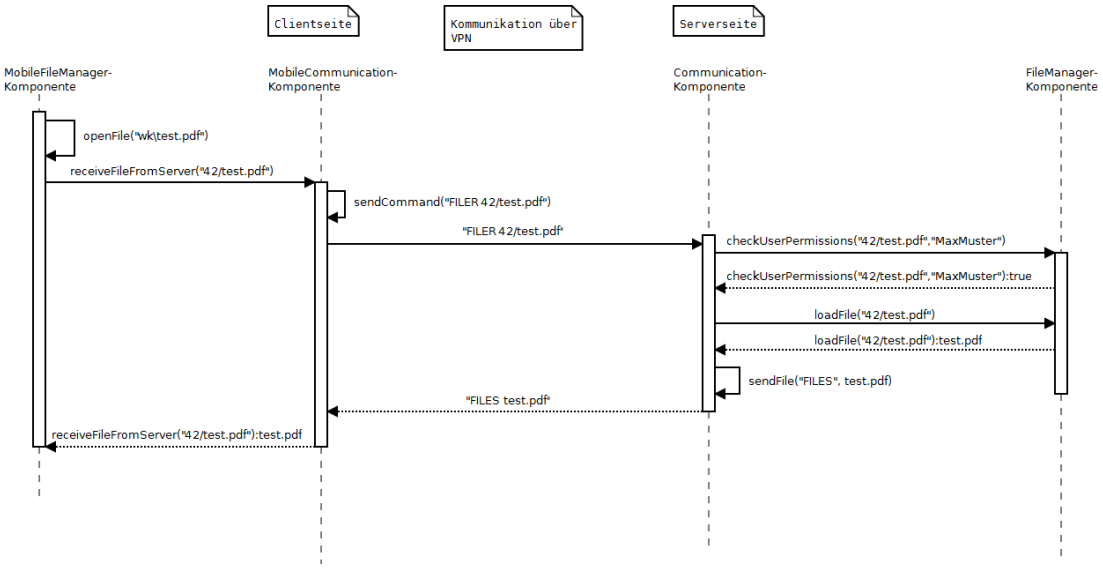

Abbildung 10: Ablauf des Betriebs - Anfordern von Daten vom Server

Wenn der Benutzer eine Datei bearbeiten möchte, versucht er diese zu öffnen. Beim Öffnen prüft die Anwendung, ob die Datei bereits auf dem Smartphone ist oder sie noch auf dem Server liegt, bzw. ob eine neuere Version der Datei auf den Server ist. Sollte die Datei nicht auf dem Smartphone sein, fordert der Client diese beim Server an. Siehe Abbildung 10.
Auf der Serverseite werden beim Anfordern der Daten zuerst die Berechtigungen geprüft. Wenn die nötigen Berechtigungen vorhanden sind, lädt die FileManagerKomponente die Datei über das in der Komponente festgelegte Protokoll und gibt diese an die CommunicationKomponente zurück. Diese sendet die Datei über VPN zurück an den Client.
Bei nicht ausreichenden Berechtigungen oder anderen Fehlern wird eine ERROR <Nachricht> Nachricht an den Client gesendet. Sollte die Datei auf dem Smartphone vorhanden sein, so wird diese durch die MobileCryptographicKomponente entschlüsselt, temporär in entschlüsselter Form im Arbeitsverzeichnis der Anwendung gespeichert und geöffnet.
Dieser Ablauf ist die Umsetzung des Anwendungsfalls „AF2: Benutzer fordert eine Datei über die Anwendung an“.

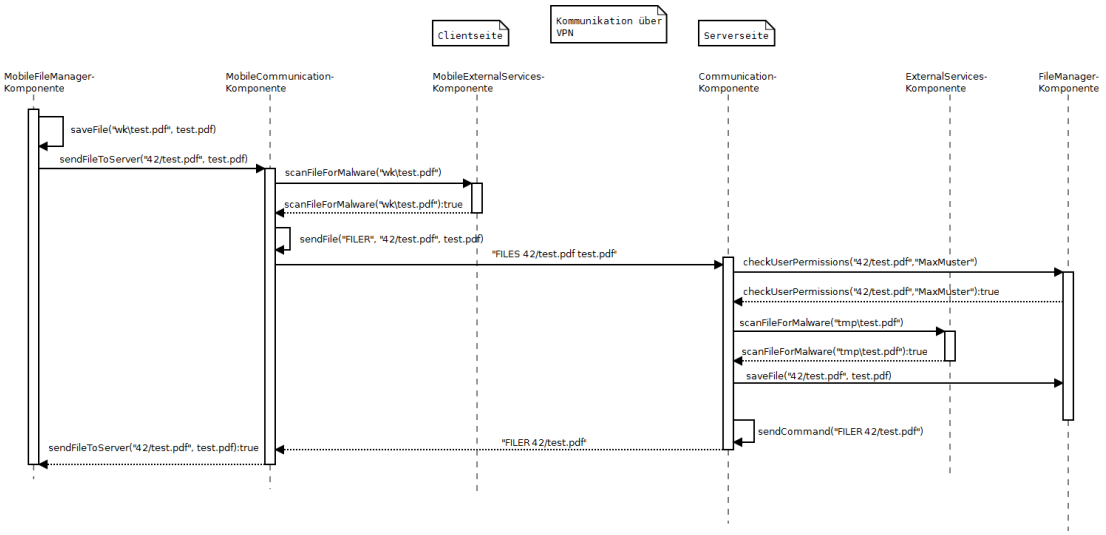

Abbildung 11: Ablauf des Betriebs – Speichern einer Datei

In Abbildung 11 ist der Vorgang des Speicherns einer Datei nach der Bearbeitung dargestellt. Hierbei wird die Datei auch an den Server gesendet, damit diese auch dort aktuell ist. Nachdem der Benutzer die Datei gespeichert hat, wird bei der MobileCommunicationKomponente die sendFileToServer Methode aufgerufen. Als nächstes wird die Datei vom Smartphone auf Malware gescannt. Wenn der Rückgabewert true ist, wurde keine Malware gefunden und die Datei wird an den Server mit der sendFile Methode gesendet. Beim Server werden als erstes die Berechtigungen überprüft. Wenn die nötigen Berechtigungen für den Zugriff vorhanden sind, wird die Datei in ein temporäres Verzeichnis abgelegt und die scanFileForMalware Methode des Servers wird angestoßen. Wenn keine Malware gefunden wurde, wird die Datei auf den Fileserver gespeichert. Im Anschluss wird dann dem Client der Erfolg durch den FILER <Pfad> Befehl mitgeteilt. Bei diesem Ablauf kann es zu 2 Abweichungen kommen. Die erste ist, wenn das Smartphone Malware in der Datei findet. In diesem Fall würde die scanForMalware Methode des IMobileExternalServices false zurückgeben und anstatt die Datei zu senden würde der Client eine SPAUF an den Server senden. Der Ablauf hierbei ist dem von Abbildung 9 sehr ähnlich. Außer dass hier nicht die connectToServer Methode false zurückgibt sondern die sendFileMethode. Danach löscht das Smartphone ebenfalls alle Dateien im Arbeitsverzeichnis.
Die zweite Abweichung wäre hierbei ein serverseitiger Malwarefund. Hier würde die scanFileForMalware Methode der ExternalServicesKomponente false zurückgeben, wodurch die Datei aus dem temporären Verzeichnis gelöscht wird und eine SPAUF an den Client zurück geschickt wird. Der Ablauf hier ist dem der Abbildung 8 sehr ähnlich. Die sendFileToServer Methode gibt hier ebenfalls false zurück und die Dateien im Arbeitsverzeichnis werden gelöscht. Außerdem sperrt der Server den Benutzer für den VPN Zugang und beendet die VPN Sitzung.
Hiermit wird der Anwendungsfall „AF5: Benutzer bearbeitet Datei X und speichert diese“ abgedeckt.

4.2.5 Umsetzung des Clients auf Android

Die Umsetzung des Clients unterscheidet sich von der des Servers. Der Server kann in beliebigen Sprachen ohne Einschränkungen implementiert werden. Der Client hingegen muss in Java implementiert werden, da die Anwendung auf einem Android Smartphone laufen soll.
Wie in 2.2 beschrieben gibt es hierbei einige Besonderheiten. Der Entwurf des Clients ist so konzeptioniert, dass die Androidanwendung nur auf das IAdapterToApplicationcore Interface zugreifen muss um die Funktionen zum Bedienen der App zu gewährleisten.
Der tatsächliche Anwendungskern bleibt von der Androidanwendung abgekapselt.
Zwischen dem tatsächlichen Anwendungskern und der Activity wird eine Adapterklasse implementiert. Diese Adapterklasse bietet die Methoden des IFileServices Interface an sowie eine Methode zum Zusammenbau des Anwendungskerns.
Die MainActivity hat eine entsprechende GUI um alle nötigen Operationen des IFileService Interface zu gewährleisten. Die restlichen Komponenten des Entwurfs von dem Client können davon unabhängig in Java implementiert werden.

5 Implementierung

In diesem Kapitel geht es um die Implementierung eines Prototyps nach dem in 4.2 beschriebenen Entwurfs. Implementiert wird hierbei nur der Kern des Entwurfs, d.h. die Kommunikation zwischen Client und Server, das Verschlüsseln und Entschlüsseln der Dateien auf dem Client und die Verbindung des Servers zum Fileserver. Andere Komponenten, wie der Antimalwarescanner und VPN, werden hierbei gemockt.

5.1 Implementierung des Servers

Die Implementierung des Servers ist in vier Unterkapitel aufgeteilt: Das erste Unterkapitel ist die API Beschreibung der Komponenten des Servers. Das zweite Unterkapitel beschreibt den einzelnen Komponenten und wie bestimmte Teile des Entwurfs hierbei umgesetzt wurden. Das dritte Unterkapitel handelt von der Kommunikation und der Communication-Komponente. Außerdem wird dort gezeigt wie diese implementiert wurden. Das vierte Unterkapitel beschreibt das Testen der Komponenten und den Verbund-Test des Servers.

5.1.1 API der Komponenten des Servers

FileManagerKomponente
Die Komponente wird über das IFileServices Interface angesprochen. Das Interface bietet folgende Methoden an:

void createFile(Zeichenkette path, Datei file)
Diese Methode dient zum Anlegen einer Datei file mit dem Pfad path.

Datei loadFile(Zeichenkette path)
Hiermit wird eine Datei mit dem Pfad path geladen und zurückgegeben.
Der Rückgabewert ist die geladene Datei.

void deleteFile(Zeichenkette path)
Diese Methode löscht eine Datei mit dem Pfad path.

void saveFile(Zeichenkette path, Datei file)
Die hier übergebene Datei file wird mit dem Pfad path gespeichert. Vorhandene Dateien werden überschrieben.
Äquivalent dazu benutzt der IFileProtocolAdapter das selbe Interface, wobei die Methoden der IFileServices an den IFileProtocolAdapter delegiert werden.

Zeichenkette getFileList(Zeichenkette path)
Fordert eine Verzeichnisliste beim Fileserver an. Die einzelnen Daten einer Datei sind mit "|" getrennt. Die Eigenschaften der kompletten Dateien sind mit ";" getrennt. Zum Beispiel würde eine Dateiliste mit 2 Dateien folgendermaßen aussehen:
„name:Datei1|lastModified:13.12.2012|path:VPNUserDirs\42\Datei1.pdf;
name:Datei2|lastModified:01.01.2012|path:VPNUserDirs\42\MyFiles\Datei2.pdf"

boolean checkUserPermissions(Zeichenkette path, Zeichenkette username, PermissionEnum permission)
Diese Methode überprüft die Zugriffsberechtigung permission(Read, Write, List, Delete) des Benutzers username auf den Pfad path.

void closeConnection() throws TechnicalException;
Beendet die Verbindung zum FileServer.

void clearTemp(String path);
Löscht die temporäre Datei mit dem Pfad path im tmp Ordner.

ExternalServicesKomponente
Die ExternalServicesKomponente hat somit ein Komponenteninterface IExternalServices und 2 Klasseninterfaces IAntiMalwareAdapter und IVPNAdapter:

IExternalServices

boolean scanFileForMalware(Zeichenkette path)
Diese Methode wird an die Implementierung des IAntiMalwareAdapters weitergeleitet.

void deleteUserForVPN(Zeichenkette username)
Diese Methode wird an die Implementierung des IVPNAdapters weitergeleitet.

void disconnectVPNConnectionToUser(Zeichenkette username)
Diese Methode wird an die Implementierung des IVPNAdapters weitergeleitet.

IAntiMalwareAdapter

boolean scanFileForMalware(Zeichenkette path)
Diese Methode ruft den Antimalware Dienst auf und übergibt ihm den Pfad path zum scannen. Nachdem das Programm fertig ist, muss das Ergebnis interpretiert und auf true oder false abgebildet werden. Der Rückgabewert ist das interpretierte Ergebnis, an diesem Ergebnis kann man ablesen, ob die Datei Malware enthält oder nicht.

IVPNAdapter

void deleteUserForVPN(Zeichenkette username)
Diese Methode löscht den VPN-Zugang für den Benutzer mit dem Benutzernamen username. Hier wird der externe VPN Dienst aufgerufen und der Zugang für den Benutzer gesperrt.

void disconnectVPNConnectionToUser(Zeichenkette username)
Diese Methode beendet die VPN Verbindung zum Benutzer, mit dem Benutzernamen username. Hier wird der externe VPN Dienst aufgerufen, um den Benutzer vom VPN Netzwerk zu trennen.

CryptographicKomponente
Die CryptographicKomponente bietet, bei dem Server, lediglich Methoden zum Ver- und Entschlüsseln von Nachrichten an.

ICryptographicServices

byte[] encryptMessage(Zeichenkette message, Zeichenkette key)
Verschlüsselt die Nachricht message mit dem asymmetrischen Algorithmus und dem ausgehandelten Schlüssel. Rückgabewert ist die verschlüsselte Nachricht in Byte Format.

Zeichenkette decryptMessage(byte[] message, Zeichenkette key)
Entschlüsselt die Nachricht message mit dem asymmetrischen Algorithmus und dem ausgehandelten Schlüssel. Rückgabewert ist die entschlüsselte Nachricht.

Zeichenkette decryptMessageWithPrivateKey(byte[] message)
Entschlüsselt die Nachricht message mit dem Private-Key des Servers.
Rückgabewert ist die entschlüsselte Nachricht.

byte[] decryptKeyWithPrivateKey(byte[] encKey, byte[] key)
Entschlüsselt den, mit den PublicKey verschlüsselten, Sessionkey und entschlüsselt dann den Sessionkey nochmal mit dem Hashwert des Userpassworts+Datum.

byte[] encryptSessionKey(byte[] sessionKey, byte[] key)
Verschlüsselt den Sessionkey mit dem Hashwert des Userpassworts und Datums.

byte[] encryptMessageWithSessionKey(Zeichenkette message, byte[] key)
Verschlüsselt die Nachricht message mit dem Hashwert des Userpassworts+Datum.

CommunicationKomponente
Die CommunicationKomponente ist die zentrale Komponente des Servers. Sie implementiert das ICommunicationServices Interface an.

ICommunicationServices

void startService()
Dies ist die Einstiegsmethode des Servers. Hier müssen die ServerSockets gestartet werden und die Verbindung zum Fileserver muss initialisiert werden.

void stopService()
Beendet den Server ordnungsgemäß. Hierbei muss beachtet werden, dass alle Änderungen, wie z.B. neu gesperrte GeräteIDs, persistiert werden müssen.

IConnectionManager

void sendCommand(Zeichenkette command)
Sendet den Befehl command an den Server.

void sendFile(Zeichenkette command, Datei file)
Sendet die Datei file mit dem Befehl command an den Server.

void sendString(Zeichenkette message)
Sendet die Zeichenkette message an den Client. Diese kann beliebige länge haben.

void sendData(byte[] data)
Sendet die Daten data an den Client. Es wird keine String Konvertierung vorgenommen, sodass die Daten erhalten bleiben.

void sendFileList(Zeichenkette command, Zeichenkette fileList)
Sendet die Dateiliste fileList mit dem Befehl command als Zeichenkette an den Client.

byte[] receiveCommandFromClient()
Methode zum Empfangen eines Commands des Clients.

byte[] receiveKeyFromClient()
Empfängt, den für die Sperranfragen nötigen, geheimen Schlüssel vom Client.

byte[] receiveDeviceIDFromClient)
Empfängt die GeräteID vom Client

byte[] receiveDataFromClient()
Empfängt einen beliebig langen Datenstrom vom Client.

byte[] receivePathFromClient()
Empfängt eine Pfadangabe vom Client.

String receiveFileFromClient(Zeichenkette tmpPath)
Empfängt eine Datei vom Client und legt diese im tmpPath Pfad ab.

void closeConnection();
Beendet die Verbindung zum Client.

5.1.2 Implementierung der Komponenten

Die Komponenten wurden, wie in 4.2.2 entworfen, in Java umgesetzt. Komponenten, die Daten aus einer Konfigurationsdatei brauchen, bekommen zusätzlich einen ConfigFileReader im Konstruktor übergeben. Somit kann jede Klasse die nötigen Daten, für sich selbst, aus der Konfigurationsdatei auslesen.
Sämtliche Exceptions, die nichts mit der Fachlichkeit zu tun haben, werden gecatcht und als TechnicalException gekapselt und dann weitergeleitet. Wenn eine Exception bei der Ver- oder Entschlüsselung auftritt, wird diese in eine CryptographicException gekapselt und weitergeleitet.

FileManagerKomponente

Die FileManagerKomponente ist in 2 Klassen unterteilt: den FileManager und den FileProtocolAdapter.
Die FileManageklasse dient als Fassade, die Aufrufe an den FileProtocolAdapter delegiert. Außerdem liest der FileManager, die für den FileProtocolAdapter nötigen, Daten aus der Configdatei aus und übergibt diese an den FileProtocolAdapter.
Der FileProtocolAdapter kommuniziert durch den Apache Commons Net FTPClient mit dem FTP Server.
Folgend nun der Konstruktor des FileProtocolAdapters in JavaCode:

```
FileProtocolAdapter(String host, int port, String username, String password) throws
TechnicalException {
        this.ftpClient = new FTPClient();
        try {
               ftpClient.connect(InetAddress.getByName(host), port);
               ftpClient.login(username, password);
               ftpClient.enterLocalPassiveMode();
        } catch (IOException e) {
               throw new TechnicalException("FTP Host kann nicht gefunden werden.
               Fehler: " + e.getMessage());
        }
    }
```

Hierbei ist es wichtig den FTPClient vor dem Senden und Empfangen auf Binary File Type zu stellen, da man sonst die Datei nicht ohne weiteres mit FileStreams bearbeiten kann. Nach dem Login, kann man, mit dem FTPClient, nun Dateien vom Server anfordern oder diese dort hochladen. Bei dem FTP Server, der hier zum Testen verwendet wurde, musste man außerdem beachten, dass Pfade mit “/“ und nicht mit “\“ angegeben werden.

ExternalServicesKomponente

Die ExternalServicesKomponente ist, bei der Prototypimplementierung, komplett gemockt. Sowohl VPN als auch der AntiMalwareAdapter haben keine Funktionalität.

CryptographicKomponente

Die CryptographicKomponente wurde wie im Entwurf angegeben implementiert. Als Verschlüsselungsalgorithmen wurden für die symmetrische Verschlüsselung AES, mit SUN als Provider, und als asymmetrische Verschlüsselung wurde RSA, mit SUN als Provider, gewählt. Die CryptographicKomponente besteht aus 2 Klassen. Die Erste ist der RSAKeygenerator. Hierbei wird, durch ein einfaches Script, ein Public- und Private Key mit der Länge 2048bit erzeugt. Die zweite Klasse ist die CryptographicServices. Hier werden die Nachrichten ver- und entschlüsselt. Bei der Implementierung wurden hierfür private Methoden erstellt, auf die die Interface Methoden zurückgeführt wurden.
Folgend nun zwei Beispiele von einer symmetrischen und einer asymmetrischen Entschlüsselung, angefangen bei der symmetrischen Entschlüsselung einer Nachricht:

```
public byte[]decryptMessage (String message, byte[] key) throws CryptographicExcepti-
on {
                try {
                    SecretKey secret = new SecretKeySpec(key, "AES");
                    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    byte[] ivBytes = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                       14, 15};
                    IvParameterSpec iv = new IvParameterSpec(ivBytes);
                    cipher.init(Cipher.DECRYPT_MODE, secret, iv);
                    return cipher.doFinal(message.getBytes());
                } catch (NoSuchAlgorithmException e) {
                    throw new CryptographicException("Verschlüsselungsalgorithmus
                       nicht gefunden. Fehler: " + e.getMessage());
```

In diesem Codebeispiel wurden weitere Catchblöcke rausgelassen. Die Verschlüsselung der Daten erfolgt hierbei, indem man `cipher.init(Cipher.DECRYPT_MODE, secret, iv);` gegen `cipher.init(Cipher.ENCRYPT_MODE, secret, iv);` austauscht. Der InitialVector wurde hier fest auf das Array ivBytes gesetzt. Dieses Codebeispiel dient zum Verständnis. Bei der Implementierung werden Daten wie der Algorithmus, die ivBytes, etc. aus der Config Datei ausgelesen.

Nun ein Beispiel für die asymmetrische Entschlüsselung einer Nachricht.

```
private byte[] decryptDataWithPrivateKey(byte[] data) throws CryptographicException {
        try {
            ////Key einlesen
            DataInputStream fileIn = new DataInputStream(new FileInputStream(keyPath
               + "private.key"));
            int prvKeyLen = fileIn.readInt();
            byte[] rawKey = new byte[prvKeyLen];
            fileIn.read(rawKey);
            ////PrivateKey aus eingelesenen Daten erzeugen
            EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(rawKey);
            PrivateKey privateKey =  KeyFactory.getInstance("RSA")
               .generatePrivate(privateKeySpec);
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            return cipher.doFinal(data);
        } catch (NoSuchAlgorithmException e) {
```

Diese private Methode wird für die Interface Methoden decryptKeyWithPrivateKey und decryptMessageWithPrivateKey verwendet.
Es wird hier zuerst der Private-Key aus der Datei, die auf dem Server liegt, ausgelesen. Danach wird aus den gelesenen Bytes der Private-Key erstellt. Da der Server keine Verschlüsselung mit dem Public-Key vornimmt, gibt es keine asymmetrischen Verschlüsselungsmethoden bei der Implementierung des Servers.

5.1.3 Implementierung der Kommunikation und des CommunicationManagers

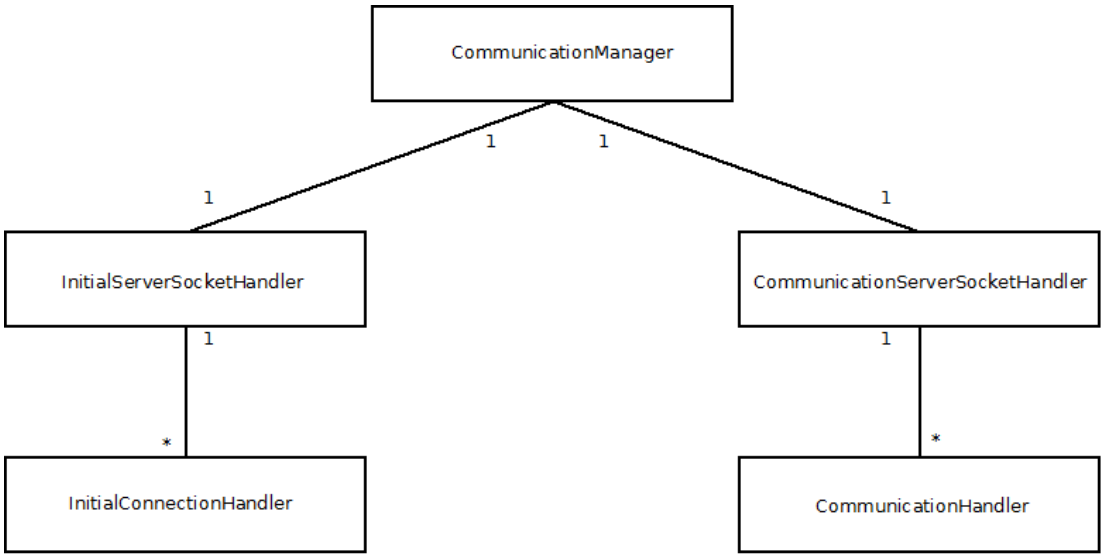

Abbildung 12: Threadmodell des CommunicationManagers

Der Server hat in dieser Arbeit 2 unterschiedliche Ports, über die er unterschiedliche Anfragen beantwortet: Die Initialkommunikation, mit Sperranfragen und Sperraufforderungen, und die darauf folgende Kommunikation wie z.B. den Dateiaustausch.
Für die Implementierung bedeutet das, dass hierfür 2 ServerSockets nötig sind. Da die ServerSockets jederzeit(auch parallel) Anfragen beantworten sollen, werden die Sockets jeweils als ein Thread, im CommunicationManager, gestartet. Die ServerSockets sind jeweils in einem ServerSocketHandler gekapselt. Diese Handler haben Start- und Stopmethoden und erstellen bei einer eingehenden Verbindung einen weiteren Handlerthread, der die Kommunikation zum Client abhandelt. Danach geht der ServerSocketHandler sofort wieder in die Acceptmethode, um schnellstmöglich wieder für Anfragen zur Verfügung zu stehen.
Den Handlern werden die nötigen Referenzen der Komponenten zum Abarbeiten der Anfragen über den Konstruktor übergeben.

Im Folgenden nun als Beispiel die run() Methode des InitialServerSocketHandlers, der für die Initialkommunikation zuständig ist:

```
@Override
    public void run() {
        try {
            handlerActive = true;
            while (handlerActive) {
                      InitialConnectionHandler initialConnectionHandler =
                      new InitialConnectionHandler(
                      externalServices, cryptographicServices,
                      serverSocket.accept(),
                      userMap, userPasswords);
                      Thread initialConnectionThread = new Thread(
                      initialConnectionHandler);
                      initialConnectionThread.start();
                      handlerList.add(initialConnectionHandler);
                      cleanList();
            }
        } catch (SocketException se) {…}
```

Bei einer eingehenden Verbindungsanfrage an den Server wird ein Handlerthread erstellt, der die Anfragen des Clients nach dem Protokoll in 4.2.1 beantwortet. Hierfür erstellt sich der Handler einen ConnectionManager, um die direkte Kommunikation über TCP abzukapseln. So, wie bei den ServerSockets, gibt es hier ebenfalls 2 unterschiedliche Handler: Der InitialServerSocketHandler erstellt einen InitialConnectionHandler und der CommunicationServerSocketHandler erstellt einen CommunicationHandler(siehe Abbildung 12).
Der InitialConnectionHandler führt als erstes einen Schlüsselaustausch durch und empfängt dann die Sperranfrage/Sperraufforderung des Clients.
Hier als Code:

```
@Override
    public void run() {
        try {
               handlerActive = true;
               ConnectionManager connectionManager = new ConnectionManager(socket);
               byte[] encMessage = connectionManager.receiveDataFromClient();
               String message = cryptographicSer-
               vices.decryptMessageWithPrivateKey(encMessage);
               String[] messageAry = message.split(" ");
               if (messageAry[0].equals("KEYEX")) {
                      String deviceID = messageAry[1];
                      byte[] userPassword = getKeyForUser(deviceID);
               ////Den mit dem Publickey verschluesselten geheimen schluessel ent-
               schluesseln
                      byte[] encKey = connectionManager.receiveKeyFromClient();
                      byte[] decKey = cryptographicSer-
                      vices.decryptKeyWithPrivateKey(encKey, userPassword);
               ////Den geheimen Shcluessel mit dem Userpasswort hash entschluesseln
                                    connectionMana-
                      ger.sendData(cryptographicServices.encryptMessage("KEYEX " +
                      deviceID, userPassword));
                                    connectionMana-
                      ger.sendData(cryptographicServices.encryptSessionKey(decKey,
                      userPassword));

               byte[] encSperrAnfrage = connectionManager.receiveDataFromClient();
```

```
            String anfrage = cryptographicServices.decryptMessage(encSperrAnfrage,
            decKey);
            if (anfrage.equals("SPANF")) {
                   deviceID = cryptographicSer-
                   vices.decryptMessage(connectionManager.
                   receiveDeviceIDFromClient(), decKey);
                   if (checkUserStatus(deviceID)) {
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey("SPANF", decKey));
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey(deviceID, decKey));
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey("CLOSE", decKey));
                          connectionManager.closeConnection();
                   } else {
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey("SPAUF", decKey));
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey(deviceID, decKey));
                                                connectionMana-
                          ger.sendData(cryptographicServices
                          .encryptMessageWithSessionKey("CLOSE", decKey));
                          connectionManager.closeConnection();
                }
            } else if (anfrage.equals("SPAUF")) {
                   externalServices.deleteUserFromVPN(userMap.get(deviceID));
                                       connectionMana-
                   ger.sendData(cryptographicServices
                   .encryptMessageWithSessionKey("SPAUF", decKey));
                                       connectionMana-
                   ger.sendData(cryptographicServices
                   .encryptMessageWithSessionKey(deviceID, decKey));
                                       connectionMana-
                   ger.sendData(cryptographicServices
                   .encryptMessageWithSessionKey("CLOSE", decKey));
                   connectionManager.closeConnection();
             } else if (anfrage.equals("CLOSE")) {
                                 connectionMana-
                   ger.sendData(cryptographicServices.encryptMessageWithSessionKey
                   ("CLOSE", decKey));
                   connectionManager.closeConnection();
             }
         }
      } catch (TechnicalException e){…}
```

Wie man hier im Code sehen kann, ist der Initialhandlerthread ein Thread der nur einmal durchläuft und danach beendet wird. Er tauscht den Schlüssel mit dem Client aus, empfängt die Sperranfrage oder Sperraufforderung, beantwortet diese und beendet sich dann. Der CommunicationHandler ist sehr ähnlich aufgebaut, nur dass dieser eine While-Schleife besitzt, die immer wieder Anfragen vom Client beantworten kann, bis dieser den CLOSE Befehl sendet.

Außerdem ist der Port des CommunicationServerSocketHandlers nur im internen Netz verfügbar, d.h. die Kommunikation muss nicht mehr verschlüsselt werden, da ab hier alles über VPN versendet wird. Es folgen nun einige Codeausschnitte der wichtigsten Funktionen des, in 4.2.1 beschriebenen, Protokolls.

Abarbeitung einer Sperraufforderung:

```
if (command.equals("SPAUF")) {
        deviceID = new String(connectionManager.receiveDeviceIDFromClient());
        if (userMap.containsKey(deviceID)) {
            deleteUser(deviceID);
            connectionManager.sendCommand("SPAUF");
            connectionManager.sendString(deviceID);
            connectionManager.sendCommand("CLOSE");
            handlerActive = false;
            connectionManager.closeConnection();
        } else {
            connectionManager.sendCommand("ERROR");
            connectionManager.sendCommand("NO ID");
            connectionManager.sendCommand("CLOSE");
            handlerActive = false;
            connectionManager.closeConnection();
        }
```

Wie man hier sehen kann wird bei einer Sperraufforderung überprüft, ob die GeräteID vorhanden ist. Danach wird der Benutzer gelöscht. Zuletzt wird auch die Verbindung beendet und der Handler wird auf „inaktiv“ gesetzt, damit er vom ServerSocketHandler aus der Liste der noch aktiven Threads entfernt werden kann.

Abarbeitung einer HALLO Message:

```
if (command.equals("HALLO")) {
        deviceID = new String(connectionManager.receiveDeviceIDFromClient());
        if (userMap.containsKey(deviceID)) {
            connectionManager.sendCommand("HALLO");
            connectionManager.sendString(deviceID);
        } else {
            connectionManager.sendCommand("ERROR");
            connectionManager.sendString("NO ID");
            connectionManager.sendCommand("CLOSE");
            handlerActive = false;
            connectionManager.closeConnection();
        }
```

Die „Hallo“ Message dient hierbei lediglich zur Identifizierung des Clients durch die GeräteID. Außerdem wird überprüft, ob die GeräteID Zugang zum Server hat.

FILER Message:

```
if (command.equals("FILER")) {
        String path = new String(connectionManager.receivePathFromClient());
        if (fileServices.checkUserPermission(userMap.get(deviceID), path, Permissio-
        nEnum.READ)) {
             File requestedFile = fileServices.loadFile(path);
             connectionManager.sendFile("FILES", requestedFile);
             fileServices.clearTemp(path);
         } else {
             connectionManager.sendCommand("ERROR");
             connectionManager.sendString("No Permission");
         }
```

Hierbei wird als erstes geprüft, ob der User Zugriff auf die Datei hat. Als nächstes wird dann, über die FileServices Schnittstelle, die Datei angefordert und an den Client gesendet.

Als letzter wichtiger Codeausschnitt ist hier nun der FILES Befehl:

```
if (command.equals("FILES")) {
        String path = new String(connectionManager.receivePathFromClient());
        String tmpPath = connectionManager.receiveFileFromClient(path);
        boolean checkPermission = fileSer-
        vices.checkUserPermission(userMap.get(deviceID), path, PermissionEnum.WRITE);
        boolean checkForMalware = externalServices.scanFileForMalware(tmpPath);
        if (!checkForMalware && checkPermission) {
             fileServices.saveFile(path, new File(tmpPath));
             connectionManager.sendCommand("FILER");
             connectionManager.sendString(path);
         } else if (checkForMalware) {
             connectionManager.sendCommand("SPAUF");
             connectionManager.sendString(deviceID);
             deleteUser(deviceID);
             connectionManager.closeConnection();
         } else {
             connectionManager.sendCommand("ERROR");
             connectionManager.sendString("No Permission");
         }
```

Hierbei werden ebenfalls die Zugriffsberechtigungen geprüft, bevor die Datei an den Fileserver gesendet wird. Außerdem wird die Datei auf Malware untersucht.

Die FILEL und FILED Befehle des Protokolls wurden, ähnlich wie die oben genannten Befehle, dem Entwurf entsprechend implementiert.

5.1.4 Tests

Die Servertests sind in 2 Testarten unterteilt: Komponententests und Verbund-Tests.
Bei den Komponententests wurde nur die CryptographicKomponente getestet, da die anderen Komponenten entweder nur Aufrufe delegieren oder aus Sockets auslesen und die Informationen weitergeben. Außerdem wird im Verbund-Test sowohl die Kommunikation als auch das Zusammenspiel der Komponenten getestet.
Im Verbund-Test wird der komplette Anwendungskern des Servers zusammengebaut, um das Kommunikationsprotokoll zu testen. Hierfür wurde, für unabhängiges Testen vom Client, eine MockClient Klasse implementiert. Diese besitzt nur die Verschlüsselungs- und Kommunikationsmethoden des Clients. Dadurch ist keine Logik im MockClient, sondern man kann lediglich Daten über TCP senden/empfangen und Daten ver-/entschlüsseln. Beim Verbund-Test wird das gesamte Zusammenspiel der Komponenten getestet. Dies schließt auch die FileManagerKomponente mit ein. Für die Tests wurde der FileZilla FTP Server verwendet, den der FileManager für Dateitransfers benutzt.

5.2 Implementierung des Clients

Dieses Kapitel ist in fünf Unterkapitel unterteilt. Das erste Unterkapitel ist die Beschreibung der API der Komponenten des Clients. Das zweite Unterkapitel handelt von der Implementierung der Komponenten und zeigt einige Codebeispiele dafür auf. Das dritte Unterkapitel handelt von der Verwaltung und Verschlüsselung der Dateien auf den Smartphone. Das vierte Unterkapitel handelt von der Schnittstelle zwischen der Android Anwendung und dem Anwendungskern des Clients, bzw. wie die Clientanwendung in die Android Anwendung integriert wird. Im letzten Unterkapitel wird ebenfalls, wie beim Server, das Testen des Clients beschrieben.

5.2.1 API der Komponenten des Clients

MobileExternalServicesKomponente
Die MobileExternalServicesKomponente bietet ähnliche Methoden wie der Server an. Sie ist ebenfalls in 3 Interfaces unterteilt.

IMobileExternalServices

boolean scanFileForMalware(Zeichenkette path)
Diese Methode wird an die Implementierung des IMobileAntiMalwareAdapters delegiert. Sie dient zum Initialisieren eines Scans einer einzelnen Datei mit dem Pfad path. Der Rückgabewert ist das interpretierte Ergebnis des Scans. True für einen Fund und false bei keinem Fund.

boolean scanPhoneForMalware()
Diese Methode wird an die Implementierung des IMobileAntiMalwareAdapters delegiert. Mit dieser Methode wird ein Komplettscan des Smartphones angestoßen. Der Rückgabewert ist das interpretierte Ergebnis des Scans. True für einen Fund und false bei keinem Fund.

void connectToVPN()
Diese Methode wird an die Implementierung des IMobileVPNAdapters delegiert. Hiermit wird eine Verbindung zum VPN aufgebaut.

void disconnectFromVPN()
Diese Methode wird an die Implementierung des IMobileVPNAdapters delegiert. Hiermit wird die VPN Verbindung getrennt.

IMobileAntiMalwareAdapter

boolean scanFileForMalware(Zeichenkette path)
Hier muss der Aufruf des externen Antimalwareprogramms angestoßen werden. Der Aufruf soll eine Datei, mit dem Pfad path, auf Malware untersuchen Der Rückgabewert ist das interpretierte Ergebnis des Scans. True für einen Fund und false bei keinem Fund.

boolean scanPhoneForMalware()
Mit dieser Methode soll ein Komplettscan des Smartphones, durch das externe Antimalwareprogramm, durchgeführt werden. Der Rückgabewert ist das interpretierte Ergebnis des Scans. True für einen Fund und false bei keinem Fund.

IMobileVPNAdapter

void connectToVPN()
Diese Methode benutzt das externe VPN Programm, um eine VPN Verbindung zum VPN Gateway aufzubauen.

void disconnectFromVPN()
Mit dieser Methode soll man die VPN Verbindung wieder trennen können, d.h. hier muss ebenfalls das externe VPN Programm, durch einen Aufruf, dazu gebracht werden, die Verbindung zu trennen.

MobileCommunicationKomponente
Diese Komponente ist zuständig für die Kommunikation vom Client zum Server.

IMobileCommunicationServices

boolean connectToServer()
Diese Methode soll die Verbindung zum Server initialisieren. Hier muss als erstes ein kompletter Scan des Smartphones stattfinden. Wenn dieser negativ ist, wird eine Sperranfrage an den Server geschickt. Erst wenn Diese auch negativ ist, wird, über IMobileExternalServices, die VPN Verbindung aufgebaut und die Methode gibt true zurück. Sollte einer der oben genannten Fälle positiv ausfallen, baut diese Methode keine Verbindung zum VPN auf und gibt false zurück. Diese Methode gibt nur false zurück wenn das Gerät gesperrt ist oder Malware gefunden wird.

void disconnectFromServer()
Mit dieser Methode beendet man die VPN Verbindung zum Server.

boolean sendFileToServer(Zeichenkette path, Datei file)
Diese Methode sendet, mit einem Sendebefehl, die Datei file, die unter dem Pfad path gespeichert werden soll, an den Server. Die Datei wird vorher, über IMobileExternalServices, nach Malware durchsucht. Sollte keine Malware gefunden werden, sendet die Methode die Datei an den Server, wartet auf die Antwort des Servers und gibt true zurück. Nur wenn Malware gefunden wird, gibt diese Methode false zurück. Malware kann einerseits vom Smartphone gefunden werden oder andererseits wenn vom Server kein FILER Kommando, sondern eine SPAUF Sperraufforderung zurückkommt.

Datei receiveFileFromServer(Zeichenkette prefix, Zeichenkette path)
Diese Methode fordert eine Datei, mit dem Pfad path, vom Server an, empfängt diese mit der receiveFromServer Methode des IMobileConnectionManager Interface und konvertiert die empfangen Daten zu der angeforderten Datei. Das Prefix ist für Android nötig. Im Prefix steht z.B. das externe Speicherverzeichnis des Androidsystems.

List<Zeichenkette> getFileListFromServer(Zeichenkette path)
Mit dieser Methode fordert der Client eine Datei und Verzeichnisliste für den Pfad path an. Diese Methode wird an die IMobileConnectionManager Methode sendCommand weitergeleitet. Die einzelnen Daten einer Datei sind mit "|" getrennt. Die Eigenschaften der kompletten Dateien sind mit ";" getrennt. Zum Beispiel würde eine Dateiliste mit 2 Dateien folgendermaßen aussehen:
„name:Datei1|lastModified:13.12.2012|path:VPNUserDirs\42\Datei1.pdf|type:file;
name:Datei2|lastModified:01.01.2012|path:VPNUserDirs\42\MyFiles\Datei2.pdf|type:file"

IMobileConnectionManager

void sendCommand(Zeichenkette command)
Sendet den Befehl command an den Server:

void sendFile(Zeichenkette command, Zeichenkette path, Datei file)
Sendet, mit dem Befehl command, die Datei file, mit dem Speicherpfad path, an den Server.

void sendString(Zeichenkette data)
Sendet den String data an den Server.

void sendData(byte[] data)
Sendet den byte[] data an den Server. Diese Methode ist z.B. für den Schlüsselaustausch nötig, da der Randomwert in Byteform nicht als String verschickt werden kann.

byte[] receiveCommandFromServer()
Empfängt einen Befehl vom Server.

byte[] receiveDeviceIdFromServer()
Empfängt eine GeräteID vom Server.

byte[] receiveDataFromServer()
Empfängt Daten vom Server. Hierbei können die Daten eine beliebige Länge haben.

Zeichenkette receiveFileFromServer(Zeichenkette path)
Empfängt eine Datei vom Server und speichert diese in den Pfad path.

MobileCryptographicKomponente
Die MobileCryptographicKomponente bietet 2 Interfaces an:
Das ICryptographicServicesForFiles Interface bietet Methoden an, die für das Ver- und Entschlüsseln von Dateien zuständig sind.
Das ICryptographicServicesForCommunication Interface bietet ähnliche Methoden an, wie die des Servers.

ICryptographicServicesForFiles

encryptFile(Zeichenkette prefix, Zeichenkette path, byte[] key)
Die Methode encryptFile verschlüsselt eine Datei, mit dem Pfad path, mit dem Schlüssel key.

decryptFile(Zeichenkette prefix, Zeichenkette path, byte[] key)
Die Methode decryptFile entschlüsselt eine Datei, mit dem Pfad path, mit dem Schlüssel key.

ICryptographicServicesForCommunication

byte[] encryptMessage(Zeichenkette Message, byte[] key)
Diese Methode verschlüsselt die Nachricht Message mit dem Schlüssel key.

Zeichenkette decryptMessage(Zeichenkette Message, byte[] key)
Diese Methode entschlüsselt die Nachricht Message mit dem Schlüssel key.

byte[] decryptSessionKey(byte[] encKey,byte[] key)
Entschlüsselt den Sessionkey encKey mit dem Hash des Benutzerpassworts key. Der Rückgabewert ist der entschlüsselte Sessionkey.

byte[] decryptDataWithSessionKey(byte[] data,byte[] key)
Entschlüsselt den Datensatz data mit dem Sessionkey key. Der Rückgabewert ist der entschlüsselte Datensatz.

byte[] encryptSessionKey(byte[] sessionKey, byte[] key)
Verschlüsselt den Sessionkey sessionKey mit dem Hash des Benutzerpassworts key. Der Rückgabewert ist der verschlüsselte Sessionkey.

byte[] encryptMessageWithSessionKey(String message, byte[] key)
Verschlüsselt die Nachricht message mit dem Sessionkey key und gibt diese zurück als byte[].

byte[] encryptMessageWithPublicKey(byte[] message)
Verschlüsselt die Nachricht message mit dem Public-Key des Servers und gibt diese zurück.

byte[] encryptKeyWithPublicKey(byte[] decKey, byte[] userPw)
Verschlüsselt den Sessionkey decKey zuerst mit dem Hash des Benutzerpassworts und dann mit dem Public-Key des Servers.

byte[] getSecureRandom()
Gibt eine sichere Randombytefolge zurück. Diese wird als Sessionkey benutzt.

MobileFileServicesKomponente
Diese Komponente ist die Schnittstelle der Clientanwendung nach Außen.
Das Interface bietet folgende Methoden an:

IMobileFileServices

void connectToServer()
Diese Methode Initialisiert die Verbindung zum Server. Sie wird an die Methode connectToServer von IMobileCommunicationServices weitergeleitet. Wenn der Rückgabewert der connectToServer Methode des IMobileCommunicationServices Interface false ist, bedeutet das, dass auf dem Smartphone Malware gefunden wurde. In diesem Fall löscht der MobileFileManager alle Dateien, die sich im Arbeitsverzeichnis der Anwendung befinden. Die Anwendung gibt die Meldung an den Benutzer weiter. Danach beendet sie sich. Wenn die Methode jedoch true zurückgibt, war der Scan negativ und die Verbindung wird aufgebaut.

void disconnectFromServer()
Mit dieser Methode beendet man die VPN Verbindung. Sie wird an die disconnectFromServer Methode des IMobileCommunicationServices Interface weitergeleitet.

void createFile(Zeichenkette path, Zeichenkette fileName)
Diese Methode erstellt eine Datei mit dem Namen fileName in dem Verzeichnis path.

void openFile(Zeichenkette path)
Diese Methode öffnet eine Datei mit dem Pfad path. Das Öffnen der Dateien erfolgt über externe Apps wie z.B. PDF Reader. Hier wird überprüft, ob die Datei sich auf dem Smartphone befindet und diese aktuell ist. Sollte es nicht der Fall sein, wird die Datei beim Server angefordert.

void saveFile(Zeichenkette path, Datei file)
Speichert die Änderungen der Datei file mit dem Pfad path. Außerdem sendet diese Methode die Datei an den Server. Hierfür wird sendFileToServer(Zeichenkette command, Datei file) benutzt. Hierbei ist wieder zu beachten, dass die sendFileToServer Methode true und false zurückgeben kann. Wenn true zurückgegeben wird, wurde die Datei an den Server geschickt. Sollte jedoch false zurückgegeben werden, bedeutet das, dass in der Datei Malwaregefunden wurde. In diesem Fall werden, wie bei der connectToServer Methode, alle Unternehmensrelevanten Dateien im Arbeitsverzeichnis gelöscht und die Anwendung beendet.

void deleteFile(Zeichenkette path)
Diese Methode löscht die Datei mit dem Pfad path. Außerdem wird, durch die deleteFileFromServer Methode des IMobileCommunicationServices Interface, die Datei auf den Server ebenfalls gelöscht.

void getFileFromServer(Zeichenkette path)
Diese Methode fordert, über das IMobileCommunicationServices Interface, eine Datei mit dem Pfad path beim Server an. Dies geschieht über die receiveFileFromServer Methode.

AndroidAdapter
Der AndroidAdapter ist der Adapter zwischen der Androidactivity und des Anwendungskerns des Clients. Der Adapter implementiert das IAdatpterToApplicationCore Interface, welches alle Methoden des IMobileFileServices anbietet und zusätzlich noch eine Methode zum Zusammenbau des Anwendungskerns.

void buildApplicationCore(AssetManager assetManagers, byte[] userPw)
Die zusätzliche Methode des Adapters zum Zusammenbau des Anwendungskerns. Der Anwendungskern wird zusammengebaut, indem diese Methode, mit Hilfe des AssetManagers, die Configdatei ausliest und dann die Komponenten des Anwendungskerns erstellt. Das userPw ist das tatsächliche Benutzerpasswort in Bytearrayform. Es wird in der Methode zu dem Hash umgewandelt, der zum Verschlüsseln der Dateien nötig ist.

5.2.2 Implementierung der Komponenten

Die Komponenten wurden, wie der Server, in Java umgesetzt. Die Implementierung erfolgte so wie sie im Entwurf beschrieben wurde. Bei der Clientanwendung wird ebenfalls den Komponenten, die Daten aus der Config lesen müssen, nur der ConfigFileReader per Konstruktor übergeben. Das Exceptionhandling erfolgt genau so wie beim Server, d.h. Diese werden in Technical- und CryptographicExceptions gekapselt und weitergereicht.

MobileExternalServicesKomponente

Die ExternalServicesKomponente ist bei der Prototypimplementierung ebenfalls komplett gemockt. Sowohl VPN als auch der AntiMalwareAdapter haben keine Funktionalität.

MobileCryptographicKomponente

Die MobileCryptographicKomponente besteht, beim Client, aus zwei Interfaces und einer Klasse, die beide Interfaces implementiert. Die Interfaces sind ICryptographicServicesForCommunication und ICryptographicServicesForFiles.
Die Verschlüsslung der Nachrichten erfolgt genau so, wie bei der CryptographicKomponente des Servers. Hierfür wird das ForCommunication Interface verwendet. Der einzige Unterschied hierbei ist, dass die erste Nachricht mit dem Public-Key des Servers verschlüsselt wird. Hier ein Codebeispiel dazu:

```
private byte[] encryptDataWithPublicKey(byte[] data) throws CryptographicException {
                    ////Key einlesen
                    DataInputStream fileIn = new DataInputStream(new FileIn-
                    putStream(keyPath + "public.key"));
                    int pubKeyLen = fileIn.readInt();
                    byte[] rawKey = new byte[pubKeyLen];
                    fileIn.read(rawKey);
                    PublicKey publicKey = KeyFacto-
                    ry.getInstance(asymmetricAlgorithm).generatePublic(new
                    X509EncodedKeySpec(rawKey));
                    Cipher rsa;
                    rsa = Cipher.getInstance(asymmetricAlgorithm);
                    rsa.init(Cipher.ENCRYPT_MODE, publicKey);
                    return rsa.doFinal(data);
```

Bei diesem Code wir der Public-Key ausgelesen und dann zum Verschlüsseln der Daten benutzt.
Der Randomwert, der in der ersten Nachricht enthalten ist, wird zuerst mit dem Userpasswort und danach nochmal mit dem Public-Key des Servers verschlüsselt.
Die ICryptographicServicesForFiles wird im Kapitel 5.2.3 genauer erläutert.

MobileCommunicationKomponente

Anders als beim Server ist die MobileCommunicationKomponente nicht die zentrale Komponente. Sie dient zur Kommunikation zum Server und wird von der FileManagerKomponente verwendet. Sie bietet Methoden zum Zugriff auf Dateien oder Daten, die sich auf dem Server befinden. Die Komponente benutzt ebenfalls einen ConnectionManager, der die Verbindungsart abstrahiert.

Die Methoden des ConnectionManagers sind dieselben wie die des Servers, da hier ebenfalls TCP verwendet wird.
Außerdem bietet die MobileCommunicationKomponente die connectToServer Methode an, die die Initialisierung der Kommunikation zum Server abhandelt. Hier nun der Code zum Ablauf des Verbindungsaufbaus:

```
public boolean connectToServer() throws TechnicalException, CryptographicException {
        try {
                initialConnectionmanager = new MobileConnectionManager(new So-
                cket(host,inittialPort));
        } catch (IOException e) {
                throw new TechnicalException("Verbindung zum Initialport fehlgeschla-
                fen. Fehler: " + e.getMessage());
        }
        String initialMsg = "KEYEX " + deviceID;
         ////Secure Random Zahl auswürfeln
        byte[] randomBytes = cryptographicSer-
        vices.getSecureRandom(symmetricKeyLength);
        ////Verschluesseln und versenden der Nachricht
        byte[] encMsg = cryptographicSer-
        vices.encryptMessageWithPublicKey(initialMsg.getBytes());
        initialConnectionmanager.sendData(encMsg);
        ////Verschluesseln und Versenden des Sessionkeys
        byte[] symEncKey = cryptographicServices.encryptSessionKey(randomBytes,
        userPw);
        byte[] encKey = cryptographicServices.encryptKeyWithPublicKey(randomBytes,
        userPw);
        initialConnectionmanager.sendData(encKey);
        ////Antwort emfpangen
        byte[] encResponse = initialConnectionmanager.receiveDataFromServer();
        byte[] encResponceKey = initialConnectionmanager.receiveDataFromServer();
        String responseMsg = cryptographicServices.decryptMessage(encResponse,
        userPw);
        byte[] decSessionKey = cryptographicServices.decryptSessionKey(encResponceKey,
        userPw);
```

Bei diesem Abschnitt des Codes wird zuerst die Verbindung zum Initialserverport aufgebaut. Als nächstes folgt dann die Erzeugung des Sessionkeys(randomBytes). Danach werden die Nachricht und der Sessionkey verschlüsselt und an den Server gesendet. Zuletzt wird hier die Antwort des Servers empfangen.

Als nächstes folgt dann der Code zur Abarbeitung der Antwort und Verschicken und Empfangen der Sperranfragen.

```
if (Arrays.equals(randomBytes, decSessionKey) && initialMsg.equals(responseMsg)) {
        boolean virusFound = externalServices.scanPhoneforMalware();
        if (!virusFound) {
                byte[] encSPANF = cryptographicSer-
                vices.encryptMessageWithSessionKey("SPANF", randomBytes);
                byte[] encDeviceID = cryptographicSer-
                vices.encryptMessageWithSessionKey(deviceID, randomBytes);
                initialConnectionmanager.sendData(encSPANF);
                initialConnectionmanager.sendData(encDeviceID);
                byte[] encSPANFResponse = initialConnectionmana-
                ger.receiveDataFromServer();
                byte[] encSPANFDeviceID = initialConnectionmana-
                ger.receiveDataFromServer();
                String decSPANFResponse = new String(cryptographicServices
                        .decryptDataWithSessionKey(encSPANFResponse, randomBytes));
                String decSPANFDeviceID = new String(cryptographicServices
                        .decryptDataWithSessionKey(encSPANFDeviceID, randomBytes));
```

Hier wird die Antwort des Servers abgearbeitet und es wird überprüft ob die Nachricht die verschickt wurde, vom Server richtig entschlüsselt wurde. Außerdem wird hier der komplette Scan des Smartphones angestoßen. Sollte hierbei Malware gefunden werden, sendet der Client eine SPAUF an den Server und beendet sich. Wenn keine Malware gefunden wurde, wird eine SPANF an den Server gesendet. Diese ist mit dem Sessionkey verschlüsselt.

```
if (decSPANFResponse.equals("SPANF") && decSPANFDeviceID.equals(deviceID)) {
        externalServices.connectToVPN();
        try {
                String closeMsg = new
                String(cryptographicServices.decryptDataWithSessionKey(
                initialConnectionmanager.receiveDataFromServer(),randomBytes));
                initialConnectionmanager.closeConnection();
                connectionmanager= new MobileConnectionManager(new Socket(host,port));
        } catch (IOException e) {
                throw  new TechnicalException("Fehler beim Verbinden mit dem Server
                über VPN. Fehler: "+e.getMessage());
        }
```

Bei diesem Abschnitt wird überprüft, ob eine SPANF vom Server zurückgesendet wurde. Sollte dies nicht der Fall sein, gibt die Methode false zurück und die Verbindung war nicht erfolgreich, da auf eine SPANF Anfrage immer eine SPANF Antwort folgen muss. Sollte eine SPANF vom Server zurückkommen, wird eine CLOSE Nachricht an den Server geschickt, da die Initialisierung des Verbindungsaufbaus über den Initialport erfolgreich war. Zum Schluss wird das VPN-Tool benutzt, um eine VPN-verbindung zum Server aufzubauen.

Nach dem die Verbindung zum VPN erfolgreich hergestellt wurde, wird die HALLO Nachricht an den Server geschickt, um das Gerät zu identifizieren. Hier der Codeausschnitt dazu:

```
connectionmanager.sendCommand("HALLO");
connectionmanager.sendString(deviceID);
String response = new String(connectionmanager.receiveCommandFromServer());
if (response.equals("HALLO")) {
        String receivedDeviceID = new
        String(connectionmanager.receiveDeviceIDFromServer());
                if (receivedDeviceID.equals(deviceID)) {
                        return true;
        }
```

Es wird die „HALLO“ Nachricht, mit der GeräteID, an den Server gesendet und dieser antwortet mit derselben Nachricht. Wenn die Nachrichten übereinstimmen, ist die Verbindung zum Server hergestellt und die connectToServer Methode gibt true zurück. Dies ist der einzige Zweig im Code, der bei dieser Methode true zurückgibt.

5.2.3 Verwaltung und Verschlüsselung von Dateien

Die zentrale Komponente beim Client ist die FileManagerKomponente. Diese nutzt die anderen Komponenten, um z.B. Dateien beim Server anzufordern. Außerdem ist die FileManagerKomponente zuständig für die Verwaltung der Dateien auf dem Smartphone.
Die Dateien werden im \files\ Verzeichnis verschlüsselt abgelegt. Sobald eine Datei geöffnet werden soll, stößt die FileManagerKomponente die CryptographicKomponente an, die Datei zu entschlüsseln. Diese wird dann im \tmp\ Verzeichnis abgelegt. Sobald die saveFile Methode aufgerufen wird, wird die Datei wieder verschlüsselt und in \files\ abgelegt und aus \tmp\ gelöscht. Hier nun ein Beispiel zum Verschlüsseln einer Datei:

```
public File encryptFile(String path,byte[] key) throws TechnicalException {
        File file = new File("tmp\\"+path);
        File newfile = new File("files\\"+ path);
        newfile.getParentFile().mkdirs();
        FileInputStream fis = null;
        try {
                fis = new FileInputStream(file);
                FileOutputStream fos =new FileOutputStream(newfile);
                ////AES Key erzeugen aus Userpassword
                SecretKeyFactory factory = SecretKeyFactory.getInstance(hashCodeMode);
                            KeySpec spec = new PBEKeySpec(new
                String(key).toCharArray(),fileSalt, hashCodeIterationsForFiles, symme-
                tricKeyLen);
                SecretKey tmp = factory.generateSecret(spec);
                SecretKey secret = new SecretKeySpec(tmp.getEncoded(), symmetricAlgo-
                rithm);
                Cipher cipher = Cipher.getInstance(symmetricMode);
                IvParameterSpec iv = new IvParameterSpec(ivBytes);
                cipher.init(Cipher.ENCRYPT_MODE, secret, iv);
                CipherOutputStream cout=new CipherOutputStream(fos, cipher);
```

```
                byte[] buf = new byte[1024];
                int read;
                while((read=fis.read(buf))!=-1)
                        cout.write(buf,0,read);
                fis.close();
                cout.flush();
                cout.close();
                return newfile;
        } catch (FileNotFoundException e) {
```

Hierbei wird die Datei aus dem \tmp\ Verzeichnis geladen und durch den CipherOutputStream in \files\ geschrieben. Verschlüsselt wird hierbei ebenfalls mit AES. Der Schlüssel ist das Userpassworthash mit einem „Salt“, der aus der Config gelesen wird. Die Parameter zur Erzeugung des Hashs werden ebenfalls aus der Configdatei ausgelesen.
Der oben beschriebene Code wird z.B. in der saveFile Methode aufgerufen. Hier der Code:

```
public void saveFile(String path, File file) throws TechnicalException {
        boolean sendSuccess = communicationServices.sendFileToServer(path,file);
        if(sendSuccess){
                cryptographicServicesForFiles.encryptFile(path,userPw);
                System.gc();
                file.delete();
        } else{
                throw new TechnicalException("Datei konnte nicht an den Server gesen-
                det werden");
        }

}
```

Es wird zuerst die Datei an den Server gesendet. Die sendFileToServer Methode der ICommunicationServices scannt die Datei auf Malware und sendet diese dann an den Server.
Sollte das Senden erfolgreich sein, wird die Datei verschlüsselt und in \files\ gespeichert. Die alte entschlüsselte Datei wird gelöscht. System.gc() wird hier aufgerufen, da Java die Datei nicht löschen kann, wenn noch die Referenzen sich irgendwo im Speicher befinden.
Die Restlichen Methoden der Komponente sind sehr ähnlich aufgebaut und delegieren meist an Methoden von anderen Komponenten weiter.

5.2.4 Integrierung des Anwendungskerns in Android

Da der Anwendungskern des Clients komplett, ohne von Android benötigten Code, implementiert wurde, wird ein Adapter zwischen den Anwendungskern und der Android Activity implementiert. Dieser Adapter bietet die in 5.2.1, unter Punkt AndroidAdapter, erwähnten Methoden an.

Erstellung des Anwendungskerns

Die Erstellung des Anwendungskerns wird durch die buildApplicationCore Methode angestoßen. In dieser Methode wird der Anwendungskern erstellt, indem durch den AssetManager die Configdatei ausgelesen wird und die nötigen Informationen an die Komponenten gegeben werden. Der AssetManager ist eine von Java angebotene Klasse, die im assets Ordner der Androidanwendung Dateien auslesen kann. Außerdem wird hier auch das Benutzerpasswort in den Hash umgewandelt, womit die Dateien und der Randomwert der Initialkommunikation verschlüsselt werden. Dieser Hash ist auch dem Server bekannt.
Der Code der Methode:

```
public void buildApplicationCore(AssetManager assetManager,byte[] userPw) throws
TechnicalException, CryptographicException {
        this.assetManager =assetManager;
        this.cfgReader = new ConfigFileReader(assetManager,"Client.cfg");
        this.cryptographicServicesForCommunication = new MobileCryptographicMana-
        ger(cfgReader);
        this.cryptographicServicesForFiles = (ICryptographicServicesForFiles) crypto-
        graphicServicesForCommunication;
        this.externalServices = new MobileExternalServices();
        MessageDigest cript = null;
        try {
                cript = MessageDigest
                .getInstance(cfgReader.readProperty("userHashAlgorithm"));
         } catch (NoSuchAlgorithmException e) {
                throw new CryptographicException("Algorithmus zum Hashbilden existiert
                nicht. Fehler: " +e.getMessage());
         }
         cript.reset();
         cript.update(userPw);
         byte[] userPwHash = cript.digest();
         this.communicationServices = new MobileCommunicationManager(externalServices,
                cryptographicServicesForCommunication,userPwHash,
                cfgReader.readProperty("deviceId"),
                Integer.parseInt(cfgReader.readProperty("randomByteLen")),
                cfgReader.readProperty("host"),
                Integer.parseInt(cfgReader.readProperty("initialCommunicationPort")),
                Integer.parseInt(cfgReader.readProperty("communicationPort")));
        this.fileServices = new MobileFileManager(communicationServices,
        cryptographicServicesForFiles,userPwHash);
    }
```

Durch diese Methode werden alle Referenzen, die zum bearbeiten sämtlicher Anfragen nötig sind, gesetzt bzw. erstellt.

Mainthread und Tasks

Da eine Androidactivity nur einen Mainthread hat, ist es nicht erlaubt Netzwerkkommunikation direkt auf dem Mainthread zu betreiben. Deshalb wird bei der Implementierung für Methoden des IAdatpterToApplicationCore, die über das Netzwerk/Internet kommunizieren müssen, ein AsyncTask erstellt. Ein AsyncTask bietet mehr Funktionen als ein einfacher Thread, man kann z.B. den Zwischenstand des Tasks abfragen. Da Methoden wie z.B. connectToServer und disconnectFromServer, die gleiche Signatur(außer dem Namen) besitzen, wurde für diese beiden Methoden nur eine Taskimplementierung erstellt, die durch Reflections die richtige Methode aufruft.
Hier ein Codebeispiel:

```
public class ConnectionTask extends AsyncTask<Method, Void, Boolean> {

            private IMobileFileServices fileServices;

            public ConnectionTask(IMobileFileServices fileServices) {
                this.fileServices = fileServices;
            }

            @Override
            protected Boolean doInBackground(Method... methods) {
                try {
                    return (Boolean) methods[0].invoke(fileServices);
                } catch (…)
            }
        }
```

Der Task kriegt hier die IMobileFileServices Referenz im Konstruktor übergeben. Der Aufruf des Tasks geschieht durch:

```
new CconnectionTask(fileServices)
.execute(fileServices.getClass().getMethod("connectToServer"))
```

Hier sieht man auch, wie die Methode durch Reflections übergeben wird und dann im Task selbst, durch invoke, aufgerufen wird. Falls ein Rückgabewert vom Task erwartet wird, ist es möglich .get() darauf aufzurufen.
Es gibt insgesamt 6 Tasks: ConnectionTask für die connect und disconnect Methoden, den DeleteFileTask für das Löschen einer Datei auf dem Server, den GetFileListTask für das anfordern der Dateiliste, den GetFileTask für das anfordern einer Datei und den OpenFileTask, da beim Öffnen einer Datei. Der OpenFileTask muss als Task implementiert werden, da die Datei die geöffnet werden soll, sich evtl. noch nicht auf dem Smartphone befindet. Dies hat zur Folge, dass die Datei vom Server angefordert werden muss. Außerdem existiert noch den SaveFileTask, der eine Datei an den Server schickt.
Es wäre möglich gewesen einen großen Task mit einem enum oder einer if else Kaskade zu implementieren oder mehrere Tasks zusammenzulegen, aber das würde die Aufgaben der einzelnen Tasks vermischen und den Code unübersichtlicher machen.

GUI und Fragments

Die Android Activity erstellt, in der onCreate Methode, den Adapter und setzt das Login-Fragment als AusgangsGUI.

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        connected = false;
        setContentView(R.layout.main);
        adatpterToApplicationCore=new AndroidAdapter();
        LoginFragment newFragment = new LoginFragment();
        FragmentTransaction transaction = getSupportFragmentMana-
        ger().beginTransaction();
        transaction.add(R.id.fragment_container, newFragment,"Login");
        transaction.addToBackStack(null);
        transaction.commit();
    }
```

Die setContentView Methode setzt die „RahmenGUI“ der Activity, in diesem fall eine Leere GUI, die den kompletten Bildschirm ausfüllt. Danach wird der Adapter zum Anwendungskern erstellt. Anschließend wird das LoginFragment erstellt und als AusgangsGUI gesetzt.

Die GUI Layouts werden, bei Android, in XML geschrieben. Diese XML Dateien werden dann bei Erstellung der GUI geladen. In der XML Datei legt man auch die Methodennamen bei z.B. Buttonclicks an.

```
<Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Login"
      android:id="@+id/loginButton" android:layout_gravity="center_horizontal"
      android:onClick="clickLoginButton"/>
```

In diesem XML Code Beispiel würde die Methode public void clickLoginButton(View view) in der Activity oder dem Fragment aufgerufen werden, wenn man auf den Login Button klickt.Für die Erstellung des Fragments wird eine eigene Klasse benötigt, die von Fragment erbt. Hier ein Codebeispiel einer solchen Klasse:

```
public class LoginFragment extends Fragment {


        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle
        savedInstanceState) {
                setHasOptionsMenu(false);
                return inflater.inflate(R.layout.login_view, container, false);
        }


}
```

Diese Klasse repräsentiert in der Implementierung das Loginfenster. Die onCreateView Methode wird beim Wechsel zwischen Fragments aufgerufen. Hier wird durch den Inflater eine View erstellt und zurückgegeben. Diese View hat das Layout R.layout.login_view.
Zum Wechseln zwischen Fragments benutzt man einen FragmentManager. Beispielcode:

```
private void swapToFileChooser(){
        fileChooser = new FileChooserFragment();
        FragmentTransaction transaction = getSupportFragmentMana-
        ger().beginTransaction();
        transaction.replace(R.id.fragment_container, fileChooser);
        transaction.addToBackStack(null);
        transaction.commit();
    }
```

In diesem Codebeispiel wird ein FileChooserFragment erstellt und durch den FragmentManager wird vom derzeitigen Fragment zu dem FileChooserFragment gewechselt. Dies wird durch eine Transaction abgehandelt.

Der FileChooser ist das zentrale GUI-Element der Anwendung. Hier werden die Dateien vom Server angezeigt. Für den FileChooser ist ebefalls eine Fragment Klasse nötig. Der Unterschied zum LoginFragment ist hierbei, dass der FileChooser ein ListFragment ist. ListFragments sind ebenfalls Fragments, außer dass Diese eine Liste in der GUI darstellen. Um die Liste darstellen zu können, benötigt das ListFragment einen speziellen ArrayAdapter, die man mit setListAdapter(adapter) setzt. Für diese Anwendung wurde ein „Custom“ Adapter implementiert, der FileData Objekte verwaltet.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;
        if (v == null) {
                LayoutInflater vi = (LayoutInfla-
                ter)c.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                v = vi.inflate(id, null);
        }
        final FileData o = items.get(position);
        if (o != null) {
        TextView t1 = (TextView) v.findViewById(R.id.TextView01);
        TextView t2 = (TextView) v.findViewById(R.id.TextView02);

        if(t1!=null)
             t1.setText(o.getName());
        if(t2!=null)
             t2.setText(o.getData());

        }
        return v;
}
```

Diese Methode wird aufgrund von Vererbung von ArrayAdapter<FileData> verlangt. Hier wird die View zur Darstellung der einzelnen Elemente der Liste erzeugt. Die FileData Klasse beinhaltet nur Strings, die die Informationen der Datei darstellt.

Des Weiteren bietet die Activity ein Kontextmenü, wenn man auf die Menütaste des Smartphones klickt. Dieses Menü bietet die Möglichkeit die Dateiliste zu aktualisieren, eine neue Datei anzulegen oder die Anwendung zu beenden. Für dieses Menü wird lediglich eine neue XML Datei, im Unterordner menu des Layout Ordners, angelegt.
Für die Erstellung einer neuen Datei wird ebenfalls ein neues Fragment angelegt. Dieses Fragment ist äquivalent zum LoginFragment, mit dem Unterschied, dass es ein anderes Layout lädt.

5.2.5 Tests

Die Clienttests sind ebenfalls in Komponententests und Verbund-Tests unterteilt.
Die MobileCryptographicKomponente ist hierbei ebenfalls die getestete Komponente, da hier, wie beim Server, die anderen Komponenten nur entweder Aufrufe delegieren oder TCP Daten empfangen bzw. versenden. Ähnlich wie beim Server wurde auch der Verbund-Test geschrieben, mit dem Unterschied, dass für den Verbund-Test der Server laufen muss. Da der Client, außer dem Server, keine anderen externen Programme benutzt, wurde der Verbund-Test auf diese Weise implementiert. Im Verbund-Test wird die Kommunikation und das Versenden und Empfangen von Dateien an den Server getestet. Außerdem wird auch das Erstellen, Speichern, Löschen und Ver-/Entschlüsseln von Dateien im Verbund-Test getestet.

6 Zusammenfassung

Das Ziel der Arbeit war es ein Konzept auszuarbeiten, dass es den Benutzer erlaubt private Androidgeräte, für Verarbeitung von Unternehmensdaten, sicher zu verwenden. Dieses Ziel wurde mit dem Konzept erreicht. Der Benutzer kann sein eigenes Smartphone für die Verarbeitung von Unternehmensdaten verwenden, indem er die App die hier beschrieben wurde startet. Die Unternehmensdaten werden sicher und verschlüsselt auf den Smartphone gelagert und der Dateitransfer geschieht nur über VPN. Da VPN die Daten verschlüsselt sendet und die Daten auch auf dem Smartphone nur in verschlüsselter Form liegen, ist es für Angreifer nicht ohne weiteres möglich diese Daten zu lesen. Bei Verlust oder Malwarebefall kann das Unternehmen das löschen sämtlicher Unternehmensdaten veranlassen. Hierbei werden aber nur die Unternehmensdaten gelöscht.
Das Smartphone bleibt, für den Benutzer selber, komplett frei benutzbar und unüberwacht. Im Vergleich zu den anderen hier erwähnten BYOD Konzepten ist diese Lösung nicht so umfangreich wie z.B. die Virtualisierung aus 2.1.2, da bei dem Konzept dieser Arbeit nur die Datenverarbeitung möglich ist. Die Lösung ist aber schnell realisierbar und muss nicht aufwendig konfiguriert werden.
Der aus dem Konzept entstandene Entwurf kann auch für andere Smartphones wie IPhone oder Windows Mobile benutzt werden, da der Entwurf des Clients sprach- und plattformunabhängig geschrieben wurde. Die im Entwurf entstandenen Komponenten können schnell und einfach bei Bedarf ausgetauscht werden.
Durch die Implementierung des Entwurfs wurde die Machbarkeit dieser Arbeit aufgezeigt. Die Implementierung ist aber nur prototypisch, d.h. es wurden Komponenten wie AntiMalwareScanner und VPN gemockt und nur der Kern wurde implementiert. Der Kern besteht aus der Kommunikation zwischen Server und Client, Kommunikation zwischen Server und Fileserver und das Ver- und Entschlüsseln der Daten auf dem Smartphone. Außerdem wäre eine bessere Bedienung/Usability der GUI bzw. der Anwendung bei einer nicht prototypischen Implementierung nötig gewesen.

7 Literaturverzeichnis

[ACK2013] Elise Ackerman:
“The Bring-Your-Own-Device Dilemma“,
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6565553
(Stand: 16.12.2013)

[ABD2012] Abdelmajid Lakbabi, Ghizlane Orhanou, Said EI Hajji:
“VPN IPSEC & SSL Technology”,
2012 Next Generation Networks and Services NGNS, 2-4 December 2012 Portugal

[AND2014] Android Developer Starter Guide
http://developer.android.com/training/index.html
(Stand: 21.03.2014)

[BLA2014] BlackBerry:
Offizielle BlackBerry Seite zum Balance Service
http://de.blackberry.com/business/software/blackberry-balance.html
(Stand:26.03.2014)

[CHU2012] Sean Chung, Sam Chung, Teresa Escrig, Yan Bai Barbara Endicott-Popovsky:
“2TAC: Distributed Access Control Architecture for ‘Bring Your Own Device’ Security“,
2012 ASE/IEEE International Conference on BioMedical Computing

[FAN2012] P. Fanfara, E. Danková and M. Dufala:
“ Usage of Asymmetric Encryption Algorithms to Enhance the Security of Sensitive Data in Secure Communication“,
Applied Machine Intelligence and Informatics (SAMI), 2012 IEEE 10th International Symposium

[GEH2002] Edhard F. Gehringer:
“Choosing Passwords: Security and Human Factors“,
Technology and Society, 2002. (ISTAS'02). 2002 International Symposium

[KHA2012] Sohail Khan, Mohammad Nauman, Abu Talib Othman, Shahrulniza Musa: "How Secure is your Smartphone: An Analysis of Smartphone Security Mechanisms",
2012 Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)

[MS2014] Microsoft
Microsoft STRIDE und DREAD:
DREAD: http://msdn.microsoft.com/en-us/library/ff648644.aspx
STRIDE: http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
(Stand: 07.06.2014)

[NAD2005] Aamer Nadeem, Dr M. Younus Javed: "A Performance Comparison of Data Encryption Algorithms", Information and Communication Technologies, 2005. I-CICT 2005

[SAM2014] Samsung
Offizielle Seite zum SAFE Service von Samsung
http://www.samsung.com/de/business/solutions-services/mobile-solutions/security/safe
(Stand: 26.03.2014)

[SCA2012] Antonio Scarfò: " New security perspectives around BYOD", 2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications

[TIT2013] Dennis Titze, Philipp Stephanow, Julian Schütte:
"A Configurable and Extensible Security Service Architecture for Smartphones",
2013 27th International Conference on Advanced Information Networking and Applications Workshops

[WAN2012] Yong Wang, Kevin Streff, and Sonell Raman:
"Smartphone Security Challenges",
Computer (Volume: 45, Issue: 12)
http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6383143
(Stand: 16.12.2013)

8 Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis:

Tabellenverzeichnis:

9 Anhang

Im Anhang dieser Arbeit befindet sich eine DVD mit dieser Arbeit in PDF-Form und außerdem den Code des Prototypen und den für den Prototypen benutzten Tools.

Inhalt der DVD:

- Bachelorarbeit als PDF im Ordner „PDF“
- Source Code des Prototypen im Ordner „Code“
- Android SDK inkl. Android Emulator im Ordner „Android SDK“
- IDE: IntelliJ Community Edition 13.0.2 im Ordner „IDE“
- Java Library: commons.net 3.3(Java FTP Client) im Ordner „Java Libraries“
- Abbildungen in PNG bzw. JPG Format im Ordner „Abbildungen“

Versicherung über Selbstständigkeit

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

Hamburg, den _____ _____