UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Institut für Wirtschafts- und Verwaltungsinformatik

NFCKeyExchange
Schlüsselaustausch mittels NFC

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
vorgelegt von

Stefan Becker

Erstgutachter:	Prof. Dr. J. Felix Hampe Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Dipl.- Inform. Nico Jahn Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im August 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Kurzbeschreibung

Aufgrund der zunehmenden Verbreitung des mobilen Internets, können E-Mails direkt von mobilen Geräten gesendet und empfangen werden. Inhalte digitaler Kommunikation sollten verschlüsselt werden, um zu verhindern, dass sie abgefangen und manipuliert werden. Entsprechende Verfahren setzen kryptographische Schlüssel ein, die zuvor ausgetauscht werden müssen. Es muss sichergestellt sein, dass ein kryptographischer Schlüssel tatsächlich der Person zugeordnet ist, zu der er angeblich gehört. Im Rahmen dieser Arbeit wurde ein Konzept für eine Smartphone-Anwendung entwickelt, um kryptographische Schlüssel im persönlichen Kontakt auszutauschen. Das Konzept besteht aus dem Entwurf eines komponentenbasierten Frameworks, mit dem sich Daten sicher austauschen lassen. Das Framework wurde anschließend erweitert und als Grundlage für die Entwicklung einer Smartphone-Anwendung eingesetzt. Die Anwendung ermöglicht es kryptographische Schlüssel zu erzeugen, zu verwalten und im persönlichen Kontakt auszutauschen. Für den Austausch wird die Near Field Communication genutzt. Durch implementierte Sicherheitsmechanismen ist gewährleistet, dass der Schlüsselaustausch weder abgehört, noch gezielt manipuliert werden kann. In Zukunft können das Konzept und die Anwendung erweitert und angepasst werden, um sie in anderen Arbeiten einzusetzen.

Abstract

Due to the increasing pervasiveness of the mobile web, it is possible to send and receive mails with mobile devices. Content of digital communication should be encrypted to prevent eavesdropping and manipulation. Corresponding procedures use cryptographic keys, which have to be exchange previously. It has to be ensured, that a cryptographic key really belongs to the person, who it is supposedly assigned to. Within the scope of this thesis a concept for a smartphone application to exchange cryptographic keys was designed. The concept consists of a specification of a component-based framework, which can be used to securely exchange data in general. This framework was extended and used as the basis for a smartphone application. The application allows creating, managing and exchanging cryptographic keys. The Near Field Communication is used for the exchange. Implemented security measures prevent eavesdropping and specific manipulation. In the future the concept and the application can be extended and adjusted to be used in other contexts.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Motivation für NFCKeyExchange

Dieses Kapitel erklärt, welches Ziel die Arbeit verfolgt. Zuerst werden aktuelle Konzepte zur sicheren, digitalen Kommunikation erläutert. Anschließend werden sie mit dem in der Arbeit entwickelten Konzept verglichen.

1.1 Konzepte für sichere, digitale Kommunikation

Die zunehmende Verbreitung [IDC] von Smartphones rückt die Bedeutung des mobilen Internets immer mehr in den Vordergrund. Durch diese Entwicklung verändert sich die Art wie Menschen digital kommunizieren. Smartphone-Nutzer sind nicht länger an einen Desktop-Computer oder einen Laptop gebunden um E-Mails zu verfassen und abzurufen. Ein 2012 veröffentlichter Report [Kno12] von *Knotice*[1] bestätigt, dass E-Mails zunehmend über Smartphones abgerufen werden.

Diese Entwicklung ist zum Beispiel für Unternehmen interessant: Die Flexibilität und Erreichbarkeit ihrer Mitarbeiter nimmt zu. Tauschen Mitarbeiter Informationen unverschlüsselt über E-Mails aus, besteht jedoch die Gefahr, dass ein Angreifer die Nachrichten liest und manipuliert. Dies stellt in einem Unternehmensumfeld, in dem sensible Daten ausgetauscht werden müssen, ein Problem dar. Ebenso kann es für Privatpersonen sinnvoll sein ihre E-Mail-Kommunikation abzusichern. Nur so kann das Mitlesen durch einen Angreifer verhindert werden.

[1] Knotice: http://www.knotice.com/, Zuletzt abgerufen am 25.08.2013.

Die beschriebe Problematik besteht nicht nur für die Kommunikation per E-Mail, sondern gilt ganz allgemein für alle Formen der digitalen Kommunikation.

Um das Problem zu lösen, können die Inhalte verschlüsselt werden. Eine im Juli 2013 von *BITKOM*[2] veröffentliche Pressemeldung [BIT13b] zeigt, dass 76 Prozent der deutschen Unternehmen im IT- und Telekommunikationsbereich Daten und E-Mails verschlüsseln. Die Pressemeldung verweist außerdem auf eine 2012 erschienene Umfrage. In der Umfrage wurden deutsche Unternehmen befragt. Fünfundzwanzig Prozent der Unternehmen gaben an, dass sie vertrauliche Dokumente nicht mittels E-Mail versenden. In einer ebenfalls 2013 durchgeführten Umfrage von BITKOM [BIT13a] gaben 43 Prozent der befragten Internetnutzer an wichtige oder vertrauliche Dokumente nicht per E-Mail sondern per Post zu versenden. Gleichzeitig gaben nur 6 Prozent der Befragten an eine Verschlüsselungssoftware für ihre E-Mails zu verwenden. Als Hauptgründe wurden angegeben, dass sich die Befragten mit Verschlüsselungssoftware nicht auskennen und dass Personen mit denen Daten ausgetauscht werden sollen keine Verschlüsselung nutzen. Demnach besteht momentan eine Diskrepanz zwischen dem Einsatz von Verschlüsselung im Unternehmen und im privaten Bereich. Der Einsatz im Unternehmen könnte jedoch dazu beitragen, das Wissen der eigenen Mitarbeiter zu steigern und somit die Motivation zu erhöhen auch im Privatbereich zu verschlüsseln.

Entsprechende Verfahren arbeiten mit kryptographischen Schlüsseln. Diese werden eingesetzt, um Informationen auf Seiten des Senders zu ver- und auf Seiten des Empfängers zu entschlüsseln. Dabei wird zwischen symmetrischer und asymmetrischer Verschlüsselung unterschieden. Bei der symmetrischen Verschlüsselung verwenden Sender und Empfänger den selben kryptographischen Schlüssel. Die Inhalte werden mit diesem verschlüsselt und entschlüsselt. Daraus ergeben sich zwei Nachteile. Zum einen muss der kryptographische Schlüssel geheim gehalten werden, da ein Angreifer ansonsten in der Lage ist, die verschlüsselten Inhalte zu entschlüsseln. Zum anderen kann ein kryptographischer Schlüssel nicht für mehrere Kommunikationspartner verwendet werden. Ansonsten ist jeder Kommunikationspartner in der Lage, die ausgetauschten Kommunikationsinhalte der jeweils anderen zu entschlüsseln.

Die asymmetrische Verschlüsselung löst die beschriebenen Probleme. Anstelle eines gemeinsamen kryptographischen Schlüssels, besitzt jeder Kommunikati-

[2]BITKOM: http://www.bitkom.org/, Zuletzt abgerufen am 25.08.2013

onspartner sein eigenes Schlüsselpaar. Dieses besteht aus einem privaten und einem öffentlichen Schlüssel. Der öffentliche Schlüssel ist nicht geheim und sowohl dem Kommunikationspartner als auch einem möglichen Angreifer bekannt. Damit jemand Inhalte verschlüsselt an einen Kommunikationspartner senden kann, benötigt er dessen öffentlichen Schlüssel. Nachrichten, die mit dem öffentlichen Schlüssel verschlüsselt werden, lassen sich nur mit dem privaten wieder entschlüsseln. Außerdem kann mit dem privaten Schlüssel eine Signatur zu einem Inhalt erzeugt werden. Die Signatur lässt sich mit dem öffentlichen Schlüssel verifizieren. Dadurch kann überprüft werden, ob der ausgetauschte Inhalt verändert wurde.

Bevor eine Verschlüsselung möglich ist, müssen die benötigten kryptographischen Schlüssel ausgetauscht werden. Es muss sichergestellt sein, dass der Schlüssel, den der Sender für die Verschlüsselung verwenden soll, von dem intendierten Empfänger stammt. Dies gilt für symmetrische und asymmetrische Verschlüsselung gleichermaßen. Gelingt es einem Angreifer, diesen Schlüssel gegen einen eigenen auszutauschen, kann er die Kommunikationsinhalte entschlüsseln. Dadurch wird die Kommunikation kompromittiert. Aus dieser Problematik ergibt sich die Fragestellung, wie sich beim initialen Austausch kryptographischer Schlüssel deren Authentizität sicherstellen lässt.

Bisherige Ansätze sind *Public-Key-Infrastrukturen* und das *Web of Trust*. In einer Public-Key-Infrastruktur (PKI) werden Zertifikate ausgestellt, die einer Person einen kryptographischen Schlüssel zuordnen. Die Zertifikate stammen von einer Zertifizierungsstelle, der alle Nutzer der Public-Key-Infrastruktur vertrauen müssen. Die Zuordnung durch eine vertrauenswürdige Instanz stellt die Authentizität des kryptographischen Schlüssels sicher. Voraussetzung ist, dass die Zertifizierungsstelle vertrauenswürdig ist. Das Web of Trust verfolgt einen dezentralen Ansatz, bei dem Nutzer ihre öffentlichen Schlüssel gegenseitig signieren. Jeder Nutzer besitzt ein Schlüsselpaar bestehend aus einem privaten und einem öffentlichen Schlüssel. Nutzer können die öffentlichen Schlüssel anderer Nutzer mit ihrem eigenen privaten Schlüssel signieren. Dadurch bestätigen sie, dass der öffentliche Schlüssel tatsächlich zu dem entsprechenden Nutzer gehört. Je häufiger ein öffentlicher Schlüssel signiert wurde, um so vertrauenswürdiger ist er. Zusätzlich kann sowohl beim Web of Trust als auch bei Public-Key-Infrastrukturen eine manuelle Prüfung vorgenommen werden. Im Web of Trust wird mittels einer Hashfunktion eine verkürzte Zeichenfolge zu einem öffentli-

chen Schlüssel erzeugt. Diese wird als Fingerprint bezeichnet. Ein Nutzer kann zu einem erhaltenen öffentlichen Schlüssel einen Fingerprint erzeugen und den Besitzers des Schlüssels fragen, ob dieser mit seinem Fingerprint übereinstimmt. In einer Public-Key-Infrastruktur wird der Fingerprint für die kryptographischen Schlüssel der Instanz erstellt, welche Zertifikate erzeugt. Dadurch muss keine Prüfung zwischen den einzelnen Teilnehmern vorgenommen werden.

Sofern keine zusätzliche manuelle Prüfung vorgenommen wird, setzen beide Ansätze eine transitive Vertrauensbeziehung voraus. Im Falle einer PKI wird der Zertifizierungsstelle vertraut und im Falle eines Web of Trusts den anderen Nutzern. Eine Public-Key-Infrastruktur garantiert die Authentizität eines kryptographischen Schlüssels, sofern die Zertifizierungsstelle vertrauenswürdig ist und nicht manipuliert wird. Der Aufbau einer PKI ist allerdings mit organisatorischem Aufwand verbunden und eignet sich daher nur für größere Unternehmen oder Behörden. Der Ansatz des Web of Trust ist zwar weniger aufwendig, kann die Authentizität eines kryptographischen Schlüssels aber nicht vollständig garantieren.

1.2 Konzept zur Schlüsselübergabe mittels NFC

Im Rahmen dieser Arbeit soll ein weiterer Ansatz entwickelt werden, um sicherzustellen, dass ein kryptographischer Schlüssel von der Person stammt, zu der er angeblich gehört. Der Ansatz soll als Anwendung realisiert werden und für den mobilen Bereich geeignet sein. Da Smartphone-Nutzer ihre Geräte die meiste Zeit mit sich führen, soll ein spontaner Austausch im direkten Kontakt möglich sein. Damit folgt der Ansatz einer Idee, welche auch im Web of Trust angewendet wird, der Verifikation von Schlüsseln im persönlichen Kontakt. Somit setzt der Ansatz verglichen mit den bisherigen auf direktes Vertrauen anstelle von transitivem. Ein Austausch ohne direkten Kontakt wird in dieser Arbeit nicht betrachtet. Die Arbeit schränkt das Anwendungsfeld gegenüber den beiden vorgestellten Ansätze ein. Durch diese Einschränkung soll ein unkomplizierter Schlüsselaustausch ermöglicht und gleichzeitig die Authentizität der kryptographischen Schlüssel garantiert werden.

Die Umsetzung als mobile Anwendung fokussiert den sicheren Schlüsselaustausch und die Interoperabilität mit anderen Anwendungen. Sicher bedeutet in diesem Zusammenhang, dass übertragene kryptographische Schlüssel nicht mit-

gelesen und unbemerkt manipuliert werden können. Die Sicherheit wird durch den Einsatz von Kryptographie gewährleistet. Die Interoperabilität ergibt sich aus der vollständigen Spezifikation der Software. Für externe Programme werden Schnittstellen angeboten. Die Anwendung wird in zwei Schritten entworfen. Als Erstes wird ein Framework entworfen, mit dem Daten sicher ausgetauscht werden können. Das Framework beschränkt sich dabei nicht auf den mobilen Bereich, weshalb die entworfene Architektur auch für andere Anwendungen einsetzbar ist. Im zweiten Schritt wird eine mobile Anwendung entworfen, um kryptographische Schlüssel im persönlichen Kontakt auszutauschen. Die Anwendung setzt die zuvor entwickelte Architektur des Frameworks um und erweitert sie um anwendungsspezifische Funktionalität. Dazu gehört, dass ausgetauschte kryptographische Schlüssel für andere Anwendungen auf dem Gerät bereitgestellt werden können.

Zur Datenübertragung soll die Near Field Communication eingesetzt werden. Es handelt sich um eine Funktechnik zur kontaktlosen Datenübertragung über kurze Distanz. Damit eignet sich NFC, um spontan Daten auszutauschen. Ein weiterer Vorteil der NFC-Technik ist ihre einfache Handhabung. Um Daten zu übertragen, müssen sich zwei NFC-fähige Geräte in einer kurzen Distanz von bis zu 10 Zentimetern befinden [HB06]. Ansonsten ist keine Konfiguration notwendig. Eine manuelle Konfiguration wie beispielsweise bei Bluetooth entfällt [LR10]. Das Ergebnis der Arbeit soll eine Smartphone-Anwendung sein, mit der kryptographische Schlüssel im persönlichen Kontakt ausgetauscht werden können.

1.3 Aufbau der Arbeit

Der Aufbau der Arbeit wird nachfolgend beschrieben. Im zweiten Kapitel werden die relevanten Grundlagen erläutert. Im dritten Kapitel wird ein Ansatz zum Austausch kryptographischer Schlüssel entwickelt. Dieser Ansatz wird als Grundlage für den Entwurf des Frameworks und der Anwendung genutzt. Für die Anwendung wird im vierten Kapitel eine Sicherheitsanalyse vorgenommen. In der Analyse werden Bedrohungen identifiziert, für die anschließend Gegenmaßnahmen entwickelt werden. Im fünften Kapitel wird die Realisierung der Anwendung beschrieben. In Kapitel sechs wird diese hinsichtlich ihres Funktionsumfangs und ihrer Sicherheit evaluiert. Abschließend wird in Kapitel sieben erklärt,

wie die Anwendung in Zukunft weiterentwickelt werden kann und ein Fazit gezogen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen für die weitere Ausarbeitung erklärt. Zu Beginn wird das Konzept eines Frameworks erläutert. Anschließend wird die Near Field Communication vorgestellt und ihre Funktionsweise skizziert. Das Kapitel schließt mit einer Einführung in die Entwicklung für Android.

2.1 Framework-Grundlagen

Ein Framework ist ein Rahmenwerk für eine bestimmte Domäne. Es stellt eine unvollständige Software-Architektur bereit, die eingesetzt werden kann, um Anwendungen in dieser Domäne zu realisieren. In der Regel muss ein Framework für das jeweilige Szenario der zu entwickelnden Anwendung angepasst werden. Diese Definition und die nachfolgenden Erklärungen orientieren sich im Wesentlichen an dem „Handbuch der Software-Architektur“ von Ralf Reussner und Wilhelm Hasselbring [RH06].

2.1.1 Typische Merkmale eines Frameworks

Ausgehend von der obigen Definition ermöglicht ein Framework eine Softwarearchitektur, die für mehrere Anwendungen wiederverwendbar ist, wodurch der Entwicklungsaufwand reduziert werden kann [RH06]. Voraussetzung dafür ist, dass sich die vom Framework bereitgestellte Architektur flexibel für verschiedene Anwendungen anpassen lässt. Reussner und Hasselbring identifizieren drei

typische Merkmale, die ein solches Framework auszeichnen: Ein Framework invertiert den Kontrollfluss der Anwendung, es definiert ein Rahmenwerk für die Anwendung und stellt Variationspunkte bereit, um diese Architektur anzupassen [RH06]. Normalerweise steuert eine Anwendung ihren Kontrollfluss selbst, indem sie eine Abfolge von Operationen festlegt. Dabei werden unterschiedliche Klassenbibliotheken aufgerufen, um die Funktionalität der Anwendung umzusetzen [RH06]. Johnson und Foote erklären in „Designing Reusable Classes“ [JF88], dass eine Anwendung, die auf einem Framework basiert, anwendungsspezifische Operationen implementieren muss, um das Framework anzupassen. Diese Operationen werden häufig aus dem Framework selbst aufgerufen, wodurch es den Hauptkontrollfluss steuert. Durch die Umkehrung des Kontrollflusses kann das Framework als erweiterbares Architekturgerüst eingesetzt werden [JF88]. Dies entspricht dem zweiten Merkmal. Das dritte Merkmal ist, dass ein Framework Variationspunkte vorsehen muss, durch die es angepasst werden kann. Variationspunkte sind Stellen in einem Architekturgerüst, an denen Entscheidungen offen gelassen werden [CBB+10]. Während der Entwicklung einer Anwendung müssen diese Entscheidungen getroffen werden, wodurch das Framework angepasst wird.

2.1.2 Unterscheidung der Arten von Frameworks

Die gegebene Definition und die beschriebenen Merkmale beziehen sich auf Frameworks im Allgemeinen. Darüber hinaus unterscheiden Reussner und Hasselbring zwischen objektorientierten Frameworks, komponentenbasierten Frameworks und Mischformen aus diesen [RH06]. Die unterschiedlichen Arten von Frameworks können dadurch differenziert werden, wie sie sich für eine Anwendung anpassen lassen. Nachfolgend werden diese Frameworks kurz erklärt.

Johnson und Foote definieren ein objektorientiertes Framework als eine Sammlung von Klassen, die zusammen ein abstraktes Konzept realisieren, um eine bestimmte Art von Problem zu lösen [JF88]. Reussner und Hasselbring leiten aus mehreren Definitionen ab, dass ein objektorientiertes Framework Klassen und Objekte beschreibt und deren Verantwortlichkeiten festlegt [RH06]. Variationspunkte innerhalb eines objektorientierten Frameworks sind abstrakte Klassen. Um das Framework anzupassen, werden für diese konkrete Unterklassen erstellt, welche die anwendungsspezifischen Funktionalitäten umsetzen [RH06].

Bei einem komponentenbasiertes Framework liegt der Fokus auf den einzelnen Komponenten der Software und deren Schnittstellen. Eine Komponente ist ein modulares Stück Software, auf dessen Funktionalitäten über klar definierte Schnittstellen zugegriffen werden kann [OMG04]. In „Component Software - Beyond Object-Oriented Programming“ von Szyperski et. al wird ein komponentenbasiertes Framework als eine Menge von Schnittstellen und Regeln definiert, die festlegen wie die einzelnen Komponenten des Frameworks interagieren [Szy02]. Zu den Komponenten des Frameworks lassen sich konkrete Instanzen erzeugen, die in das Framework eingefügt werden können. Das Framework regelt dann die Interaktion der Instanzen. Ein komponentenbasiertes Framework lässt sich also durch erstellte Instanzen an die Anforderungen einer Anwendung anpassen. Daraus folgt, dass die Komponenten die Variationspunkte des Frameworks sind [RH06].

Zusätzlich zu den reinen objektorientierten und komponentenbasierten Frameworks gibt es hybride Frameworks, welche beide Arten kombinieren. Ein solches Framework kann durch Vererbungen und erstellte Instanzen an eine Anwendung angepasst werden [RH06].

2.2 NFC-Grundlagen

Die Near Field Communication (NFC) ist eine von NXP Semiconductors und Sony entwickelte Funktechnik zur kontaktlosen Datenübertragung über eine Distanz von bis zu zehn Zentimetern. NFC basiert auf der RFID-Technik und arbeitet im Frequenzbereich von 13,56 MHz. Daten können mit 106, 212 oder 424 kbit/s übertragen werden. Für weiterführende Informationen zu den technischen Grundlagen von RFID sei an dieser Stelle auf das „RFID Handbook“ von Finkenzeller verwiesen [Fin10].

Die Near Field Communication ist in zwei Protokollen definiert, die jeweils als ISO- und ECMA-Standard vorliegen. In dem Near Field Communication Protocol-1 (NFCIP-1) [ISO13, ECM04] wird unter anderem das verwendete Transportprotokoll, die Modulationsverfahren, die Kollisionsbehandlung und die Kodierung der Daten festgelegt. Das Near Field Communication Protocol-2 (NFCIP-2) [ISO12, ECM10] definiert einen Mechanismus für die Wahl zwischen verschiedenen Kommunikationsmodi. Unterstützt werden die Modi NFC (NFCIP-1), PCD (ISO/IEC 14443 [ISO08]) und VCD (ISO/IEC 15693 [ISO10]). Neben diesen bei-

den Standards gibt es weitere Spezifikationen, um den Umgang mit NFC zu vereinheitlichen. Dazu gehört die Spezifikation eines Datenaustauschformats.

2.2.1 Grundlegende Funktionsweise

NFC-Geräte übertragen Daten über ein elektromagnetisches Feld. Um eine Kommunikation aufzubauen, müssen zwei NFC-Geräte aneinandergehalten werden. Die Geräte treten dann in den Rollen *Initiator* und *Target* auf [ISO13]. Ein Initiator ist eine aktive Komponente, die ein elektromagnetisches Feld erzeugt, welches eine passive Komponente mit Energie versorgt. Begonnen wird eine NFC-Kommunikation von einem Gerät in der Rolle des Initiators. Es werden die drei Kommunikationsmodi Peer-to-Peer, Reader/Writer und Card-Emulation untersützt [LR10].

Im Peer-to-Peer-Modus können Daten zwischen zwei NFC-Geräten ausgetauscht werden [LR10]. Dabei wird zwischen dem aktiven und dem passiven Kommunikationsmodus unterschieden. Im aktiven Kommunikationsmodus wechseln die Geräte zwischen den Rollen Initiator und Target und erzeugen abwechselnd das elektromagnetische Feld zur Datenübertragung [ISO13]. Im Gegensatz dazu bleiben die Geräte im passiven Kommunikationsmodus in den Rollen Initiator und Target [ISO13]. Das elektromagnetische Feld wird für die gesamte Dauer der Kommunikation vom Initiator erzeugt.

Im Reader/Writer-Modus kann ein NFC-Gerät mit einem RFID-Transponder kommunizieren [LR10]. Das NFC-Gerät ist in diesem Fall der Initiator und der RFID-Transponder das Target. In diesem Modus, kann ein NFC-Gerät in bestehenden RFID-Systemen genutzt werden.

Im Card-Emulation-Modus emuliert das NFC-Gerät eine kontaktlose Chipkarte und kann so mit einem RFID-Lesegerät kommunizieren [LR10]. Das NFC-Gerät ist in diesem Modus das Target und übernimmt die Aufgabe einer kontaktlosen Chipkarte. Das RFID-Lesegerät agiert als Initiator. Die Kombination aus Reader/Writer-Modus und Card-Emulation-Modus ermöglicht eine bidirektionale Kommunikation wie der zuvor beschrieben Peer-to-Peer-Modus [LR10]. Die NFC-Geräte agieren abwechselnd als Lesegerät und kontaktlose Chipkarte. Aus den drei Kommunikationsmodi ergeben sich die in Tabelle 2.1 dargestellten Konfigurationen.

Gerät A	Gerät B	Beschreibung
Aktiv	Aktiv	Die beiden Geräte erzeugen abwechselnd das elektromagnetische Feld.
Aktiv	Passiv	Gerät A erzeugt das elektromagnetische Feld.
Passiv	Aktiv	Gerät B erzeugt das elektromagnetische Feld.

Tabelle 2.1: Konfiguration der Kommunikationsmodi, angelehnt an [HB06].

2.2.2 Verwendetes Datenformat

Das NFC Data Exchange Format (NDEF) ist ein Datenformat für die Kapselung von Anwendungsdaten in einer Nachricht und den Austausch mittels NFC [For06]. Eine solche NDEF-Message besteht aus einem oder mehreren NDEF-Records. NDEF-Records bestehen wiederum aus einem Header und einer Payload. Der Header beschreibt die Länge und den Typ der Payload. Optional kann auch ein Uniform Resource Identifier (URI) [1] angegeben werden, um die Payload zu identifizieren. Die Payload enthält die eigentlichen Daten. Abbildung 2.1 stellt den Aufbau einer solchen Nachricht dar.

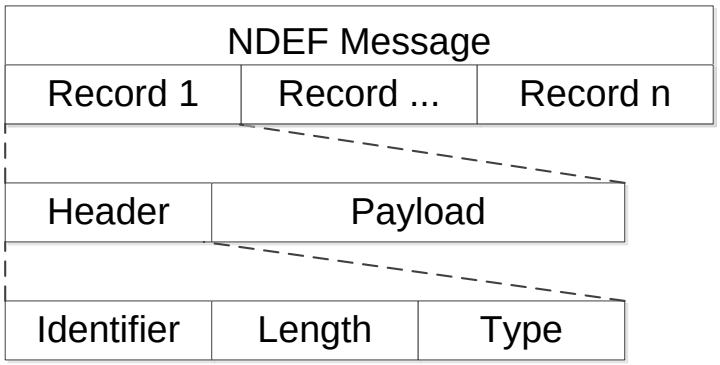


Abbildung 2.1: Aufbau einer NDEF-Message, nach [Rah11].

[1] Ein Uniform Resource Identifier ist eine Zeichenfolge um abstrakte oder physische Ressourcen zu identifizieren [HKRS08].

2.3 Android-Grundlagen

Dieser Abschnitt gibt einen Überblick über das Android-Betriebssystem. Die Informationen stammen aus „Android 2: Grundlagen und Programmierung“ [BP10] sowie dem Entwickler-Abschnitt der offiziellen Android-Webseite[2].

2.3.1 Zugrundeliegende Systemarchitektur

Nachfolgend wird die Systemarchitektur von Android kurz vorgestellt, Abbildung 2.2 zeigt diese im Überblick.

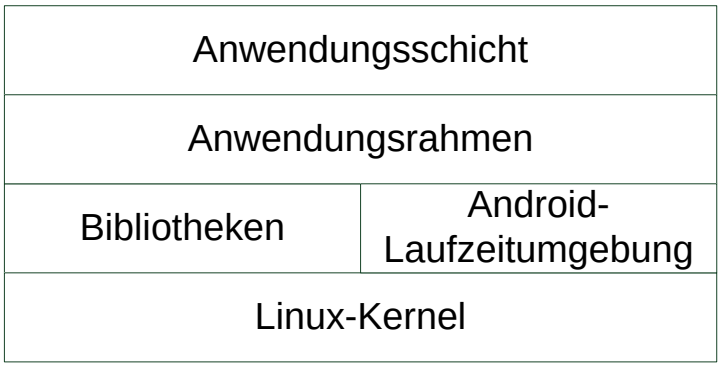


Abbildung 2.2: Aufbau der Systemarchitektur, nach [BP10].

Android basiert auf einem *Linux-Kernel*, der Treiber und verschiedene Optimierungen beinhaltet. Die Optimierungen beziehen sich vor allem auf die Speicherverwaltung und den Energieverbrauch. Die *Android-Laufzeitumgebung* besteht aus der *Dalvik Virtual Machine* (DVM) und verschiedenen Kern-Bibliotheken. Die Kern-Bibliotheken bieten den Großteil des Funktionsumfangs der Programmiersprache Java. Für jede Anwendung wird ein separater Prozess gestartet, in dem eine DVM läuft. Innerhalb der DVM wird die Anwendung ausgeführt. Dadurch sind Anwendungen voneinander unabhängig und teilen sich keinen gemeinsamen Speicher. Sie können nur auf ihre eigenen Daten zugreifen. Android stellt neben den Kern-Bibliotheken weitere *Bibliotheken* bereit. Unter anderem eine Biblio-

[2] Android Developers: http://developer.android.com/, Zuletzt abgerufen am 25.08.2013

thek für SQLite-Datenbanken. Der *Anwendungsrahmen* ermöglicht Entwicklern den Zugriff auf Hardwarekomponenten. Er stellt eine Abstraktionsstufe der eigentlichen Hardware dar, auf der Anwendungen aufsetzen können. Hervorzuheben sind der *Activity Manager*, der *Resource Manager* und die *Content Provider*. Der *Activity Manger* verwaltet den Lebenszyklus einer Anwendung. Der *Resource Manager* ermöglicht den Zugriff auf Ressourcen wie beispielsweise Strings, Grafiken oder Layouts. Über *Content Provider* können Anwendung eigene Daten bereitstellen oder auf die Daten anderer Anwendungen zugreifen. Die *Anwendungsschicht* beinhaltet Android-Anwendungen. Dazu gehören Standard-Anwendungen wie das Adressbuch oder der Mail-Client, eigens entwickelte Anwendungen oder Anwendungen von Drittanbietern.

2.3.2 Komponenten einer Anwendung

Android definiert vier unterschiedliche Arten von Komponenten, aus denen eine Anwendung aufgebaut werden kann.

2.3.2.1 Activity

Activities sind Komponenten mit einer grafischen Oberfläche, über die der Nutzer mit der Anwendung interagieren kann. Jede Activity entspricht dabei einem Bildschirm, dessen statische Oberfläche durch eine Layout-Datei beschrieben wird. In der Regel besteht eine Anwendung aus mehreren solcher Activities.

Eine Activity durchläuft einen Lebenszyklus. Es wird zwischen vier Hauptzuständen unterschieden. Wenn eine Activity im Vordergrund des Bildschirms vollständig sichtbar ist hat sie den Fokus. In diesem Zustand ist sie aktiv. Wenn eine Activity nur teilweise sichtbar ist, verliert sie den Fokus und wird pausiert. Das ist zum Beispiel der Fall, wenn eine neue Activity gestartet wird, die nicht den ganzen Bildschirm einnimmt. In diesem Zustand bleiben alle Daten der Activity erhalten. Eine pausierte Activity wird in seltenen Fällen vom System beendet, wenn nur sehr wenig Speicher verfügbar ist. Ist eine Activity nicht mehr sichtbar, wird sie gestoppt. Gestoppte Activities behalten ebenfalls ihre Daten, werden jedoch häufig vom System beendet, wenn Speicher benötigt wird. Wird eine Activity beendet, gehen alle ihre Daten verloren und müssen bei einem erneuten Start, falls nötig, wiederhergestellt werden. Abbildung 2.3 zeigt die einzelnen Phasen als Diagramm.

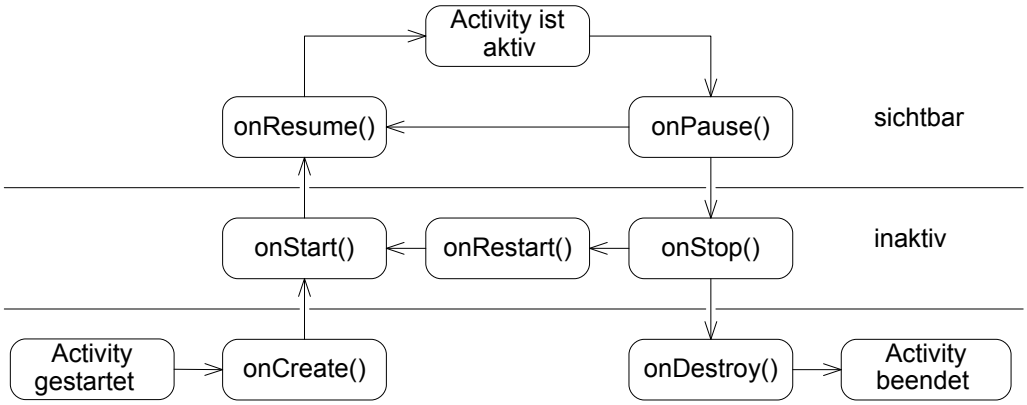


Abbildung 2.3: Lebenszyklus einer Activity, nach [BP10].

Die gesamte Lebenszeit einer Activity spielt sich zwischen den Methodenaufrufen `onCreate()` und `onDestroy()` ab. Eine Activity ist sichtbar, nachdem die Methode `onStart()` aufgerufen wurde. Sie bleibt solange sichtbar bis `onStop()` ausgeführt wird. Allerdings ist sie nur zwischen den Aufrufen von `onResume()` und `onPause()` vollständig sichtbar, ansonsten ist sie nur teilweise sichtbar.

Android verwaltet Activities in einem Stack. Eine neu gestartete Activity wird dem Stack hinzugefügt. Drückt der Nutzer den Zurück-Button, verlässt er die zu diesem Zeitpunkt angezeigte Activity. Die Activity wird beendet und vom Stack entfernt. Die Anordnung innerhalb des Stacks wird nicht verändert, es gilt dementsprechend das Prinzip Last In - First Out. Das beschriebene Verhalten kann anhand von Grafik 2.4 nachvollzogen werden. Im Ursprungszustand ist der Stack leer. Nacheinander werden Activity 1 und Activity 2 gestartet und dem Stack hinzugefügt. Dem Nutzer wird zu diesem Zeitpunkt Activity 2 angezeigt, welche als letztes hinzugefügt wurde. Um zu der vorherigen Activity 1 zurückzukehren, drückt der Nutzer den Zurück-Button. Activity 2 wird beendet und vom Stack entfernt.

2.3.2.2 Service

Ein Service ist eine Komponente, ohne grafische Oberfläche. Services werden von anderen Komponenten gestartet und in der Regel für Operationen eingesetzt, die für eine längere Zeit im Hintergrund laufen sollen.

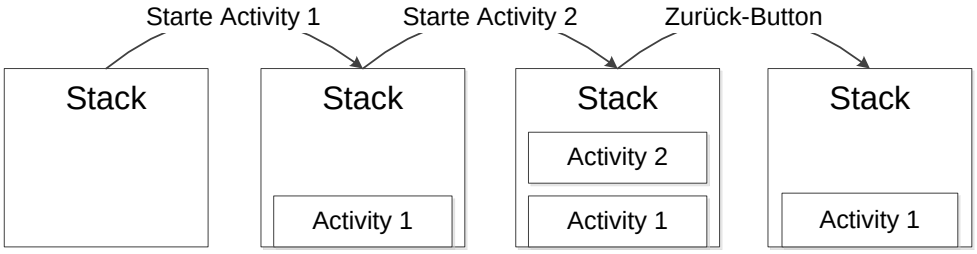


Abbildung 2.4: Arbeitsweise des Stacks, nach [Andb].

2.3.2.3 Content Provider

Content Provider erlauben es, Daten über die Grenzen der eigenen Anwendung hinaus zur Verfügung zu stellen. Wie in Abschnitt 2.3.1 beschrieben, wird jede Anwendung in einem separaten Prozess ausgeführt, weshalb andere Anwendungen normalerweise nicht auf ihre Daten zugreifen können. Ein Content Provider ermöglicht das Abfragen der Dateninhalte, liefert jedoch nicht die Daten selbst. Er stellt somit eine Abstraktion der eigentlichen Persistenzschicht dar. Android bietet beispielsweise einen Content Provider für gespeicherte Kontakte.

2.3.2.4 Broadcast Receiver

Ein Broadcast Receiver empfängt Systemnachrichten. Android verwendet solche Nachrichten beispielsweise um über einen niedrigen Akkustand zu informieren. Broadcast Receiver erlauben es einer Anwendung dynamisch auf die Änderung des Systemzustands zu reagieren.

2.3.3 Kommunikation mittels Intents

Die unterschiedlichen Komponenten und Anwendungen kommunizieren über Intents. Ein Intent besteht aus einer auszuführenden Aktion und einem URI, der die Daten festlegt auf denen gearbeitet wird. Es wird zwischen expliziten und impliziten Intents unterschieden. Explizite Intents legen eine Komponente fest, die gestartet werden soll. Implizite Intents legen hingegen keine Komponente fest. Sie müssen stattdessen Informationen enthalten, anhand derer das System entscheiden kann, welche Komponente gestartet werden soll. Listing 2.1 und 2.2

zeigen Beispiele für solche Intents. Listing 2.1 definiert einen Intent um Activity A zu starten. Der erste Parameter definiert den Absender und der zweite den Empfänger. Listing 2.2 zeigt einen impliziten Intent um einen Anruf abzusetzen. Der erste Parameter legt fest, dass es sich um die Aktion *Anrufen* handelt. Der zweite Parameter ist die anzurufende Nummer. Damit eine Activity auf einen Intent reagieren kann muss im zugehörigen Manifest ein Intent-Filter definiert werden. Ein solcher Filter legt alle Intents fest, auf die eine Activity reagieren soll.

```
1 Intent i = new Intent(this, ActivityA.class);
```

Listing 2.1: Expliziter Intent

```
1 Intent i = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:(+49)978654321"));
```

Listing 2.2: Impliziter Intent

2.3.4 Aufbau des Manifests

Jede Anwendung muss ein Manifest besitzen, in dem grundsätzliche Informationen über die Anwendung enthalten sind. Dazu gehört, dass alle Activities aufgeführt werden, aus denen die Anwendung besteht. Zu den Activities müssen Intent-Filter angegeben werden, falls sie auf bestimmte Intents reagieren sollen. Listing 2.3 zeigt beispielhaft den Aufbau eines Manifests. Als Erstes wird mit den Elementen `uses-sdk` und `uses-feature` festgelegt, für welche Android Version die Anwendung geeignet ist und welche Hardwareanforderungen gelten. Mit dem `uses-permission`-Element werden Berechtigungen vom Betriebssystem eingefordert. In diesem Beispiel benötigt die Anwendung Zugriff auf die NFC-Hardware des Geräts. Während der Installation einer Anwendung wird dem Nutzer mitgeteilt, welche Berechtigungen angefordert werden. Ist er nicht bereit diese Berechtigungen zu erteilen, wird die Installation abgebrochen. Innerhalb des `application`-Elements werden die einzelnen Komponenten der Anwendung festgelegt. Für jede Komponente kann definiert werden, auf welche Intents sie reagiert.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example" >
 <uses-sdk
   android:minSdkVersion="14"
   android:targetSdkVersion="17" />

 <uses-feature android:name="android.hardware.nfc" />
 <uses-permission android:name="android.permission.NFC" />

 <application
        android:allowBackup="true"
        android:label="@string/app_name" >
        <activity
            android:name="keyexchange.cryptography.SecureChannelStep1"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
 </application>
</manifest>
```

Listing 2.3: Aufbau eines Manifests

2.3.5 NFC-Unterstützung in Android

Seit Version 2.2 [Gin] unterstützt Android NFC. Mit Version 4.0 [ICS] wurde die Beam-API eingeführt, um Daten möglichst einfach zwischen zwei Geräten zu übertragen. Werden zwei NFC-fähige Android-Geräte aneinander gehalten, öffnet sich ein Dialog, mit dem die Nutzer Daten an das jeweils andere Gerät senden können. Beide Geräte müssen dafür Beam aktiviert haben. Android nutzt Beam beispielsweise, um Kontakte zwischen zwei Geräten auszutauschen.

Beam hat gegenüber Bluetooth den Vorteil, dass aufgrund der NFC-Technik keine manuelle Konfiguration notwendig ist. Obwohl Beam laut Android einen „einfachen peer-to-peer Datenaustausch“ [Anda] erlaubt, ist bisher keine bidirektionale Kommunikation zwischen zwei Geräten möglich. Wenn eine Anwendung Beam unterstützt muss sie die zu übertragenden Daten in einer NDEF-Message kapseln. Um die Daten an ein anders Gerät zu senden, müssen diese beiden zusammengehalten werden. Es erscheint ein Dialog mit dem Text „Zum Beamen berühren“. Sobald der Nutzer den Vorgang bestätigt hat, werden die Daten an

das zweite Gerät gesendet. Für jede Datenübertragung muss dieser Vorgang wiederholt werden. Es ist nicht möglich im gleichen Vorgang Daten zu senden und anschließend zu empfangen. Dadurch, dass jede Übertragung vom Nutzer bestätigt werden muss, wird ein möglicher Missbrauch erschwert. Gleichzeitig ist es jedoch schwierig komplexere Anwendungen über Beam zu realisieren.

Kapitel 3

Konzeption der Anwendung

Dieses Kapitel beschreibt den Entwurf eines allgemeinen Konzepts, um kryptographische Schlüssel zwischen mobilen Geräten auszutauschen. Das Konzept wird im weiteren Verlauf des Kapitels präzisiert und als Grundlage verwendet, um ein Framework und eine darauf basierende Anwendung zu entwerfen.

3.1 Konzept zum sicheren Schlüsselaustausch

In diesem Abschnitt wird ein Konzept entwickelt, um kryptographische Schlüssel zwischen mobilen Geräten auszutauschen. Das Konzept soll folgendes Szenario abdecken: Zwei Personen, die sich treffen, möchten in Zukunft sicher kommunizieren. Dafür müssen sie kryptographische Schlüssel austauschen. Wichtig ist, dass die ausgetauschten kryptographischen Schlüssel nur den beiden Personen bekannt sind und nicht von einer dritten Person abgefangen werden können. Darüber hinaus muss gewährleistet sein, dass ein empfangener kryptographischer Schlüssel tatsächlich vom Sender stammt und nicht manipuliert wurde. Das Konzept fokussiert also einen unkomplizierten und gleichzeitig sicheren Schlüsselaustausch.

Aus dem beschriebenen Szenario lassen sich Aspekte ableiten, die beachtet werden müssen. Das Konzept muss zwischen drei unterschiedlichen Rollen und dem Gegenstand des Schlüsselaustauschs unterscheiden: Ein Sender ist eine Person, die einer anderen einen kryptographischen Schlüssel mitteilt. Die Person, welcher der kryptographische Schlüssel mitgeteilt wird, ist der Empfänger. Ein Angreifer verfolgt das Ziel kryptographische Schlüssel, die ein Sender einem

Empfänger mitteilt, abzufangen und zu verändern. Der Gegenstand des Schlüsselaustauschs ist der kryptographische Schlüssel, der dem Empfänger mitgeteilt werden soll. Um die Schlüssel zu übertragen, müssen Sender und Empfänger über einen Kanal verbunden sein. Darüber hinaus benötigen sowohl der Sender als auch der Empfänger eine Möglichkeit, um kryptographische Schlüssel zu verwalten. Die letzten beiden Anforderungen müssen durch eine Anwendung realisiert werden. Abbildung 3.1 zeigt die Zusammenhänge zwischen Sender, Empfänger, Angreifer und dem Übertragungskanal.

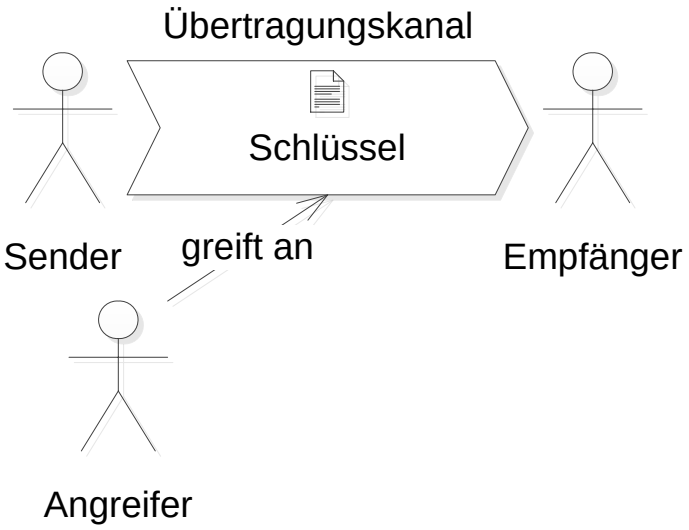


Abbildung 3.1: Zusammenhänge der Personen

Für eine Umsetzung des Konzepts muss eine Anwendung entwickelt werden, die es erlaubt kryptographische Schlüssel zu verwalten, zu versenden und zu empfangen. Um Daten auszutauschen, muss die Anwendung einen Übertragungskanal zwischen Sender und Empfänger etablieren können. Durch kryptographische Verfahren muss das Abfangen und Manipulieren der kryptographischen Schlüsseln ausgeschlossen werden. Gemäß dem Konzept soll der Schlüsselaustausch wie folgt ablaufen: Der Sender wählt den zu übertragenden kryptographischen Schlüssel in seiner Anwendung. Anschließend etabliert die Anwendung einen Übertragungskanal zwischen Sender und Empfänger, über den der

kryptographische Schlüssel gesendet wird. Nachdem die Anwendung des Empfängers den kryptographischen Schlüssel erhalten hat, kann ihn dieser speichern. Die Absicherung des Übertragungskanals und der Anwendung wird gesondert in Kapitel 4 behandelt.

Das in diesem Konzept behandelte Problem des sicheren Schlüsselaustauschs, lässt sich auf jegliche Form des Datenaustauschs übertragen. Um das Konzept zu realisieren, muss zuerst die Frage beantwortet werden, wie ein sicherer Datenaustausch umgesetzt werden kann. Eine Lösung für dieses Problem wird dann im nächsten Schritt als Grundlage genutzt, um kryptographische Schlüssel sicher zu übertragen. Im weiteren Entwurf wird deshalb zwischen dem sicheren Datenaustausch im Allgemeinen und einer Umsetzung des Konzepts unterschieden. In Abschnitt 3.2 wird ein Framework entworfen, das einen sicheren Datenaustausch zwischen zwei Parteien ermöglicht. Aufbauend auf diesem Framework wird in Abschnitt 3.3 eine Anwendung entworfen, um den im Konzept beschriebenen Ansatz umzusetzen. Um kryptographische Schlüssel spontan zu übertragen, soll die Near Field Communication (NFC) eingesetzt werden. NFC erlaubt es, Daten ohne vorherige Konfiguration auszutauschen und eignet sich daher für diesen Zweck. Das Ergebnis des Entwurfs ist eine Architektur für eine Umsetzung des entwickelten Konzepts und eine Architektur für einen sicheren Datenaustausch in anderen Anwendungen. Da das Problem des sicheren Datenaustauschs über den Rahmen dieser Arbeit hinaus relevant ist, wird somit ein zusätzlicher Mehrwert geschaffen.

3.2 Entwurf des Frameworks

Wie bereits in Abschnitt 3.1 beschrieben, wird die Anwendung in zwei Phasen entworfen: Zuerst muss das Problem des sicheren Datenaustauschs durch das Framework gelöst werden. Darauf aufbauend wird der sichere Datenaustausch in der Anwendung realisiert. Das Framework soll ein kompaktes Grundgerüst festlegen, das für den sicheren Datenaustausch genutzt werden kann. Hierfür wird ein komponentenbasierter Ansatz gewählt. Das Framework definiert Komponenten und deren Abhängigkeiten. Die Komponenten bieten ihre Funktionalität über festgelegte Schnittstellen an. In einer Anwendung, die das Framework umsetzt, müssen die Komponenten als konkrete Instanzen realisiert werden. Dabei ist entscheidend, dass die Instanzen die vom Framework vorgegebenen Schnittstellen

anbieten. Wie eine solche Instanz intern aufgebaut ist und wie sie arbeitet ist nicht festgelegt.

Aus dem in Abschnitt 3.1 vorgestellten Konzept ergeben sich vier wesentliche Aspekte, die zu berücksichtigen sind. Das Framework muss Methoden bieten, um Daten verwalten, speichern und versenden zu können. Um eine sichere Datenübertragung zu ermöglichen, muss das Framework entsprechende kryptographische Methoden bereitstellen. Das Framework realisiert diese Funktionalitäten durch die in Abbildung 3.2 dargestellten Komponenten. Die `Management`-Komponente wird zur Verwaltung eingesetzt. Die `Storage`-Komponente stellt den Persistenzmechanismus des Frameworks dar. Die `Exchange`-Komponente realisiert in Kombination mit der `Cryptography`-Komponente die sichere Datenübertragung. Nachfolgend werden die Komponenten im Einzelnen vorgestellt. Die genauen Spezifikationen der Komponenten und deren Schnittstellen befinden sich in Anhang A. Anschließend wird erklärt wie die Komponenten für den Datenaustausch eingesetzt werden. Die Beschreibung behandelt damit, die in Abschnitt 2.1.2 identifizierten Merkmale eines komponentenbasierten Frameworks. Nach Szyperski sind dies die Komponenten mit ihren Schnittstellen und die Interaktion der Komponenten untereinander [Szy02].

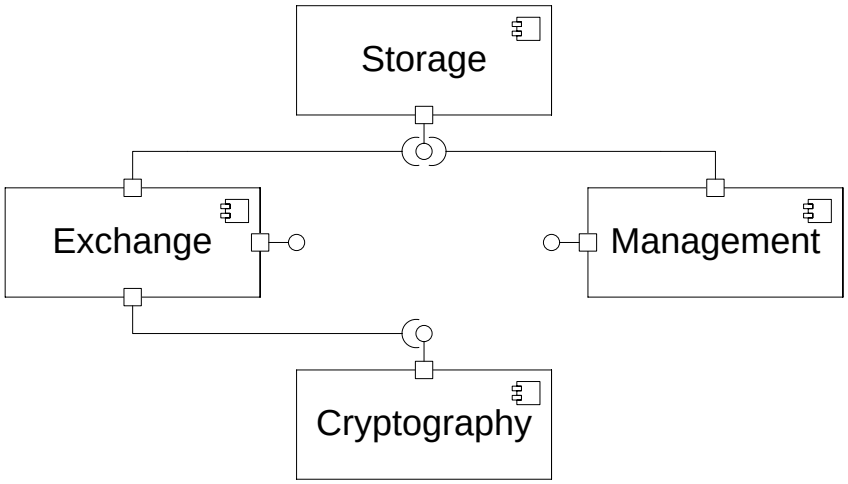


Abbildung 3.2: Komponenten des Frameworks

3.2.1 Management-Komponente

Die `Management`-Komponente ermöglicht es Daten zu erzeugen und zu speichern. Bereits vorhandene Daten können bearbeitet und gelöscht werden. Die Komponente nutzt die Schnittstelle der `Storage`-Komponente, um neu erstellte Daten zu speichern oder bereits vorhandene Daten abzufragen.

3.2.2 Storage-Komponente

Die `Storage`-Komponente erlaubt es Daten persistent zu speichern, sie abzufragen oder zu löschen. Eine konkrete Instanz dieser Komponente muss auf einem bereits vorhandenem Persistenzmechanismus, wie beispielsweise einer Datenbank, aufbauen. Die Schnittstellen der Komponente kapseln den Programmcode, der benötigt wird, um mit dem jeweiligen Persistenzmechanismus zu arbeiten.

3.2.3 Exchange-Komponente

Die `Exchange`-Komponente regelt den Datenaustausch zwischen zwei Parteien. Ihre Aufgabe ist es, einen Kanal zwischen den zwei Parteien aufzubauen, auf dem Daten übertragen werden können. Um gespeicherte Daten zu senden und empfangene Daten zu speichern, werden die Schnittstellen der `Storage`-Komponente genutzt. Es wird zwischen Methoden unterschieden, die benötigt werden, um einen Kanal auf- beziehungsweise abzubauen und einer Methode, die Daten entgegen nimmt um diese zu übertragen. Um zu verhindern, dass die übertragenen Daten einem Angreifer bekannt werden, wird auf Funktionalitäten der `Cryptography`-Komponente zurückgegriffen.

3.2.4 Cryptography-Komponente

Die `Cryptography`-Komponente stellt Schnittstellen für die Absicherung des Übertragungskanals bereit. Wie die Komponente umgesetzt wird, hängt maßgeblich von dem verwendeten Übertragungskanal ab. Gegebenenfalls werden weitere Methoden benötigt, um Daten auf dem abgesicherten Kanal übertragen zu können.

3.2.5 Ablauf des Datenaustauschs

Der Datenaustausch, wird in fünf nacheinander ablaufende Phasen unterteilt. Es wird zwischen den Phasen *Vorbereitung*, *Verbindungsaufbau*, *Übertragung*, *Verbindungsabbau* und *Nachbereitung* unterschieden. Die klare Unterscheidung ist, wichtig da es um ein allgemeines Konzept zum Datenaustausch geht. Je nach konkreter Realisierung, können einzelne Phase unterschiedlich umgesetzt werden. Die anderen Phasen sind davon nicht betroffen. Abbildung 3.3 stellt die Phasen als Aktivitätsdiagramm dar. Weitere Abbildungen, welche die Abläufe innerhalb der einzelnen Phasen visualisieren, befinden sich in Anhang B.

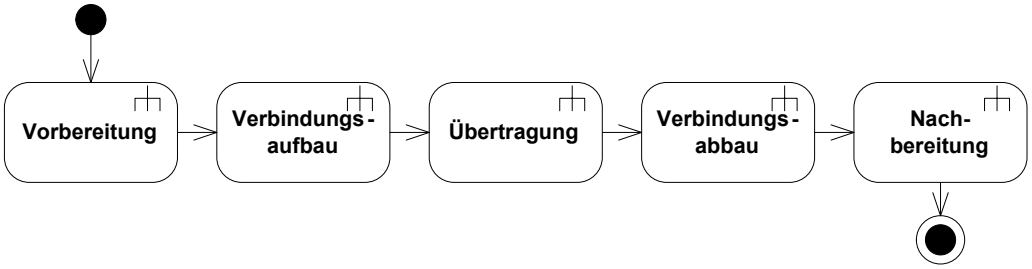


Abbildung 3.3: Ablauf des Datenaustauschs

In der Vorbereitungsphase wählt der Sender Daten aus, die dem Empfänger mitgeteilt werden sollen. Der Zugriff auf gespeicherte Daten erfolgt über die `Storage`-Komponente. Die Daten werden dann in ein geeignetes Format transformiert, um sie zu übertragen. Anschließend findet der Verbindungsaufbau statt. Über die `Exchange`-Komponente führen Sender und Empfänger mehrere Schritte durch um eine Verbindung aufzubauen. Welche Schritte im Einzelnen notwendig sind, hängt maßgeblich von dem verwendeten Übertragungskanal ab. Nachdem ein Kanal etabliert wurde, muss dieser gegen Angriffe abgesichert werden. Hierfür wird die `Cryptography`-Komponente eingesetzt. Die Entscheidung wie der Kanal gesichert wird, muss während dem Entwurf einer Anwendung getroffen werden. Nachdem der Verbindungsaufbau durchgeführt wurde, besteht ein abgesicherter Kanal, über den die `Exchange`-Komponente des Senders die ausgewählten Daten an den Empfänger schickt. Der Empfänger erhält die Daten und sendet als Antwort eine Bestätigung. Nachdem die Bestätigung beim Sender eingegangen ist, kann die Verbindung abgebaut werden. In der Nachbereitungsphase transformiert der Empfänger die erhaltenen Daten und

stellt dadurch die ursprünglichen Daten des Senders wieder her. Abschließend werden die Daten über die Schnittstelle der `Storage`-Komponente auf Seiten des Empfängers gespeichert.

3.3 Entwurf der Anwendung

Die mobile Anwendung soll es dem Nutzer ermöglichen kryptographische Schlüssel zu erstellen, zu verwalten und mit anderen Personen auszutauschen. Die kryptographischen Schlüssel müssen um Zusatzinformationen ergänzt werden, damit sie einer Person zugeordnet werden können. Ohne solche Informationen ist nicht klar, von wem ein empfangener kryptographischer Schlüssel stammt. Eine Zusatzinformation kann zum Beispiel ein Name oder ein Pseudonym sein. Der kryptographische Schlüssel und dessen Zusatzinformationen werden in einem Objekt zusammengefasst. Im Folgenden werden solche Objekte als Schlüsseleinträge bezeichnet. Das Erzeugen und Verwalten von Schlüsseleinträgen wird durch eine Instanz der `Management`-Komponente realisiert. Um die Schlüsseleinträge persistent zu speichern wird eine Instanz der `Storage`-Komponente genutzt. Die sichere Datenübertragung wird durch Instanzen der `Exchange`- und `Cryptography`-Komponente umgesetzt. Alle Grundfunktionen können somit über konkrete Instanzen der Framework-Komponenten realisiert werden. Darüber hinaus werden anwendungsspezifische Funktionalitäten benötigt, die im Folgenden beschrieben werden.

Um gespeicherte kryptographische Schlüssel auf dem Gerät nutzen zu können, müssen sie für andere installierte Anwendungen bereitgestellt werden. Beispielsweise könnten sie vom Mail-Client genutzt werden um E-Mails zu verschlüsseln. Es sollte daher eine Form des Zugriffsmanagements implementiert werden. Um die kryptographischen Schlüssel nicht nur auf dem Smartphone nutzen zu können, sondern auch auf anderen Geräten, wie beispielsweise einem Laptop, muss ein Export möglich sein. Gleichermaßen muss es möglich sein, kryptographische Schlüssel, die auf anderen Geräten gespeichert sind, in die Anwendung zu importieren. Für den Export und Import müssen die Schlüsseleinträge in ein anderes Format, wie beispielsweise XML, transformiert werden. Der Zugriff auf die Anwendung sollte durch ein selbst gewähltes Passwort beschränkt werden. Außerdem sollte die Anwendung Mehrsprachigkeit unterstützen, damit sie von einem möglichst großen Nutzerkreis verwendet werden kann.

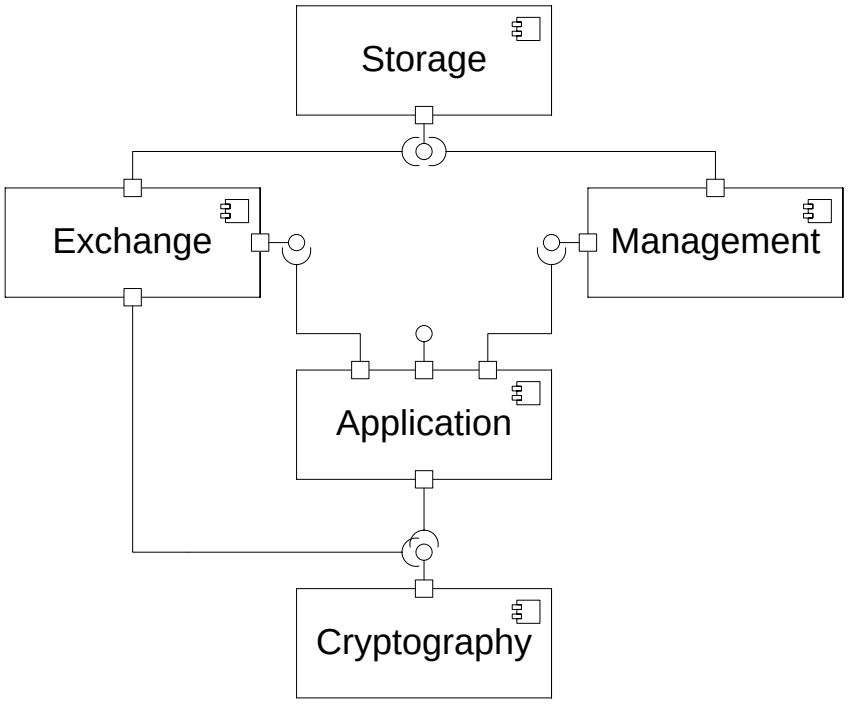


Abbildung 3.4: Komponenten der Anwendung

Die beschriebenen anwendungsspezifischen Funktionalitäten werden in der `Application`-Komponente zusammengefasst. Abbildung 3.4 visualisiert die einzelnen Komponenten der Anwendung und ihre Zusammenhänge. Innerhalb der `Application`-Komponente sind weitere Komponenten für das Bereitstellen von kryptographischen Schlüsseln, das Exportieren beziehungsweise Importieren und die Konfiguration der Anwendung enthalten. Die in der Grafik dargestellten Komponenten bilden somit die Grundstruktur der Anwendung. Der im Framework beschriebene Datenaustausch legt den Kontrollfluss fest.

Kapitel 4

Sicherheitsanalyse für die Anwendung

Im Folgenden wird eine Sicherheitsanalyse für die Anwendung durchgeführt. Das Vorgehen orientiert sich an einem von Grim et al. entwickelten Referenzmodell zur IT-Sicherheitsanalyse [GBK11]. Zuerst werden die zu schützenden Werte und Güter identifiziert. Anschließend sind Sicherheitsanforderungen für diese Werte und Güter zu definieren. Im nächsten Schritt werden Bedrohungen betrachtet, die sich gegen die Werte und Güter richten und damit die gestellten Sicherheitsanforderungen verletzen. Aus diesen Bedrohungen werden Angriffe abgeleitet. Abschließend müssen Sicherheitsmaßnahmen festgelegt werden, welche die Sicherheitsanforderungen realisieren und die identifizierten Angriffe verhindern. Die Zusammenhänge der einzelnen Elemente sind in Abbildung 4.1 visualisiert. Das Ziel der Analyse ist eine Übersicht über notwendige Sicherheitsmaßnahmen, um den identifizierten Angriffen zu begegnen.

4.1 Zu schützende Werte und Güter

Die Anwendung soll eingesetzt werden, um kryptographische Schlüssel zwischen zwei Personen auszutauschen. Die zu schützenden Güter sind demnach die auszutauschenden kryptographischen Schlüssel. Die kryptographischen Schlüssel können genutzt werden, um die Inhalte einer Kommunikation der beiden Personen zu verschlüsseln. Die verschlüsselten Informationen, sind ein ebenfalls zu

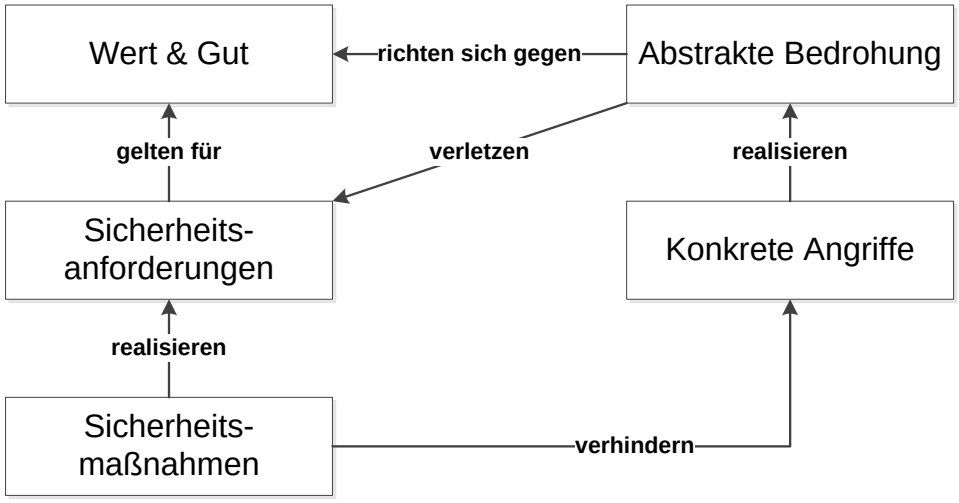


Abbildung 4.1: Übersicht über das Referenzmodell, nach [GBK11].

schützendes Gut. Jede Bedrohung, die sich gegen die kryptographischen Schlüssel richtet, gilt gleichermaßen für die verschlüsselten Informationen. Aus diesem Grund werden in der weiteren Analyse nur die kryptographischen Schlüssel betrachtet.

4.2 Gewünschte Sicherheitsanforderungen

Nachfolgend werden die Sicherheitsanforderungen angeführt, die für die kryptographischen Schlüssel gelten sollen.

4.2.1 Vertraulichkeit

Bei den kryptographischen Schlüsseln handelt es sich um sensible Daten, die vor unautorisiertem Zugriff geschützt werden müssen. Demnach muss Vertraulichkeit gewährleistet sein. Dies ist der Fall, wenn keine unerlaubte Informationsgewinnung möglich ist [Eck06]. Die Sicherheitsanforderung gilt sowohl für den Zugriff, als auch für den Austausch von kryptografischen Schlüsseln.

4.2.2 Authentizität

Nach Eckert [Eck06] ist Authentizität die Echtheit und Glaubwürdigkeit eines Objekts, in diesem Fall der kryptographischen Schlüssel. Authentizität ist gegeben, wenn sowohl die Integrität als auch die Originalität sichergestellt sind. Wird eine dieser Anforderungen verletzt, ist die Authentizität nicht länger gewährleistet. In Abschnitt 4.2.3 und 4.2.4 werden die beiden Teilaspekte Integrität und Originalität behandelt.

4.2.3 Integrität

Die Integrität der kryptographischen Schlüssel ist gewährleistet, wenn es für einen Angreifer nicht möglich ist diese unbemerkt zu manipulieren [Eck06]. Die Anforderung ist im Hinblick auf den Datenaustausch zu beachten, der bei der Near Field Communication unverschlüsselt erfolgt.

4.2.4 Originalität

Amann und Atzmüller [AA92] definieren Originalität als die Übereinstimmung der angenommenen Herkunft mit der tatsächlichen Herkunft. Es muss sichergestellt sein, dass ein erhaltener kryptographischer Schlüssel tatsächlich von der Person stammt, mit der zuvor ein Austausch durchgeführt wurde.

4.2.5 Zusammenfassung der Sicherheitsanforderungen

Zusammenfassend sind folgende Sicherheitsanforderungen festzuhalten:

SA1 Vertraulichkeit der Datenübertragung

SA2 Vertraulichkeit der gespeicherten kryptographischen Schlüssel

SA3 Integrität der übertragenen kryptographischen Schlüssel

SA4 Originalität der kryptographischen Schlüssel

4.3 Relevante Bedrohungen

Dieser Abschnitt beschreibt Bedrohungen, die sich gegen die Werte und Güter richten und die Sicherheitsanforderungen verletzen. Die Vertraulichkeit wird verletzt, wenn die ausgetauschten kryptographischen Schlüssel einem Angreifer bekannt werden. Es sind dabei zwei Fälle zu betrachten: Ein Angreifer kann die kryptographischen Schlüssel abfangen, indem er die Datenübertragung abhört oder er greift direkt auf die in der Anwendung gespeicherten Schlüssel zu. Bezogen auf die Integrität muss zwischen dem Zerstören der Daten und dem gezielten Manipulieren unterschieden werden. Ein Angreifer kann die Datenübertragung stören, wodurch die ausgetauschten kryptographischen Schlüssel unbrauchbar werden. Wenn es einem Angreifer gelingt die Übertragung abzuhören und die Daten zu verändern, kann er die kryptographischen Schlüssel anpassen, ohne dass diese unbrauchbar werden. Während zerstörte oder verfälschte Daten leicht erkannt werden können, da sie nicht nutzbar sind, ist dies bei einer gezielten Manipulation nicht ohne weiteres möglich. Die Originalität der Daten wird gefährdet, wenn es einem Angreifer gelingt, eigene Daten in die Datenübertragung einzubringen. Zusammenfassend sind folgende Bedrohungen möglich:

B1 Bekanntwerden der kryptographischen Schlüssel (Verletzt SA1)

B2 Unautorisierter Zugriff auf kryptographische Schlüssel (Verletzt SA2)

B3 Zerstören der kryptographischen Schlüssel (Verletzt SA3)

B4 Manipulieren der kryptographischen Schlüssel (Verletzt SA3 und SA4)

4.4 Identifizierte Angriffe

Aus den ermittelten Bedrohungen lassen sich verschiedene Angriffe ableiten. Dabei stehen mobile Geräte und der Datenaustausch über die Near Field Communication im Vordergrund. Mögliche Angriffe auf NFC wurden bereits von Haselsteiner und Breitfuß [HB06] beschrieben und werden an dieser Stelle aufgegriffen.

4.4.1 Übertragung abhören

Die Daten werden über NFC übertragen, sind daher also unverschlüsselt und können mit einer entsprechenden Antenne abgehört werden. Verfügt ein Angrei-

fer über das Wissen, wie die ausgetauschten Daten kodiert sind, kann er diese wiederherstellen. Es muss davon ausgegangen werden, dass ein Angreifer sowohl die technischen Geräte als auch das Wissen für einen solchen Angriff besitzt.

Um Daten zu übertragen, müssen sich NFC-fähige Geräte in einer Distanz von bis zu 10 Zentimetern befinden. Allerdings können die Signale auch über eine größere Distanz empfangen werden. Haselsteiner und Breitfuß identifizieren verschiedene Faktoren, welche die Distanz beeinflussen [HB06]. Dazu gehören unter anderem die Charakteristika des elektromagnetischen Felds des Senders, die technische Ausstattung des Angreifers und die Umgebung, in der die Datenübertragung stattfindet. Aufgrund der unterschiedlichen Faktoren ist es nicht möglich eine exakte Distanz für einen Angriff festzulegen. Haselsteiner und Breitfuß merken an, dass die Distanz maßgeblich davon beeinflusst wird in welchem Modus sich das sendende Gerät befindet. Sie geben als grobe Richtwerte 10 Meter für Geräte im aktiven Modus und 1 Meter für Geräte im passiven Modus an [HB06]. Der Angriff ist als realistisch anzusehen.

4.4.2 Übertragung stören

Ein Angreifer kann den Übertragungskanal stören. Nach Haselsteiner und Breitfuß ist dies möglich, indem der Angreifer selbst Daten sendet [HB06]. Allerdings muss er dafür Kenntnisse über die verwendete Modulation und Kodierung haben, um zum korrekten Zeitpunkt zu senden. Die kryptographischen Schlüssel werden in diesem Fall nicht gezielt manipuliert, sondern unbrauchbar.

4.4.3 Übertragung manipulieren

Es gibt drei Angriffe, die darauf abzielen die Übertragung zu manipulieren: die gezielte Manipulation der Kommunikationsinhalte, das Einfügen eigener Daten und der Man-in-the-Middle-Angriff.

4.4.3.1 Gezielte Manipulation

Damit die Daten unbemerkt verändert werden können, müssen sie nach der Manipulation weiterhin gültig sein. Damit ist dieser Angriff schwieriger als die Daten einfach zu zerstören. Haselsteiner und Breitfuß kommen zu dem Schluss, dass

die Durchführbarkeit eines solchen Angriffs bezogen auf NFC von der verwendeten Modulation abhängt [HB06].

4.4.3.2 Daten in die Übertragung einfügen

Anstatt die Daten gezielt zu manipulieren, kann ein Angreifer eigene Daten in die Übertragung einfügen. Bezogen auf NFC gilt, dass sich zeitgleich übertragene Daten überlagern und dadurch unbrauchbar werden. Der Angriff muss also zu Beginn der Datenübertragung stattfinden, bevor der eigentliche Sender Daten versendet.

4.4.3.3 Man-in-the-Middle

Bei einem Man-in-the-Middle-Angriff agiert der Angreifer als Mittelsmann zwischen zwei kommunizierenden Parteien. Dabei wird vorausgesetzt, dass der Angreifer die Kommunikation abhören kann. Abbildung 4.2 visualisiert beispielhaft den Ablauf eines solchen Angriffs auf die Kommunikation zwischen zwei Personen. Alice und Bob sind die Kommunikationspartner und Eve ist der Angreifer. Damit Eve als Mittelsmann agieren kann, baut sie eine Verbindung zu Alice und zu Bob auf. Alice und Bob sind sich nicht bewusst, dass sie nicht direkt miteinander kommunizieren. Eine Nachricht, welche Alice an Bob sendet, wird zuerst von Eve empfangen (1). Eve kann die Nachricht nun manipulieren (2) und anschließend an Bob senden (3).

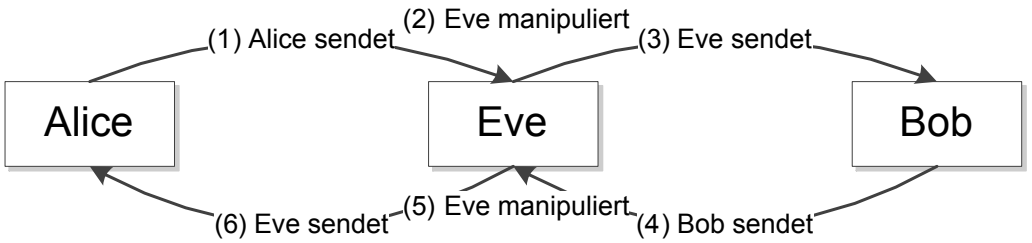


Abbildung 4.2: Ablauf von Man-in-the-Middle

Angewendet auf die Near Field Communication sind zwei Fälle zu unterscheiden. In der ersten Variante ist einer der Kommunikationspartner dauerhaft im aktiven Modus und erzeugt das elektromagnetische Feld für beide Kommunikationspartner. In der zweiten Variante erzeugen die Kommunikationspartner abwechselnd das elektromagnetische Feld.

Zuerst wird der Fall betrachtet, dass Alice das elektromagnetische Feld für die gesamte Kommunikationsdauer erzeugt. Dieser Ablauf ist in Abbildung 4.3 dargestellt. Nachdem Alice das elektromagnetische Feld erzeugt hat, sendet sie Daten an Bob (1). Eve muss die gesendeten Daten abfangen und gleichzeitig verhindern, dass Bob die Daten empfängt (1). Wie in Abschnitt 4.4.2 beschrieben, ist es möglich die Übertragung zu stören. Allerdings kann eine solche Störung von dem sendenden Gerät erkannt werden. Ausgehend davon, dass die Störung unbemerkt bleibt, manipuliert Eve die Daten (2). Um die manipulierten Daten an Bob zu senden, muss Eve ein eigenes elektromagnetisches Feld erzeugen. Das elektromagnetische Feld von Alice ist allerdings weiterhin aktiv und überlagert sich mit dem neu Erzeugten (2). Haselsteiner und Breitfuß weisen darauf hin, dass es praktisch unmöglich ist, die beiden elektromagnetischen Felder anzugleichen [HB06]. Dementsprechend können die Daten nicht ohne Störung übertragen werden, weshalb Bob verfälschte Daten erhält, mit denen er nicht weiterarbeiten kann.

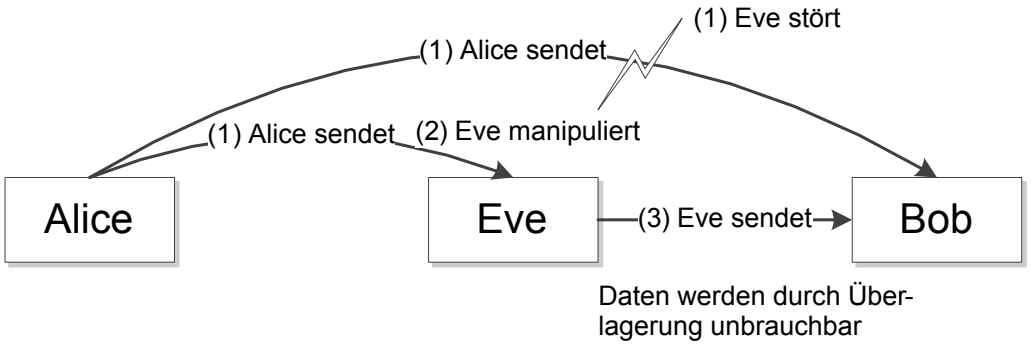


Abbildung 4.3: Ablauf von Man-in-the-Middle, wenn das elektromagnetische Feld dauerhaft von Alice erzeugt wird

Wird das elektromagnetische Feld hingegen im Wechsel erzeugt, ist der Angriff theoretisch durchführbar. Die einzelnen Schritte sind in Abbildung 4.4 visualisiert. Alice sendet Daten an Bob (1). Eve empfängt die Daten und muss gleichzeitig die Übertragung stören, damit Bob die Daten nicht erhält (1). Sofern die Störung unbemerkt bleibt, manipuliert Eve die Daten (2) und sendet sie anschließend weiter an Bob (3). Dazu muss Eve ein eigenes elektromagnetisches Feld erzeugen. Zu diesem Zeitpunkt ist das elektromagnetische Feld von Alice nicht mehr aktiv und sie wartet auf eine Antwort von Bob. Die von Eve gesendeten

Daten werden dementsprechend sowohl von Bob als auch von Alice empfangen, wodurch der Angriff aufgedeckt wird (3).

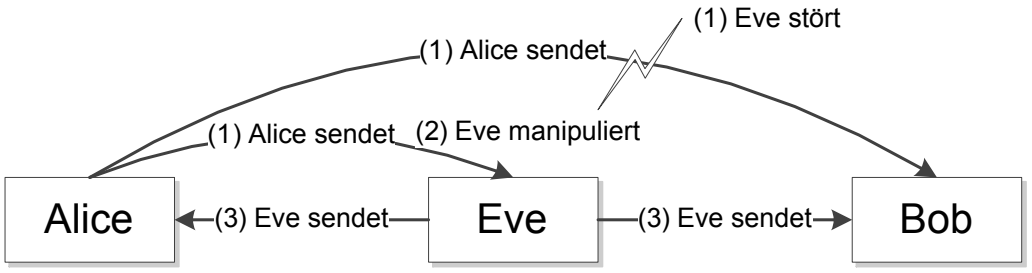


Abbildung 4.4: Ablauf von Man-in-the-Middle, wenn das elektromagnetische Feld im Wechsel erzeugt wird

4.4.4 Diebstahl/Verlust des Geräts

Die Anwendung speichert die ausgetauschten kryptographischen Schlüssel auf dem verwendeten Gerät selbst, dabei handelt es sich um ein Smartphone. Wird das Gerät gestohlen oder geht verloren, kann ein Angreifer auf die gespeicherten kryptographischen Schlüsseln zugreifen.

4.4.5 Zusammenfassung der Angriffe

Als Ergebnis der Analyse wurden die nachfolgenden Angriffe identifiziert:

A1 Übertragung abhören (Realisiert B1 und B2)

A2 Übertragung stören (Realisiert B3 und B4)

A3 Übertragung gezielt manipulieren (Realisiert B4)

A4 Daten in die Übertragung einfügen (Realisiert B4)

A5 Man-in-the-Middle (Realisiert B1 und B4)

A6 Verlust/Diebstahl (Realisiert B2)

4.5 Notwendige Sicherheitsmaßnahmen

Um den möglichen Angriffen zu begegnen, werden in diesem Abschnitt Sicherheitsmaßnahmen vorgestellt. Hierbei stehen technische Maßnahmen im Vordergrund. Um zu verhindern, dass die Übertragung abgehört und manipuliert werden kann, müssen die Daten verschlüsselt werden. Ausgehend davon, dass ein Man-in-the-Middle-Angriff nicht möglich ist oder erkannt wird, ist es ausreichend vor einer Datenübertragung ein Schlüsselaustauschprotokoll wie Diffie-Hellman durchzuführen. Mit dem erzeugten gemeinsamen Geheimnis werden die Daten verschlüsselt, bevor sie von dem Gerät gesendet werden. Nach der Übertragung werden sie auf der Seite des Empfängers entschlüsselt. Dieser Ansatz verhindert das Abhören und die Manipulation. Außerdem ist es einem Angreifer so nicht möglich eigene Daten in die Übertragung einzubringen.

Es ist weiterhin möglich, dass ein Angreifer die Datenübertragung stört. Nach Haselsteiner und Breitfuß kann ein solcher Angriff aber durch Überprüfen des elektromagnetischen Feld festgestellt werden [HB06]. Um unautorisierten Zugriff auf die gespeicherten Schlüssel zu verhindern, muss die Anwendung durch ein Passwort geschützt sein. Ohne das Passwort kann die Anwendung nicht geöffnet werden. Ein Angreifer ist also nicht in der Lage auf die gespeicherten kryptographischen Schlüssel zuzugreifen. Die Architektur von Android sieht vor, dass Anwendungen nur auf ihre eigenen Daten zugreifen können und nicht auf die von anderen Anwendungen. Dadurch ist sichergestellt, dass andere Anwendungen auf dem Gerät nicht auf die gespeicherten kryptographischen Schlüssel zugreifen können. Handelt es sich um ein gerootetes Gerät, ist nicht gewährleistet, dass die Sicherheitsmechanismen von Android greifen. Die in dieser Arbeit zu entwickelnde Anwendung richtet sich an reguläre Installationen. Sicherheitsmaßnahmen, die nur für gerootete Geräte nötig sind, werden dementsprechend nicht.

Als Ergebnis der Analyse wurden folgende Sicherheitsmaßnahmen erarbeitet, um allen Angriffen, ausgenommen der Übertragungsstörung, zu begegnen.

S1 Verschlüsselte Datenübertragung (Verhindert A1, A3 und A4)

S2 Zugriff auf Anwendung durch Passwort beschränken (Verhindert A6)

4.6 Konsequenz für die Anwendung

Um die in Abschnitt 4.5 beschriebene Sicherheitsmaßnahme umzusetzen, wird vor jeder Datenübertragung ein Diffie-Hellman-Schlüsselaustausch durchgeführt. Das Protokoll wird eingesetzt, um über einen ungesicherten Kanal ein gemeinsames Geheimnis zu vereinbaren. Das Geheimnis ist nur den Personen bekannt, die den Schlüsselaustausch durchgeführt haben. Die zu übertragenden Daten werden anschließend mit dem ausgehandelten Geheimnis verschlüsselt.

Im Nachfolgenden wird ein allgemeiner Überblick über den Ablauf des Protokolls gegeben, die mathematischen Grundlagen werden dabei nicht näher erläutert. Eine detaillierte Erklärung findet sich beispielsweise in Buchmanns „Einführung in die Kryptographie“ [Buc08]. Der Ablauf des Protokolls lässt sich in drei Phasen unterteilen. In der ersten Phase wählen die beiden Kommunikationspartner Alice und Bob eine Primzahl p und eine weitere Zahl g. Die Zahlen p und g sind öffentlich. Es ist möglich die beiden Zahlen im Voraus festzulegen, in diesem Fall entfällt die erste Phase. In der zweiten Phase wählen die Kommunikationspartner eine geheime Zahl k_A beziehungsweise k_B. Diese Zahlen werden nicht bekannt gegeben. Stattdessen berechnen Alice und Bob aus p, g und ihrer geheimen Zahl eine neue Zahl A beziehungsweise B. In der dritten Phase werden A und B dem jeweils anderen Kommunikationspartner mitgeteilt. Alice und Bob können nun aus den Zahlen A beziehungsweise B, ihrer geheimen Zahl und der Primzahl p ein gemeinsames Geheimnis berechnen. Ein Angreifer ist zwar in der Lage die ausgetauschten Zahlen abzufangen, kennt jedoch nicht die geheime Zahl von Alice oder Bob und ist deshalb nicht in der Lage das gemeinsame Geheimnis zu berechnen.

Um die Phasen 2 und 3 des Protokolls durchzuführen, müssen zweimal Daten ausgetauscht werden. Alice sendet A an Bob und dieser sendet B an Alice. Insgesamt müssen dreimal Daten übertragen werden, zweimal um den Diffie-Hellman-Schlüsselaustausch durchzuführen und einmal um die eigentlichen Daten zu übertragen.

Kapitel 5

Realisierung der Anwendung

Dieses Kapitel beschreibt wie die mobile Anwendung realisiert wurde. Die Anwendung setzt das entworfene Framework um und erweitert es um anwendungsspezifische Funktionalitäten. Die in Kapitel 4 gewonnenen Erkenntnisse werden dabei genutzt, um möglichen Angriffen zu begegnen.

Die Anwendung wird für das Betriebssystem Android entwickelt. Für das Betriebssystem spricht, dass mit Version 4.0 die Unterstützung der Near Field Communication weiter ausgebaut wurde. Das von Apple entwickelte Betriebssystem iOS unterstützt bisher kein NFC [Art12]. Unabhängig von der NFC-Unterstützung spricht auch der Marktanteil von Android für das Betriebssystem. Nach einem 2013 von comSocre[1] veröffentlichten Bericht, ist Android zum Zeitpunkt der Veröffentlichung, mit über 50% das Betriebssystem mit dem größten Marktanteil im Smartphone-Bereich [com13]. Für die Implementierung wird die Entwicklungsumgebung Eclipse[2] mit dem von Android bereitgestellten ADT-Plugin[3] genutzt. Als Testgeräte werden das Samsung Nexus S und das Samsung Galaxy Nexus verwendet. Auf den Geräten sind, zum Zeitpunkt der Entwicklung, die Versionen 4.1 und 4.0.4 installiert.

[1] Analytics for a Digital World: http://www.comscore.com, Zuletzt abgerufen am 25.08.2013
[2] Eclipse: http://www.eclipse.org, Zuletzt abgerufen am 25.08.2013
[3] ADT Plugin: http://developer.android.com/tools/sdk/eclipse-adt.html, Zuletzt abgerufen am 25.08.2013

5.1 Umsetzung der Komponenten

Im Folgenden wird erklärt, wie die Instanzen für die im Framework spezifizierten Komponenten implementiert worden sind. Die einzelnen Klassen und Activities aus denen die Anwendung besteht, sind in Java-Packages strukturiert. Diese setzen jeweils die Funktionalitäten einer Komponente um. Die Anwendung besteht aus den Packages: `Cryptography`, `Exchange`, `Management`, `Storage`, `Application`, `Misc` und `Facade`. Die ersten vier Packages setzen im Wesentlichen die Funktionalitäten des Frameworks um. Je nach Funktionsumfang sind sie in weitere Packages unterteilt. Das `Application`-Package enthält den Login-Dialog und das Hauptmenü. Im `Misc`-Package sind allgemeine Klassen, Activities und Objekte enthalten, die von mehreren anderen Packages genutzt werden. Durch diese Aufteilung wird redundanter Code vermieden. Das `Facade`-Package enthält Facaden, die Schnittstellen für die Packages `Cryptography`, `Exchange`, `Management` und `Storage` anbieten. An dieser Stelle wurde das Facade-Pattern [FFSB04] angewendet, um Abhängigkeiten zwischen den Packages zu minimieren. Eine Facade kapselt mehrere Klassen und Activities, indem sie deren Methoden gesammelt bereitstellt. Benötigt eine Activity, eine Funktionalität aus einem anderen Package, nutzt sie die entsprechende Facade. Wenn sich Klassen oder Activities in einem Package ändern, muss nur die Facade angepasst werden. Änderungen in den anderen Packages werden so vermieden. Darüber hinaus erhöht das Pattern die Wiederverwendbarkeit. Ein Entwickler kann durch die Facade die Funktionalität eines Packages nutzen, ohne sich vorher mit dem eigentlichen Package auseinandersetzen zu müssen. Es ist aber auch weiterhin möglich die Methoden eines Packages direkt aufzurufen, ohne die Facade zu nutzen.

5.1.1 Application-Komponente

Das `Application`-Package beinhaltet beinhaltet die `LoginActivity` und die `MenuActivity`. Die `LoginActivity` dient als Startbildschirm in dem ein Passwort eingegeben werden muss um die Anwendung zu starten. Die `MenuActivity` ist das Hauptmenü der Anwendung vom dem aus alle anderen Bestandteile gestartet werden. Weitere anwendungsspezifische Klassen und Activities sind als Unterpackages von `Cryptography`, `Exchange`, `Management` und `Storage`

umgesetzt. Der Programmcode wird mit einer Textdatei ausgeliefert, die darauf hinweist, welche Packages zum Framework gehören und welche neu hinzugekommen sind.

5.1.2 Management-Komponente

Die Funktionalität der in Abschnitt 3.2.1 beschriebenen Komponente wird in der Anwendung durch das `Management`-Package realisiert. Es ist unterteilt in die Packages `Detail`, `Overview`, `Access` und `Settings`. Die Packages `Access` und `Settings` realisieren anwendungsspezifische Funktionalitäten, die nicht aus dem Framework stammen.

Die `DetailActivity` aus dem `Detail`-Package erlaubt es, Schlüsseleinträge zu erstellen und zu bearbeiten. Die graphische Oberfläche besteht aus zwei Textfeldern. Das erste Feld ist für eine Beschreibung des kryptographischen Schlüssels und das zweite für den Ersteller des Schlüssels vorgesehen. Mittels eines Buttons kann der Nutzer ein RSA-Schlüsselpaar erzeugen. Dieses besteht aus einem öffentlichen und einem privaten Schlüssel und kann zur asymmetrischen Verschlüsselung genutzt werden. Zum Erzeugen des Schlüsselpaars werden Methoden aus dem `Cryptography`-Package verwendet. Anschließend kann der Nutzer den Schlüsseleintrag speichern. Wie im Framework vorgesehen, nutzt die Activity dazu die Schnittstellen der `Storage`-Komponente.

Die gespeicherten Schlüsseleinträge lassen sich mit der `OverviewActivity` aus dem `Overview`-Package verwalten. Die Activity visualisiert gespeicherte Schlüsseleinträge in einer Liste. Es kann zwischen zwei Ansichten gewechselt werden, in denen entweder selbst erstellte Schlüsseleinträge angezeigt werden oder Schlüsseleinträge von anderen Personen. Die Liste zeigt nur die Beschreibung und den Ersteller eines Schlüssels an. Die Inhalte werden über die Schnittstelle der `Storage`-Komponente abgerufen. In beiden Ansichten können Schlüsseleinträge zum Löschen markiert werden. Klickt ein Nutzer auf einen der Schlüsseleinträge, wird die `DetailActivity` geöffnet. Die Textfelder enthalten die Informationen aus dem Schlüsseleintrag. Die Informationen können bearbeitet und anschließend gespeichert werden. Es ist nicht möglich, das eigentliche RSA-Schlüsselpaar nachträglich zu verändern.

Die `AccessActivity` aus dem `Access`-Package ermöglicht es, den Zugriff auf die gespeicherten Schlüsseleinträge zu verwalten. Alle auf dem Gerät in-

stallierten Anwendungen werden in einem Dropdown-Menü angezeigt. Systemanwendungen sind in der Auswahl nicht enthalten. Ein grüner Pfeil oder ein rotes Kreuz neben einer Anwendung zeigt an, ob diese auf die Schlüsseleinträge zugreifen darf oder nicht. Über zwei Buttons, kann der Zugriff gewährt oder entzogen werden. Wie in Abschnitt 2.3.2.3 beschrieben, kann eine Anwendung ihre Daten mit einem Content Provider für andere Anwendungen bereitstellen. Diese können in ihrem Manifest definieren, dass sie auf den Content Provider zugreifen dürfen. In diesem Fall ist der Zugriff auf den Content Provider dauerhaft. Im Rahmen dieser Anwendung ist ein dauerhafter Zugriff unerwünscht, da es dem Nutzer möglich sein soll, den Zugriff dynamisch zu gewähren und wieder zu entziehen. Um den Zugriff zu erlauben und zu entziehen, werden die von Android bereitgestellten Methoden `grantUriPermission()` und `revokeUriPermission()` benutzt. Die Methode `grantUriPermission()` erlaubt es einer Anwendung solange auf den Content Provider zuzugreifen, bis die Erlaubnis widerrufen wird. Wird das Smartphone ausgeschaltet, werden alle temporären Erlaubnisse automatisch widerrufen. Durch diese Einschränkung muss der Nutzer nach jedem Neustart die `AccessActivity` öffnen, damit die Erlaubnisse abermals erteilt werden.

Das `Settings`-Package beinhaltet die `SettingsActivity`. Mit der Activity kann die Passworteingabe beim Start der Anwendung deaktiviert werden. Außerdem lässt sich das Passwort ändern. In beiden Fällen muss der Nutzer das bisherige Passwort eingeben, um die Änderung vorzunehmen.

5.1.3 Storage-Komponente

Im Framework wird die `Storage`-Komponente genutzt, um Daten zu speichern. In der Anwendung wird sie eingesetzt, um Schlüsseleinträge zu speichern. Die Anwendung verwendet hierfür die in Android integrierte SQLite-Datenbank. Die Tabelle für Schlüsseleinträge besteht aus folgenden Spalten: `id`, `description`, `creator`, `publicModulus`, `publicExponent`, `privateModulus` und `privateExponent`. In der Spalte `description` wird eine Beschreibung zum kryptographische Schlüssel gespeichert und in der Spalte `creator` der Name des Erstellers oder ein Pseudonym. Die restlichen Felder können genutzt werden, um die Bestandteile eines öffentlichen und eines privaten Schlüssel zu speichern. Diese Struktur wird in der Klasse `KeyTable` festgelegt. Die Klassen `KeyDatabase-`

`Helper` und `KeyContentProvider` kapseln den Programmcode, der benötigt wird, um mit der Datenbank arbeiten zu können.

5.1.4 Exchange-Komponente

Die `Exchange`-Komponente soll Schnittstellen anbieten, um einen Kanal auf- und abzubauen, auf dem Daten übertragen werden können. Im Rahmen der Anwendung wird die geforderte Funktionalität nicht durch eine einzige Instanz realisiert, sondern auf mehrere Klassen und Activities aufgeteilt. Dies ist der Tatsache geschuldet, dass zur Datenübertragung die Beam-API genutzt wird. Die API wird eingesetzt, um entsprechend kodierte Daten mittels NFC zu übertragen. Nachfolgend wird erklärt, wie die Anwendung Daten kodiert, sendet und empfängt.

Die Übertragung mittels Beam läuft wie folgt ab: Die Activity muss eine NDEF-Message an die API übergeben. Anschließend muss das Smartphone an ein zweites Smartphone gehalten werden, das ebenfalls die Beam-API unterstützt. Der Nutzer wird aufgefordert, die Übertragung zu bestätigen, woraufhin die Daten gesendet werden. Die Anwendung selbst muss dementsprechend keine eigene Funktionalität bereitstellen, um einen Übertragungskanal auf- und abzubauen.

Activities, die Daten übertragen sollen, nutzen die Methoden der `NFCHelper`-Klasse aus dem `Misc`-Package, um die Daten zu kodieren. Die `NFCHelper`-Klasse bietet eine Schnittstelle an, um Daten in einer NDEF-Message zu kapseln, die sich aus Records zusammensetzt. Diese bestehen wiederum aus den eigentlichen Daten und einer Zusatzinformation, um die Art der Daten zu beschreiben. Die Art der Daten wird in der Anwendung als MIME-Type angegeben. Innerhalb der `NFCHelper`-Klasse werden NDEF-Records mit der Methode `createMimeRecord()` erstellt. Die Methode nimmt als Parameter den Typ, der zu übertragenden Daten und die Daten selbst entgegen.

Anschließend werden die kodierten Daten an die Beam-API übergeben. Die Activities implementieren dazu das `CreateNdefMessageCallback`-Interface und überschreiben die Methode `createNdefMessage()`. Wenn sich zwei Smartphones in kurzer Distanz befinden, wird die Methode automatisch aufgerufen. Nach der Bestätigung durch den Nutzer werden die Daten übertragen. Wenn ein Smartphone eine NDEF-Message empfängt, wird intern ein Intent gesendet. Damit eine Activity auf diesen Intent reagieren kann, muss sie einen passenden

Intent-Filter definieren. Listing 5.1 zeigt einen solchen Filter.

```
<intent-filter>
  <action android:name="android.nfc.action.NDEF_DISCOVERED" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:mimeType="application/keyexchange.cryptography.step1" />
</intent-filter>
```

Listing 5.1: Intent-Filter der `DiffieHellman`-Activity

Der Filter legt fest, dass auf empfangene NDEF-Messages reagiert werden soll, sofern die Daten zu dem festgelegten MIME-Typ passen. Nachdem eine NDEF-Message empfangen wurde, kann die Activity die gesendeten Daten aus den einzelnen NDEF-Records wiederherstellen und mit ihnen arbeiten.

Abbildung 5.1 zeigt beispielhaft die Schnittstellen, welche beim Empfangen einer Nachricht angesprochen werden. Zuerst werden die Daten über die Schnittstelle des Geräts empfangen, dies geschieht mittels der Beam-API. Anschließend reagiert die Anwendung auf den vom System gesendeten Intent und leitet die Daten an die Activity weiter, die in diesem Fall zur `Cryptography`-Komponente gehört.

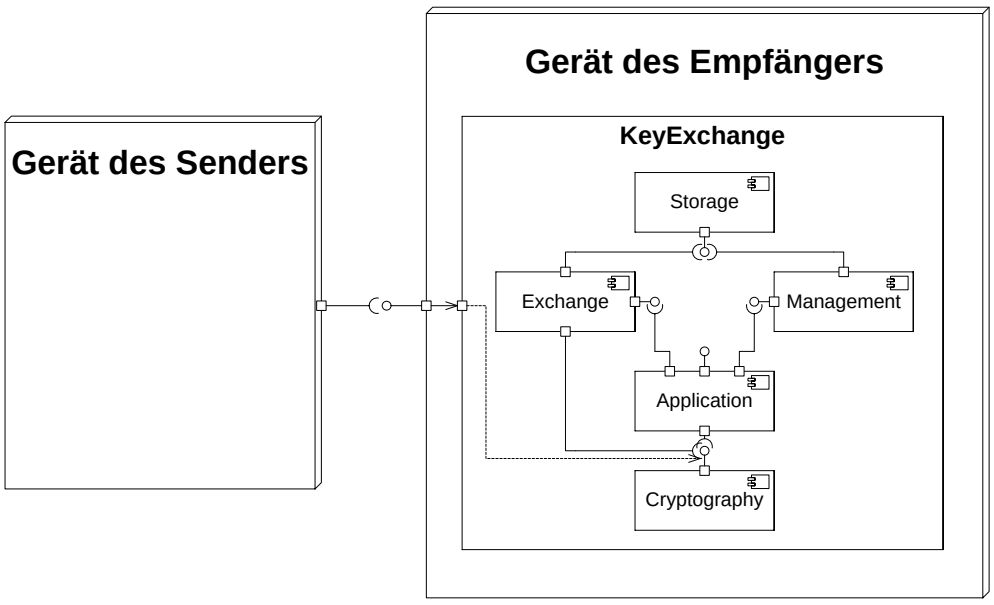


Abbildung 5.1: Empfangen einer Nachricht

Die Datenübertragung wird, wie zuvor beschrieben, von der Beam-API umgesetzt und nicht vom `Exchange`-Package wie im Framework vorgesehen. Das Package besteht stattdessen aus den Activities `TransferActivity` und `SaveActvity`. Die `TransferActivity` zeigt alle selbst erstellen Schlüsseleinträge an. Der Sender kann auswählen, welche Einträge er übertragen möchte. Nachdem eine Auswahl getroffen wurde, wird eine Activity aus dem `Cryptography`-Package aufgerufen, um die Daten sicher zu übertragen. Dieser Vorgang wird in Abschnitt 5.1.5 beschrieben. Nachdem die Schlüsseleinträge gesendet wurden und beim Empfänger eingegangen sind, werden sie auf dessen Gerät mittels der `SaveActivity` angezeigt. Der Empfänger kann auswählen, welche der Schlüsseleinträge gespeichert werden sollen. Um die Einträge zu speichern wird, die Funktionalität des `Storage`-Packages verwendet.

5.1.5 Cryptography-Komponente

Die im Entwurf des Frameworks beschriebene `Cryptography`-Komponente bietet Schnittstellen an, um den Übertragungskanal abzusichern. Gemäß der in Abschnitt 4.6 gewonnenen Erkenntnisse findet die Absicherung in zwei Phasen statt. In der ersten Phase wird ein Diffie-Hellman-Schlüsselaustausch zwischen Sender und Empfänger durchgeführt. Das ausgehandelte Geheimnis wird in der nächsten Phase benutzt, um die zu übertragenden Daten zu verschlüsseln. Die Phasen sind auf die Activities `SecureChannelStep1` und `SecureChannelStep2` aufgeteilt. Diese Activities befinden sich in den Unterpackages `Step1` und `Step2`. Zusätzlich gehören zu dem `Cryptography`-Package die Packages `Generation` und `Encryption`. Das `Generation`-Package bietet Methoden, um RSA- und Diffie-Hellman-Schlüssel zu erzeugen. Es wird unter anderem von der Activity `SecureChannelStep1` verwendet, um das Diffie-Hellman-Protokoll durchzuführen. Das `Encryption`-Package beinhaltet die Klasse `AESEncryptionWorker`, mit der sich Daten mit dem Advanced Encryption Standard verschlüsseln und entschlüsseln lassen. Diese Methoden werden von der Activity `SecureChannelStep2` genutzt.

Als Erstes wird der Diffie-Hellman-Schlüsselaustausch durchgeführt. Dazu nutz die Activity `SecureChannelStep1` Funktionalitäten des `DHWorker` aus dem `Generation`-Package. Dieser verwendet Klassen aus dem `javax.cryp`

`to`-Package[4], um ein Diffie-Hellman-Schlüsselpaar zu erzeugen und später das gemeinsame Geheimnis zu erstellen. Für die Zahlen p und g werden feste Werte verwendet. Diese Werte stammen aus dem technischen Dokument RFC 5114 [LK08]. Das Dokument beschreibt acht Diffie-Hellman Gruppen, die in einer Implementation verwendet werden können. Die verwendete Primzahl ist 1024 Bit lang.

Der Diffie-Hellman-Schlüsselaustausch findet in drei Schritten statt. In der `SecureChannelStep1`- Activity wird zwischen den vier Zuständen `A`, `B`, `C` und `D` unterschieden. Je nach Zustand sollen die einzelnen Methoden der Activity unterschiedlich arbeiten. So soll die Methode `updateInterface()` abhängig vom Zustand einen anderen Informationstext anzeigen. Außerdem sollen abhängig vom Zustand unterschiedliche Daten an das andere Gerät gesendet werden. Um dieses Verhalten zu realisieren, wurde das State Pattern [FFSB04] angewendet. Es wurde ein Interface erstellt, das alle Methoden definiert, die zustandsabhängig anders arbeiten sollen. Für jeden Zustand wurde eine Klasse erzeugt, die das Interface implementiert und die Methoden entsprechend anpasst. Die `SecureChannelStep1`-Activity instanziiert das Interface mit der Klasse, die den aktuellen Zustand umsetzt und ruft die Methoden dieser Instanz auf. Wenn sich der Zustand ändert, wird das Interface neu instanziiert. Die Methoden werden weiterhin über die Instanz des Interfaces aufgerufen, arbeiten jedoch in diesem Fall anders.

Die einzelnen Abläufe sind in Abbildung 5.2 dargestellt. Im ersten Schritt wird, auf der Seite des Senders, ein Schlüsselpaar für den Diffie-Hellman-Schlüsselaustausch erzeugt (1). Das Schlüsselpaar soll nach dem Start der Activity erzeugt werden und dann für die gesamte Lebensdauer zur Verfügung stehen. Aus diesem Grund wird es in `onCreate()` erzeugt. Das Schlüsselpaar besteht aus einem privaten und einem öffentlichen Schlüssel. Die Anwendung befindet sich in Zustand `A`. Anschließend wird der öffentliche Schlüssel an den Empfänger gesendet (2). Die Datenübertragung erfolgt wie in Abschnitt 5.1.4 beschrieben. Damit ist der erste Schritt abgeschlossen und die Anwendung des Senders erwartet eine Rückmeldung des Empfängers. Sie wechselt in den Wartezustand `D`.

Nachdem das Gerät des Empfängers die Nachricht erhalten und eine Instanz der `SecureChannelStep1`-Activity erzeugt hat, wird die Methode `onCreate()`

[4]Crypto-Package: http://developer.android.com/reference/javax/crypto/package-summary.html, Zuletzt abgerufen am 25.08.2013

aufgerufen und ein Schlüsselpaar erstellt (3). Mit dem eigenen privaten Schlüssel und dem empfangenen öffentlichen Schlüssel des Senders wird ein gemeinsames Geheimnis generiert (4). Das gemeinsame Geheimnis wird durch den `DHWorker` erzeugt. Die Activity ist zu diesem Zeitpunkt in Zustand `B`. Damit die Anwendung des Senders ebenfalls, das gemeinsame Geheimnis erzeugen kann, überträgt die Anwendung des Empfängers den öffentlichen Schlüssel (5). Nachdem der öffentliche Schlüssel gesendet wurde, ist der Diffie-Hellman-Schlüsselaustausch auf der Seite des Empfängers abgeschlossen und die Activity wechselt in Endzustand `C`.

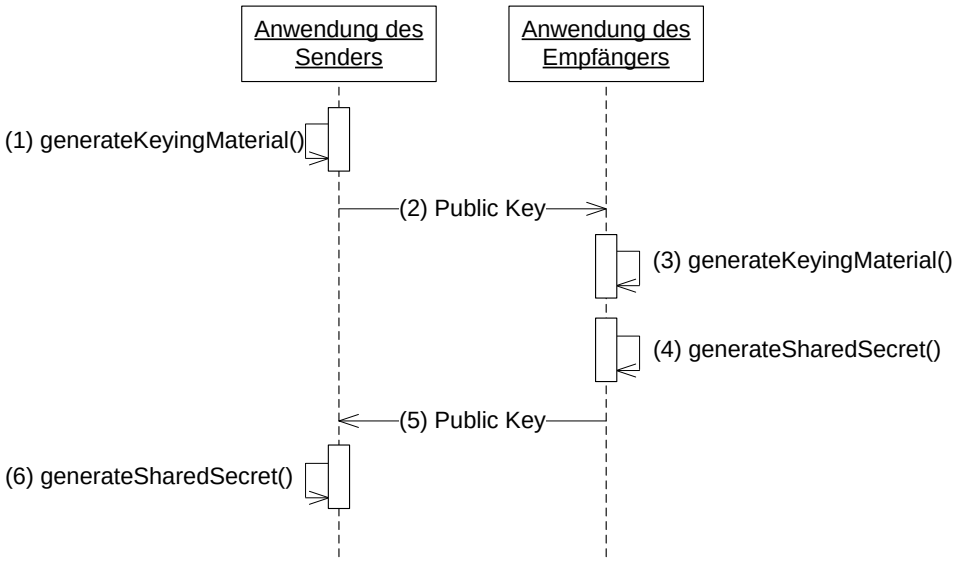


Abbildung 5.2: Ablauf des Diffie-Hellman-Schlüsselaustauschs

Nachdem der ursprüngliche Sender den öffentlichen Schlüssel des Empfängers erhalten hat, erzeugt er ebenfalls, das gemeinsame Geheimnis Abbildung (6) und wechselt anschließend ebenfalls in Zustand `C`. Sowohl Sender als auch Empfänger verfügen zu diesem Zeitpunkt über das gemeinsame Geheimnis, womit der Diffie-Hellman-Schlüsselaustausch abgeschlossen ist.

Um den Diffie-Hellman-Schlüsselaustausch durchzuführen, müssen die Werte, die in den einzelnen Schritten berechnet wurden für den gesamten Ablauf

verfügbar sein. Dies hat im Rahmen der Implementierung zu einem Problem geführt. Wie zuvor beschrieben, werden die Daten mittels NDEF-Messages ausgetauscht. Empfangene NDEF-Messages lösen einen Intent aus, welcher von der `SecureChannelStep1`-Activity behandelt wird. Ein Intent erzeugt eine neue Instanz, einer Activity. Übertragen auf die Anwendung bedeutet dies Folgendes: Startet die Activity auf Seiten des Senders, wird die Methode `onCreate()` aufgerufen und ein Schlüsselpaar erzeugt. Der Empfänger erhält den öffentlichen Schlüssel vom Sender. Auf Seiten des Empfängers wird eine Instanz der `SecureChannelStep1`-Activity erzeugt. Es werden nacheinander die Funktionen `onCreate()` und `onResume()` aufgerufen. Das gemeinsame Geheimnis wird also in einer einzigen Instanz der Activity erstellt. Wenn der Empfänger allerdings seinen öffentlichen Schlüssel sendet, wird auf Seiten des Senders eine neue Instanz der `SecureChannelStep1`-Activity erzeugt. In dieser Instanz wird ein neues Schlüsselpaar generiert, wodurch das berechnete gemeinsame Geheimnis nicht mit dem des Empfängers übereinstimmt.

Um dieses Problem zu lösen, wird im Manifest für die `SecureChannelStep1`-Activity der Wert des Attribut `launchMode` auf `singleTask` gesetzt. Dadurch wird verhindert, dass jeder Intent eine neue Instanz der Activity erzeugt. Stattdessen werden Intents über den Aufruf von `onNewIntent()` an die bestehende Instanz weitergeleitet. Die `SecureChannelStep1`-Activity muss die `onNewIntent`-Methode überschreiben. Ein ähnliches Problem besteht mit der Funktionalität des Zurück-Buttons. Wie in Abschnitt 2.3.2.1 beschrieben verwaltet Android Activities in einem Stack.

Mit dem Zurück-Button kann der Nutzer zum vorherigen Bildschirm zurückkehren. Dabei wird die aktuell dargestellte Activity zerstört. Dieses Verhalten ist während des Ablaufs des Diffie-Hellman-Schlüsselaustauschs und der anschließenden Verschlüsselung unerwünscht. Sollte der Nutzer eine der Activities schließen, muss der gesamte Ablauf erneut durchgeführt werden, da die zu diesem Zeitpunkt vorhandenen Daten gelöscht werden. Um dies zu verhindern, wurde die Funktionalität des Zurück-Buttons überschrieben. Wird der Button gedrückt, erzeugt die Anwendung einen Dialog, der den Nutzer darauf hinweist, dass diese Aktion die Übertragung abbricht. Der Nutzer hat dann die Möglichkeit die Übertragung abzubrechen oder sie fortzuführen. Die `SecureChannelStep2`-Activity, welche für die Verschlüsselung verantwortlich ist, benutzt ebenfalls den modifizierten Zurück-Button. Nach dem der Diffie-Hellman-Schlüssel-

austausch durchgeführt wurde, ruft die Anwendung des Senders die `SecureChannelStep2`-Activity auf. Der Ablauf der Ver-/Entschlüsselung ist in Abbildung 5.3 dargestellt. Die Activity verfügt ebenfalls über unterschiedliche Zustände, die mittels des State Patterns realisiert worden sind.

Auf der Seite des Senders und des Empfängers wird das gemeinsame Geheimnis genutzt, um Schlüssel für den Advanced Encryption Standard (AES) zu erzeugen (1). AES unterstützt Schlüssel mit einer Länge von 128, 192 oder 256 Bit. In der Anwendung wird ein Schlüssel mit einer Länge von 256 Bit verwendet. Aus diesem Grund werden nur die ersten 32 Byte (256 Bit) des gemeinsamen Geheimnisses verwendet. Der so erzeugte Schlüssel wird anschließend genutzt, um die zu übertragenden Daten auf der Seite des Senders zu verschlüsseln und auf der Seite des Empfängers wieder zu entschlüsseln. Für das Erzeugen des Schlüssels und die anschließende Verschlüsselung wird die Klasse `AESEncryptionWorker` aus dem `Encryption`-Package genutzt. Die Anwendung des Senders verschlüsselt die Daten im nächsten Schritt (2). Es werden nur die Beschreibung, der Ersteller des Schlüssels und die Bestandteile des öffentlichen Schlüssels gesendet. Die verschlüsselten Daten werden wie zuvor mittels NFC an den Empfänger übermittelt (3). Der Empfänger entschlüsselt die Daten mit dem Schlüssel und kann sie anschließend verwenden (4). Nachdem die Daten entschlüsselt

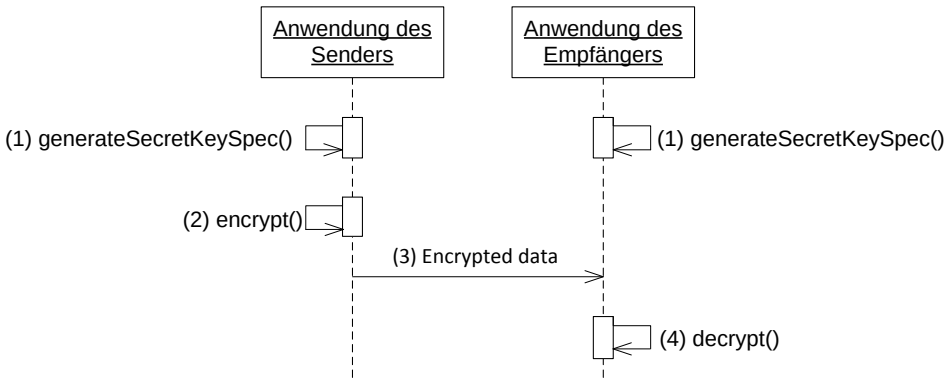


Abbildung 5.3: Ablauf der Ver-/Entschlüsselung

worden sind, werden sie an die `SaveActivity` aus dem `Management`-Package übergeben.

5.2 Nachteile der Beam-API

Während der Entwicklung der Anwendung sind Einschränkungen der Beam-API deutlich geworden. Sie ist zum Zeitpunkt der Entwicklung die einzige Möglichkeit Daten mittels NFC zwischen zwei Smartphones auszutauschen. Jede Datenübertragung muss manuell vom Nutzer bestätigt werden. So wird ausgeschlossen, dass unbemerkt Daten gesendet werden. Gleichzeitig wird es dadurch jedoch schwierig, NFC für bidirektionale Übertragungen zu nutzen. Wie in Abschnitt 5.1.5 beschrieben, wird vor jeder Datenübertragung ein Diffie-Hellman-Schlüsselaustausch durchgeführt, damit die zu übertragenden Daten verschlüsselt werden können.

Um den Diffie-Hellman-Schlüsselaustausch durchzuführen, muss der Sender im ersten Schritt seinen öffentlichen Schlüssel an den Empfänger senden, woraufhin dieser im zweiten Schritt seinen öffentlichen Schlüssel an den Sender schickt. Im Anschluss kann der Schlüsseleintrag verschlüsselt und übertragen werden. Insgesamt müssen also dreimal Daten übertragen werden, wobei jede Übertragung bestätigt werden muss. Außerdem müssen die Smartphones nach jeder Übertragung neu aneinander gehalten werden, um erneut Daten mittels Beam zu übertragen. Die eigentlich einfache Übertragung, wird durch diese Einschränkungen aufwendiger. Der Vorteil von NFC, Daten schnell und unkompliziert zu übertragen, wird dadurch abgeschwächt.

Aufgrund der Tatsache, dass die Near Field Communication Daten unverschlüsselt überträgt muss eine Absicherung auf Seiten der Anwendung realisiert werden. Eine wie in dieser Arbeit umgesetzte Verschlüsselung ist mit der aktuellen API allerdings nur schwer zu realisieren. Es sollte stattdessen möglich sein, dass die beiden Nutzer einer Übertragung zustimmen, in deren Verlauf Daten bidirektional ausgetauscht werden können. Dieser Ansatz würde die Datenübertragung vereinfachen. Zwei Nutzer müssten ihre Smartphones aneinander halten und der Übertragung zustimmen. Anschließend erfolgt automatisch der Diffie-Hellman-Schlüsselaustausch. Die Daten werden verschlüsselt und übertragen. Anstatt drei einzelner Schritte, die die Interaktion des Nutzers erfordern, findet der gesamte Ablauf in einer einzigen Übertragung statt. Dadurch, dass die Nut-

zer die Übertragung zuvor bestätigen müssen, ist es nicht möglich, dass Daten vollkommen unbemerkt gesendet werden. Allerdings ist es für die Nutzer weniger transparent, welche und wie viele Daten tatsächlich ausgetauscht werden.

Unabhängig davon, dass die eingeschränkte Funktionalität der Beam-API im Rahmen dieser Arbeit zu Schwierigkeiten geführt hat, ist eine bidirektionale Kommunikation allgemein sinnvoll. Aktuell ist es zum Beispiel nicht möglich eine Rückmeldung zu erhalten, ob die gesendeten Daten beim Empfänger angekommen sind. Der Empfänger kann am Verhalten seiner Anwendung feststellen, dass Daten nicht korrekt oder gar nicht übertragen worden sind. Dann muss im Anschluss eine neue Übertragung durchgeführt werden. Eine Rückmeldung, dass die Daten empfangen worden sind, ermöglicht es den Prozess zu automatisieren.

Abschließend lässt sich festhalten, dass eine weniger restriktive Variante der Beam-API sinnvoll ist, um bidirektionale Kommunikation zu ermöglichen. Im März 2012 wurde eine entsprechende Änderung als Verbesserungsvorschlag auf der Projektseite von Android eingereicht[5]. Einer der Projektverantwortlichen hat angekündigt, dass Erweiterungen der Beam-API geplant sind. Zum aktuellen Zeitpunkt, August 2013, ist eine bidirektionale Kommunikation über Beam weiterhin nicht möglich.

[5] Android Issue 28014: http://code.google.com/p/android/issues/detail?id=28014, Zuletzt abgerufen am 25.08.2013

Kapitel 6

Evaluierung der Anwendung

Im Folgenden werden die umgesetzte Anwendung, die Sicherheitsmaßnahmen und das zugrunde liegende Framework evaluiert.

6.1 Aktueller Funktionsumfang

Die Anwendung unterstützt die in Kapitel 3.1 vorgesehenen Grundfunktionen. Wie in dem anfänglichen Szenario beschrieben, ermöglicht sie den sicheren Austausch von kryptographischen Schlüsseln zwischen zwei Personen, welche die Anwendung nutzen. Um die Sicherheit zu gewährleisten, werden die kryptographischen Schlüssel vor der Datenübertragung verschlüsselt. Dadurch ist es einem Angreifer nicht möglich die kryptographischen Schlüssel abzufangen und zu manipulieren. Darüber hinaus können kryptographische Schlüssel direkt in der Anwendung erzeugt und verwaltet werden. Die gespeicherten kryptographischen Schlüssel können anderen Anwendungen zur Verfügung gestellt werden. Die Anwendung ermöglicht somit nicht nur den sicheren Datenaustausch, sondern kann auch zur Verwaltung der kryptographischen Schlüssel genutzt werden. Zukünftig kann die Anwendung erweitert und an verschiedene Anwendungsfälle angepasst werden. Eine mögliche Erweiterung ist die Einführung weiterer kryptographischer Schlüssel. Aktuell werden bereits RSA-Schlüssel unterstützt. Mögliche andere Anwendungsfälle werden im Ausblick vorgestellt.

Der Export und Import von kryptographischen Schlüsseln wurde in der Anwendung nicht umgesetzt. Es handelt sich hierbei zwar um eine sinnvolle Funktionalität, die jedoch über das eigentliche Konzept des sicheren Datenaustauschs

hinausgeht. Da in der Realisierung die Umsetzung der Funktionalität im Vordergrund stand, sind die grafische Oberfläche und die Menüführung schlicht gehalten. In Zukunft sollte ein Usability-Test durchgeführt werden, um Feedback einzuholen wie die Menüführung verbessert werden kann.

6.2 Umsetzung des Frameworks

Zuerst wird beschrieben, inwieweit die Spezifikation umgesetzt werden konnte. Anschließend wird eine Einschätzung gegeben, wie gut sich die einzelnen Komponenten des Frameworks wiederverwenden lassen. Der überwiegende Teil der im Entwurf spezifizierten Framework-Komponenten konnten ohne Änderung implementiert werden. Eine Ausnahme bildet die `Exchange`-Komponente, die nicht mit einer einzelnen Instanz realisiert wurde. Stattdessen wurde die Funktionalität in mehrere Activities integriert. Damit konnte die klare Trennung aus dem Framework, an dieser Stelle nicht eingehalten werden. Diese Änderung ergab sich aus der Nutzung der Beam-API. Wie in Abschnitt 5.1.4 erklärt, baut Beam eine Verbindung zwischen zwei NFC-fähigen Smartphones auf, die für eine einzelne Datenübertragung genutzt werden kann. Die in der Spezifikation vorgesehenen Methoden, um eine Verbindung auf- und abzubauen, entfallen dementsprechend. Damit eine Activity Daten mittels Beam übertragen kann, muss sie bestimmte Interfaces nutzen und deren Methoden überschreiben. Daher konnte der Datenaustausch nicht über eine einzelne Instanz realisiert werden.

Betroffen sind die Activities `SecureChannelStep1` und `SecureChannel-Step2`, welche Daten übertragen müssen. Ursprünglich war vorgesehen, dass die Activities zur Datenübertragung die `Exchange`-Komponente nutzen. In der Anwendung müssen die Activities direkt die Beam-API nutzen. Dazu müssen sie die zu übertragenden Daten in einer NDEF-Message kapseln. Die Funktionalität, um solche NDEF-Messages zu erzeugen, wurde in eine eigene Klasse ausgelagert. Diese wird von beiden Activities genutzt werden kann. Dadurch wurde Redundanz im Code vermieden und die Wartbarkeit erleichtert.

Außerdem konnte der Datenaustausch nicht genau wie in Abschnitt 3.2.5 beschrieben umgesetzt werden. Aufgrund der genutzten API ist keine bidirektionale Datenübertragung möglich, weshalb darauf verzichtet wurde eine Bestätigung für empfangene Daten zu senden. Die Absicherung der Verbindung sollte eigentlich im Hintergrund erfolgen. Dadurch, dass der Nutzer jede Daten-

übertragung bestätigen muss, ist der Ablauf aufwendiger als ursprünglich angedacht. Sollte die Beam-API in Zukunft bidirektionale Kommunikation unterstützen, kann dieser Aspekt der Anwendung angepasst werden, um die Datenübertragung zu vereinfachen. In der aktuellen Version werden dem Nutzer während der Durchführung des Diffie-Hellman-Schlüsselaustauschs und der Verschlüsselung Informationen angezeigt, um den Vorgang für den Nutzer transparenter zu gestalten.

Die in der Anwendung enthaltenen Packages entsprechen im Wesentlichen den im Framework beschriebenen Komponenten. Anwendungsspezifische Funktionalitäten wurden als Unterpackages realisiert. Durch diese Trennung sind die Funktionalitäten des Frameworks und der Anwendung entkoppelt. Die Packages bieten ihre Funktionalitäten über Schnittstellen an. Die konkreten Schnittstellen weichen dabei teilweise von der Spezifikation des Frameworks ab, da diese bewusst sehr allgemein gehalten wurde und sich nicht speziell auf Android bezog. Intern wird die Funktionalität über Activities und Klassen realisiert. Jedes Package kann gegen ein Neues ausgetauscht werden,dafür ist lediglich die zugehörige Facade anzupassen. Weiter Änderungen in anderen Packages werden somit vermieden. Zudem besteht die Möglichkeit ein Package in einer anderen Anwendung einzusetzen. Hierfür muss der Entwickler lediglich die Schnittstellen aus der Facade kennen. Nur wenn die Funktionalität angepasst werden soll, muss sich ein Entwickler mit dem Programmcode beschäftigen. Dadurch ist der Programmcode modular und kann gut wiederverwendet werden.

6.3 Bewertung der Sicherheit

Die in Kapitel 4 erarbeiteten Sicherheitsmaßnahmen wurden in der Anwendung umgesetzt. Allerdings ist die NFC-Unterstützung von Android eingeschränkt, woraus sich Probleme ergeben haben, die im Folgenden erklärt werden: Der in Abschnitt 4.4.2 beschriebene Angriff zielt darauf ab, die Verbindung zu stören, damit mittels NFC keine Daten übertragen werden können. Das Android-Betriebssystem stellt keine Methoden zur Verfügung, mit denen das elektromagnetische Feld überprüft werden kann, um eine solche Störung zu erkennen. Wird die Verbindung während des Schlüsselaustauschs gestört, werden die übertragenen Daten zerstört und können nicht genutzt werden. Dies führt zu einem Fehler in der Anwendung, der vom Nutzer erkannt werden kann. Der Schlüsselaustausch

muss in diesem Fall erneut durchgeführt werden. Es handelt sich hierbei um einen klassischen Denial-of-Service-Angriff. Die Vertraulichkeit der kryptographischen Schlüssel wird nicht gefährdet, weshalb das Risiko eines solchen Angriffs akzeptiert wird.

Das Fehlen von Funktionalität, um das elektromagnetische Feld zu prüfen, wirkt sich auch auf einen Man-in-the-Middle-Angriff aus. Der Angriff kann nur durchgeführt werden, solange kein Diffie-Hellman-Schlüsselaustausch stattgefunden hat. Nachdem der Diffie-Hellman-Schlüsselaustausch durchgeführt wurde, werden die Daten verschlüsselt, bevor sie übertragen werden. Dadurch ist ein Angreifer nicht mehr in der Lage die abgefangenen Daten sinnvoll zu manipulieren. Wie in Abschnitt 4.4.3.3 beschrieben, muss bei einem solchen Angriff die Verbindung gestört werden, damit ein Angreifer die gesendeten Daten manipulieren kann, bevor er sie an den eigentlichen Empfänger sendet. Da die Störung nicht erkannt wird, liegt es an den Nutzern einen solchen Angriff zu bemerken. Wie in Abschnitt 4.4.3.3 beschrieben, warten sowohl der Sender als auch der Empfänger auf Daten. Erhalten beide Nutzer gleichzeitig Daten, stammen diese von einem Angreifer und der Schlüsselaustausch muss abgebrochen werden.

Sofern die NFC-Unterstützung in Zukunft um eine Funktionalität erweitert wird, mit der sich das elektromagnetische Feld prüfen lässt, sollte die Anwendung angepasst werden, um die Sicherheit weiter zu erhöhen. Die Schwäche, der umgesetzten Sicherheitsmaßnahmen liegt darin, dass während dem Diffie-Hellman-Schlüsselaustausch ein Man-in-the-Middle-Angriff erfolgen kann, den der Nutzer erkennen muss. Ausgehend davon, dass die Nutzer diesen Angriff bemerken, sorgen die realisierten Sicherheitsmaßnahmen dafür, dass die ausgetauschten kryptographischen Schlüssel nicht von einem Angreifer abgefangen werden können. Gleichermaßen ist es nicht möglich die kryptographischen Schlüssel während der Übertragung gezielt zu manipulieren. Es kann jedoch nicht verhindert werden, dass die Übertragung gestört und die kryptographischen Schlüssel unbrauchbar werden. Die implementierten Sicherheitsmaßnahmen bieten einen ausreichenden Schutz gegen die identifizierten Risiken. In Zukunft sollte die Sicherheit der Anwendung in einer weiteren unabhängigen Arbeit geprüft werden.

Kapitel 7

Weiterentwicklung der Arbeit

In diesem Kapitel wird beschrieben, wie die Anwendung in Zukunft für verschiedene Anwendungsfälle erweitert werden kann. Abschließend wird die Arbeit zusammengefasst und ein Fazit gezogen.

7.1 Zukünftige Erweiterungen

Im Folgenden werden allgemeine Anwendungsfälle aufgeführt, für die die Anwendung verwendet werden kann. Einer dieser Fälle wird beispielhaft weiter ausgeführt.

7.1.1 Allgemeine Anwendungsfälle

Die Kernfunktionalität der Anwendung zielt darauf ab, Daten zu verwalten und sicher auszutauschen. Dadurch, dass die Daten vor der Datenübertragung verschlüsselt werden, kann ein Angreifer die Daten nicht abfangen und gezielt manipulieren. Die Anwendung eignet sich also allgemein für Anwendungsfälle, in denen sensible Daten zwischen zwei Personen ausgetauscht werden sollen. Da es sich um eine Smartphone-Anwendung handelt, sind nur diejenigen Anwendungsfälle relevant, bei denen Daten im persönlichen Kontakt ausgetauscht werden.

Anwendungsfälle sind beispielsweise mobiles Bezahlen und mHealth-Anwendungen. Wenn eine Anwendung zum mobilen Bezahlen eingesetzt wird, ist es wichtig, dass die Datenübertragung nicht manipuliert werden kann. Ansons-

ten könnte ein Angreifer zum Beispiel den zu zahlenden Betrag verändern. Die entwickelte Anwendung stellt sicher, dass keine gezielte Manipulation stattfinden kann. Eine Anwendung im mHealth-Bereich könnte zum Beispiel eingesetzt werden, um medizinische Daten zwischen einem Patienten und seinem Arzt auszutauschen. Bei diesen Daten könnte es sich um die Krankheitshistorie des Patienten oder medizinische Daten handeln. Die ausgetauschten Daten sind in solch einem Fall sehr sensibel und es ist nicht gewünscht, dass jemand anders außer dem Arzt und dem Patienten auf die Daten zugreift. Dadurch, dass die Daten vor der Übertragung verschlüsselt werden, wird ausgeschlossen, dass sie von jemanden mitgelesen werden.

Diese Anwendungsfälle zeigen, dass sich die Anwendung auch für andere Szenarien als den Austausch kryptographischer Schlüssel einsetzen lässt. Nachfolgend wird näher beschrieben wie die Anwendung eingesetzt werden kann, um kryptographische Schlüssel zu signieren. Der beschriebene Anwendungsfall passt damit thematisch zu dem in dieser Arbeit umgesetzten Austausch von kryptographischen Schlüsseln.

7.1.2 Beispiel: Keysigning-Party

Aufbauend auf dieser Arbeit kann eine Anwendung realisiert werden, mit der sich öffentliche Schlüssel signieren lassen. Dies spielt im Zusammenhang mit PGP oder vergleichbaren Diensten eine große Rolle. Nachfolgend wird kurz das Konzept von PGP erklärt.

7.1.2.1 Pretty Good Privacy

Pretty Good Privacy (PGP) ist eine von Philip Zimmermann entwickelte Software, um Daten zu verschlüsseln und zu signieren. Die nachfolgenden Informationen stammen aus Zimmermanns „PGP User's Guide, Volume I“ [Zim94]. Typischerweise wird die Software zur E-Mail-Verschlüsselung eingesetzt. Jeder Nutzer besitzt ein Schlüsselpaar bestehend aus einem privaten und einem öffentlichen Schlüssel. Nachrichten, die an einen Nutzer gesendet werden, müssen zuvor mit dessen öffentlichen Schlüssel verschlüsselt werden. Mit dem dazugehörigen privaten Schlüssel kann die Nachricht dann anschließend entschlüsselt werden. In einem öffentlichen Schlüssel ist die Information enthalten, zu welcher

Person dieser öffentliche Schlüssel gehört. Außerdem existiert zu jedem öffentlichen Schlüssel ein Hashwert, der sogenannte Fingerprint.

Um sicherzustellen, dass ein öffentlicher Schlüssel tatsächlich zu der Person gehört, der er zu geordnet ist, muss er von anderen Personen signiert werden. Mit einer Signatur bestätigt eine Person, dass der öffentliche Schlüssel tatsächlich zu dem angegebenen Besitzer gehört. Je häufiger ein öffentlicher Schlüssel durch eine Signatur bestätigt wird, umso größer ist das Vertrauen in seine Echtheit. Damit das System funktioniert, sollten öffentliche Schlüssel möglichst oft signiert werden. Aus diesem Grund werden Keysigning-Partys organisiert, auf denen Personen ihre öffentlichen Schlüssel gegenseitig signieren. Der Ablauf einer solchen Veranstaltung wird im nächsten Abschnitt beschrieben.

7.1.2.2 Ablauf von Keysigning-Partys

Die nachfolgende Beschreibung von Keysigning-Partys orientiert sich an dem Dokument „GnuPG Keysigning-Party HOWTO“ [Bre03]. Es wird zwischen informellen und strukturierten Keysigning-Partys unterschieden. Informelle Veranstaltungen erfordern keinen besonderen Organisationsaufwand. Jeder Teilnehmer ist dafür verantwortlich einen Zettel mitzubringen, auf dem der Fingerprint seines öffentlichen Schlüssels aufgedruckt ist und einen Schlüsselserver, von dem der öffentliche Schlüssel bezogen werden kann. Schlüsselserver enthalten Sammlungen von öffentlichen Schlüssel und werden eingesetzt, um diese zu verbreiten. Die Teilnehmer tauschen untereinander ihre Zettel aus und weisen dabei ihre Identität nach. Die anderen Teilnehmer wissen dadurch, dass der öffentliche Schlüssel, der mit dem aufgedruckten Fingerprint übereinstimmt tatsächlich zu dieser Person gehört. Damit ist die Keysigning-Party abgeschlossen.

Um größere Keysigning-Partys strukturierter durchzuführen, kann ein Veranstalter im Voraus die Fingerprints der öffentlichen Schlüssel der Teilnehmer anfordern und diese auf einer Liste zusammenstellen, zusätzlich können Schlüsselserver angegeben werden. Die Liste wird zu Beginn der Veranstaltung an alle Teilnehmer ausgeteilt. Jeder Teilnehmer bestätigt dann, dass der aufgedruckte Fingerprint tatsächlich zu dem eigenen öffentlichen Schlüssel gehört. Der Veranstalter kann dann eine Reihenfolge festlegen, in der die Teilnehmer ihre Identität nachweisen. Anschließend ist die Veranstaltung beendet. Die Teilnehmer können im Anschluss an die Keysigning-Party die öffentlichen Schlüssel von den angege-

benen Servern beziehen, mit ihrer Software signieren und sie anschließend wieder dem Server mitteilen.

7.1.2.3 Einsatz der Anwendung

Beide Organisationsformen sind mit einem Aufwand für die Teilnehmer verbunden. Bei einer informellen Veranstaltung müssen im voraus Zettel erstellt werden, auf denen der Fingerprint und die Adresse des Schlüsselserver aufgedruckt sind. Wird zuvor eine Liste erstellt, müssen die Teilnehmer die Informationen dem Veranstalter mitteilen und auf der Veranstaltung selbst nachprüfen, ob die Informationen auf der Liste korrekt sind. Nur so kann ein Fehler oder eine Manipulation des Veranstalters erkannt werden. Außerdem müssen die Teilnehmer im Anschluss den öffentlichen Schlüssel anhand seines Fingerprints von einem Schlüsselserver beziehen und signieren.

Eine Anwendung, die auf dem in dieser Arbeit vorgestellten Konzept basiert, kann den Aufwand für die Teilnehmer reduzieren und bestimmte Schritte automatisieren. Die Anwendung muss derart angepasst werden, dass ein Nutzer den Fingerprint seines öffentlichen Schlüssel hinterlegen kann. Wenn sich zwei Personen treffen, können sie gemäß des beschriebenen Datenaustauschs ihre jeweiligen Fingerprints austauschen. Dadurch, dass die Daten vor der Übertragung verschlüsselt werden, können sie nicht gezielt von einem Angreifer manipuliert werden. Der Fingerprint muss also nicht mehr manuell überprüft werden. Nachdem die Fingerprints ausgetauscht wurden, kann die Anwendung den zugehörigen Schlüssel automatisch von einem Schlüsselserver beziehen und signieren. Der gesamte Vorgang wird dadurch automatisiert. Der Nutzer muss lediglich seinen Fingerprint übertragen. Falls die Nutzer ihre öffentlichen Schlüssel direkt austauschen wollen ohne dabei einen Schlüsselserver zu nutzen, ist dies ebenfalls möglich. In diesem Fall wird der öffentliche Schlüssel übertragen, signiert und dann wieder an den ursprünglichen Besitzer gesendet.

Eine solche Anwendung kann den überwiegenden Teil der bisher implementierten Methoden nutzen. Die Methoden, um öffentlichen Schlüsseln zu signieren müssen allerdings neu implementiert werden. Eine Anwendung zum Signieren von öffentlichen Schlüssel kann nicht nur auf Keysigning-Partys genutzt werden, sondern eignet sich auch dafür öffentliche Schlüssel spontan zu signieren. Es ist nicht mehr notwendig Zettel mit den notwendigen Informationen mitzu-

führen, da der gesamte Vorgang mit dem Smartphone abgewickelt werden kann. Die Anwendung kann gleichzeitig für die Verwaltung von öffentlichen Schlüsseln genutzt werden.

7.2 Zusammenfassung der Arbeit

Am Anfang der Arbeit stand die Frage, wie ein Konzept aussehen muss, mit dem sich kryptographische Schlüssel sicher zwischen zwei Personen im direkten Kontakt austauschen lassen. Das Konzept sollte gewährleisten, dass die ausgetauschten Schlüssel tatsächlich zu der jeweils anderen Person gehören und nicht manipuliert worden sind. Ausgehend von dieser Fragestellung wurde ein Konzept für den Schlüsselaustausch entwickelt. Das Konzept sieht vor, dass der Schlüsselaustausch im direkten Kontakt erfolgt. Gleichzeitig wird verhindert, dass die kryptographischen Schlüssel abgefangen und manipuliert werden können.

Aufbauend auf dem Konzept wurde eine mobile Anwendung entworfen. Sie kann zum Verwalten und Austauschen von kryptographischen Schlüssel genutzt werden. Die Übertragung erfolgt mittels der Near Field Communication. Der Entwurf erfolgte in zwei Schritten. Zuerst wurde ein komponentenbasiertes Framework entworfen, das einen sicheren Datenaustausch im Allgemeinen erlaubt. Das Framework ist nicht auf den mobilen Bereich oder den Austausch von kryptographischen Schlüsseln festgelegt. Es definiert vier grundlegende Komponenten, um Daten zu verwalten und sicher auszutauschen. Die Komponenten und ihre Schnittstellen sind in vier Spezifikationen beschrieben. Zusätzlich wurde beschrieben wie die Komponenten zusammenarbeiten um den Datenaustausch zu realisieren. Der Entwurf des Frameworks kann zukünftig als Grundlage für andere Anwendungen genutzt werden. Um die geforderten Funktionalitäten der Anwendung zu realisieren, wurde der Entwurf des Frameworks entsprechend erweitert. Im Fokus stand dabei der Austausch kryptographischer Schlüssel im persönlichen Kontakt über Smartphones.

Im Anschluss an den Entwurf wurde eine Sicherheitsanalyse durchgeführt, um Angriffe zu identifizieren und zu behandeln. Da die Datenübertragung mittels NFC erfolgen sollte, standen mögliche Angriffe auf diese Funktechnik im Vordergrund. Aus der Sicherheitsanalyse ergab sich, dass die kryptographischen Schlüssel vor der Übertragung verschlüsselt werden müssen. Basierend auf dem Entwurf und der vorgenommenen Sicherheitsanalyse wurde eine Anwendung

für das Android-Betriebssystem entwickelt. Während der Entwicklung hat sich gezeigt, dass Anpassungen gegenüber dem Entwurf nötig waren. Die notwendigen Anpassungen ergaben sich aus der eingeschränkten NFC-Unterstützung des Betriebssystems. Die umgesetzte Anwendung realisiert das anfangs entwickelte Konzept. Abschließend wurden Anwendungsfälle beschrieben, für die sich aufbauend auf dieser Arbeit, Anwendungen entwickeln lassen. Im Fokus stand eine Anwendung mit der sich kryptographische Schlüssel signieren lassen. Dies stellt eine sinnvolle Erweiterung der Anwendung dar.

7.3 Fazit

Das in der Arbeit entwickelte Konzept eignet sich für den Austausch kryptographischer Schlüssel im persönlichen Kontakt mittels zweier Smartphones. Das spezifizierte Framework kann eingesetzt werden um Daten allgemein sicher auszutauschen und ist nicht auf den mobilen Bereich beschränkt. Es kann zukünftig auch in anderen Arbeiten verwendet werden, die keinen Bezug zu der entwickelten Anwendung oder der Fragestellung dieser Arbeit haben. Die auf dem Framework basierende Anwendung, welche das beschriebene Konzept realisiert, bietet die nötigen Funktionalitäten um kryptographische Schlüssel zu verwalten und zu versenden. Momentan können mit der Anwendung RSA-Schlüssel erstellt, bearbeitet und gelöscht werden. Die gespeicherten Schlüsseln können für andere Anwendungen bereitgestellt werden, die ebenfalls auf dem Smartphone installiert sind. Der Export und Import von kryptographischen Schlüssel geht über die anfängliche Zielsetzung hinaus und wurde deshalb nicht realisiert. Grundsätzlich sind solche Funktionalitäten aber sinnvoll, damit kryptographische Schlüssel die beispielsweise auf einem Desktop oder Laptop vorliegen in die Anwendung aufgenommen werden können. Ebenso ist es sinnvoll in der Anwendung gespeicherte kryptographische Schlüssel auf einen Desktop oder Laptop überführen zu können.

Aufgrund der eingeschränkten NFC-Unterstützung von Android ist der Datenaustausch zum aktuellen Zeitpunkt aufwendiger als ursprünglich angedacht. Außerdem ist es wichtig, dass die Nutzer während sie Daten austauschen, auf einen möglichen Man-in-the-Middle-Angriff achten, da dieser aufgrund der eingeschränkten NFC-Unterstützung bisher nicht von der Anwendung erkannt werden kann. In zukünftigen Arbeiten sind Usability- und Sicherheitstests durch-

zuführen um die Anwendung weiter zu verbessern. Aktuell ist die entwickelte Anwendung ein erweiterbares und wiederverwendbares Grundgerüst, auf dem andere Anwendungen aufbauen können.

Literaturverzeichnis

[AA92] Esther Amann and Hugo Atzmüller. It-sicherheit - was ist das? In *Datenschutz und Datensicherung (DuD) 6/92*. Vieweg Verlag, 1992.

[Anda] Nfc basics. http://developer.android.com/guide/topics/connectivity/nfc/nfc.html, Zuletzt abgerufen am 25.08.2013.

[Andb] Tasks and back stack. http://developer.android.com/guide/components/tasks-and-back-stack.html, Zuletzt abgerufen am 25.08.2013.

[Art12] Charles Arthur. iphone 5 shows that apple still considers nfc as not for commerce, September 2012. http://www.guardian.co.uk/technology/2012/sep/14/apple-iphone-5-near-field-communication-nfc, Zuletzt abgerufen am 25.08.2013.

[BIT13a] BITKOM. Presseinformation: It-unternehmen setzen auf verschlüsselung, Juli 2013. https://bitkom.org/files/documents/BITKOM-Presseinfo_Verschluesselung_in_Unternehmen_01_08_2013.pdf, Zuletzt abgerufen am 25.08.2013.

[BIT13b] BITKOM. Sicherheit und vertrauen im netz, Juli 2013. http://www.bitkom.org/files/documents/BITKOM_PK_Sicherheit_im_Netz_Charts_25_07_2013.pdf, Zuletzt abgerufen am 25.08.2013.

[BP10] Arno Becker and Marcus Pant. *Android 2: Grundlagen und Programmierung*. dpunkt.verlag, 2010.

[Bre03] V. Alex Brennen. Gnupg keysigning-party howto, 2003. http://rhonda.deb.at/projects/gpg-party/gpg-party.de.html, Zuletzt abgerufen am 25.08.2013.

[Buc08] Johannes Buchmann. *Einführung in die Kryptographie*. Springer-Verlag, 2008.

[CBB+10] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond, Second Edition*. Addison-Wesley Professional, 2010.

[com13] comScore. comscore reports april 2013 u.s. smartphone subscriber market share, Juni 2013. http://www.comscore.com/Insights/Press_Releases/2013/6/comScore_Reports_April_2013_U.S._Smartphone_Subscriber_Market_Share, Zuletzt abgerufen am 25.08.2013.

[Eck06] Claudia Eckert. *IT-Sicherheit - Konzepte, Verfahren, Protokolle*. Oldenbourg Wissenschaftsverlag, 2006.

[ECM04] Standard ecma-340 - near field communication interface and protocol (nfcip-1), 2004.

[ECM10] Standard ecma-352 - near field communication interface and protocol - 2 (nfcip-2), 2010.

[FFSB04] Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. *Head First Design Patterns*. O'Reilly Media, 2004.

[Fin10] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley & Sons, 2010.

[For06] NFC Forum. Nfc data exchange format (ndef). Technical report, NFC Forum, 2006.

[GBK11] Rüdiger Grimm, Katharina Bräunlich, and Andreas Kasten. Referenzmodell für ein vorgehen bei der it-sicherheitsanalyse - eine taxonomie der grundlegenden. Bisher unveröffentlichter Forschungsbericht der Universität Koblenz-Landau von 2011, 2011.

[Gin] Gingerbread overview. http://developer.android.com/about/versions/android-2.3-highlights.html, Zuletzt abgerufen am 25.08.2013.

[HB06] Ernst Haselsteiner and Klemens Breitfuß. Security in near field communication (nfc) - strengths and weaknesses. In *Dominkus S (Ed) Workshop on RFID Security*, 2006.

[HKRS08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web Grundlagen*. Springer-Verlag, 2008.

[ICS] Ice cream sandwich overview. http://developer.android.com/about/versions/android-4.0-highlights.html, Zuletzt abgerufen am 25.08.2013.

[IDC] Idc - press release - smartphones expected to outship feature phones for first time in 2013. http://www.idc.com/getdoc.jsp?containerId=prUS23982813, Zuletzt abgerufen am 25.08.2013.

[ISO08] Identification cards - contactless integrated circuit cards - proximity cards, iso/iec 14443, 2008.

[ISO10] Identification cards - contactless integrated circuit cards - vicinity cards, iso/iec 15693, 2010.

[ISO12] Information technology - telecommunications and information exchange between systems - near field communication - interface and protocol - 2 (nfcip-2), iso/iec 21481, 2012.

[ISO13] Information technology - telecommunications and information exchange between systems - near field communication - interface and protocol (nfcip-1), iso/iec 18092, 2013.

[JF88] Ralph E. Johnson and Brian Foote. Designing reusable classes. *Journal of Object - Oriented Programming*, 1:22–35, 1988.

[Kno12] Knotice. Mobile email opens report. Technical report, Knotice, 2012.

[LK08] M. Lepinski and S. Kent. Rfc 5114: Additional diffie-hellman groups for use with ietf standards, Januar 2008. http://tools.ietf.org/html/rfc5114, Zuletzt abgerufen am 25.08.2013.

[LR10] Jossef Langer and Michael Roland. *Anwendung und Technik von Near Field Communication (NFC)*. Springer-Verlag, 2010.

[OMG04] Unified modeling language: Superstructure - version 2.0, 2004.

[Rah11] Mahbub Rahman. Understanding nfc data exchange format (ndef) messages, 2011. http://www.developer.nokia.com/Community/Wiki/Understanding_NFC_Data_Exchange_Format_%28NDEF%29_messages, Zuletzt abgerufen am 25.08.2013.

[RH06] Ralf Reussner and Wilhelm Hasselbring. *Handbuch der Software-Architektur*. dpunkt.verlag, 2006.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[Zim94] Philip Zimmermann. Pgp user's guide, volume i: Essential topics, 1994. http://www.pa.msu.edu/reference/pgpdoc1.html, Zuletzt abgerufen am 25.08.2013.

Anhang A

Spezifikation des Frameworks

UNIKO – Technical Guideline

Name:	SecureExchange-Framework
Identifier:	UNIKO TR-xxxxx-x
Version:	0.0.1
Date:	2013-08-26

1 Overview of the SecureExchange-API-Framework

The *SecureExchange-Framework* provides functionality to manage data and to transfer this data through a secured communication channel. The described functionality is realized by four components, as visualized in Figure 1, which provide their functions through interfaces. An implementation has to provide instances of these components which offer the specified interfaces.

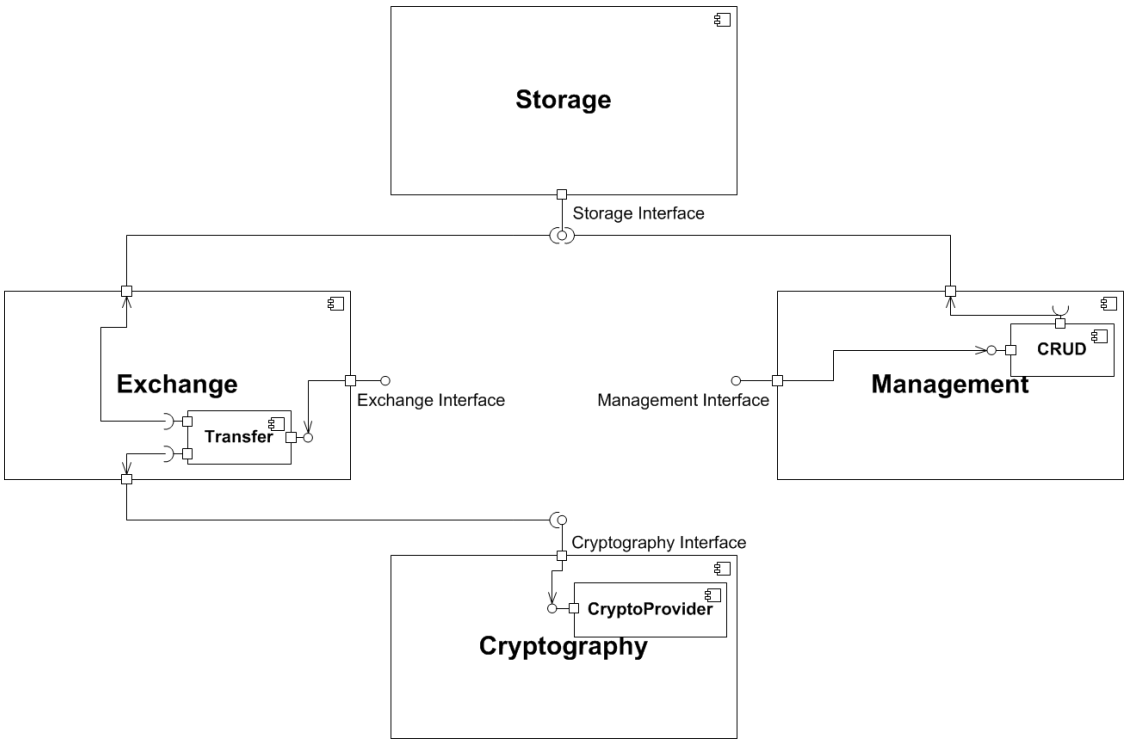


Figure 1 Components of the Framework

The framework consists of the following components:

- **Storage**
  The *Storage-Interface* provides functionality to save and manage data. This includes adding, removing, and returning data.
- **Management**
  The *Management-Interface* provides functionality to create, save, update and delete data.
- **Cryptography**
  The *Cryptography-Interface* provides functionality to secure a communication channel.
- **Exchange**
  The *Exchange-Interface* provides functionality to establish a channel between communication parties, which can be used to securely transmit data.

2 Overview of the Storage-Interface

2.1 Objective

The *Storage-Interface* provides functionality to save and manage data. This includes adding, removing and returning data.

2.2 Functions

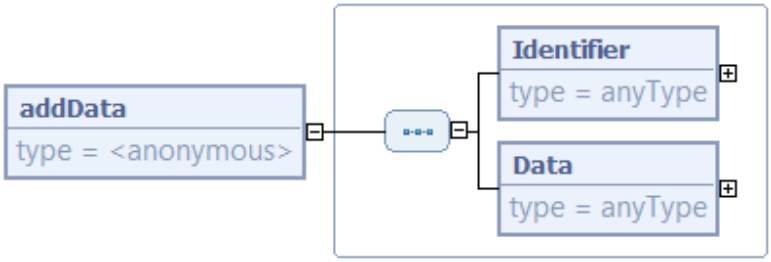
The *Storage-Interface* provides the following function group:

- **Function Group Storage**
  - `addData`
    The `addData` function is invoked to save data.
  - `removeData`
    The `removeData` function is invoked to remove data.
  - `getData`
    The `getData` function is invoked to return data.

3 Specification of the Storage-Interface

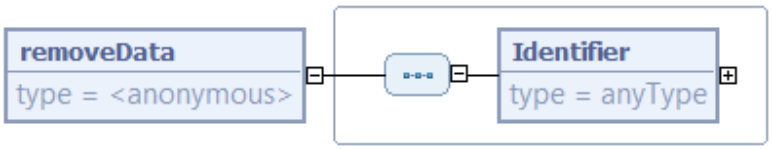
3.1 Function Group Storage

3.1.1 addData

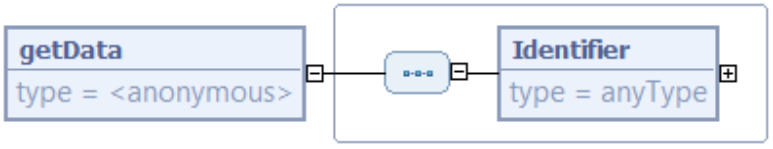
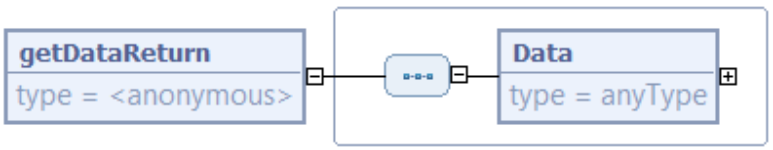
<table>
<tr><td>Name</td><td colspan="2"><code>addData</code></td></tr>
<tr><td>Description</td><td colspan="2">The <code>addData</code> function associates the given data with the given identifier and saves it.</td></tr>
<tr><td>Invocation parameters</td><td colspan="2"> 
</td></tr>
<tr><td></td><td>Name</td><td>Description</td></tr>
<tr><td></td><td><code>Identifier</code></td><td>The identifier which should be associated with the given data.</td></tr>
<tr><td></td><td><code>Data</code></td><td>The data which should be saved.</td></tr>
<tr><td>Return parameter</td><td colspan="2"></td></tr>
<tr><td>Precondition</td><td colspan="2">No data associated with the given identifier is currently stored.</td></tr>
<tr><td>Postcondition</td><td colspan="2">The given data has been associated with the given identifier and has been saved.</td></tr>
<tr><td>Note</td><td colspan="2">General information:
The structure of the <code>Identifier</code> and <code>Data</code> objects depends on the application which implements this framework.

Specific information for this function:
An implementation of the Storage-Component has to use a mechanism to persistently save data, e.g., the data could be stored in a database or in a XML file. This method encapsulates the code which is needed to save data according to the used mechanism.</td></tr>
</table>

3.1.2 removeData

Name	`removeData`	
Description	The `removeData` function removes the data associated with the given identifier.	
Invocation parameters		
	Name	**Description**
	`Identifier`	The identifier which is associated with the data that should be removed.
Return parameter		
Precondition	Data associated with the given identifier is currently stored.	
Postcondition	The data which was associated with the given Identifier has been removed.	
Note	General information: The structure of the `Identifier` object depends on the application which implements this framework. Specific information for this function: An implementation of the *Storage-Component* has to use a mechanism to persistently save data, e.g., the data could be stored in a database or in a XML file. This method encapsulates the code which is needed to remove data according to the used mechanism.	

3.1.3 getData

Name	`getData`	
Description	The `getData` functions returns data.	
Invocation parameters		
	Name	**Description**
	`Identifier`	The identifier which is associated with the data that should be returned.
Return parameter		

	Name	**Description**
	`Data`	The data associated with the given Identifier.
Precondition	Data associated with the given identifier is currently stored.	
Postcondition	The data which was associated with the given identifier has been returned.	
Note	General information: The structure of the `Identifier` and `Data` objects depends on the application which implements this framework. Specific information for this function: An implementation of the *Storage-Component* has to use a mechanism to persistently save data, e.g., the data could be stored in a database or in a XML file. This method encapsulates the code which is needed to return data according to the used mechanism.	

4 Overview of the Management-Interface

4.1 Objective

The *Management-Interface* provides functionality to create, save, update and delete data.

4.2 Functions

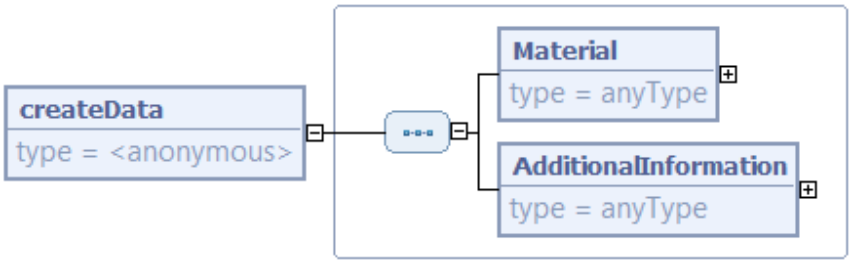
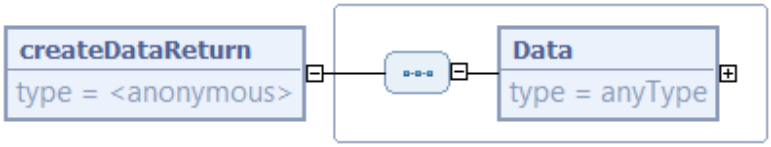
The *Management-Interface* provides the following function group:

- **Function Group CRUD (acronym for Create, Read, Update, Delete)**
  - **createData**
    The `createData` function is invoked to create data.
  - **saveData**
    The `saveData` function is invoked to save data.
  - **updateData**
    The `updateData` function is invoked to update data.
  - **deleteData**
    The `deleteData` function is invoked to delete data.

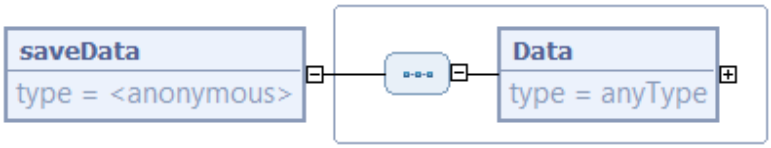
5 Specification of the Management-Interface

5.1 Function Group CRUD

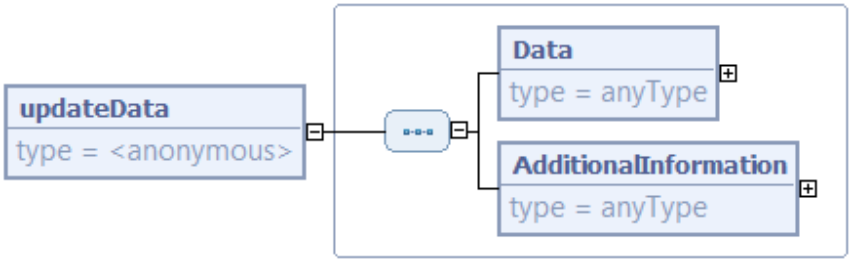
5.1.1 createData

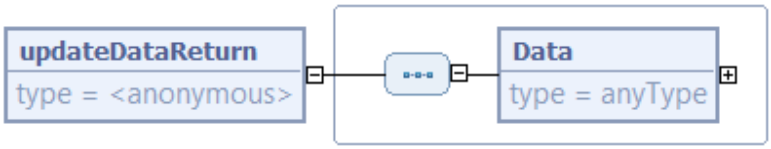
<table>
<tr><td>Name</td><td colspan="2"><code>createData</code></td></tr>
<tr><td>Description</td><td colspan="2">The <code>createData</code> function creates data from the given material and incorporates the given additional information.</td></tr>
<tr><td>Invocation parameters</td><td colspan="2"> 
</td></tr>
<tr><td></td><td>Name</td><td>Description</td></tr>
<tr><td></td><td><code>Material</code></td><td>The sum of basic information from which the data should be generated.</td></tr>
<tr><td></td><td><code>Additional Information</code></td><td>Additional information that should be included in the data to provide further information.</td></tr>
<tr><td>Return parameter</td><td colspan="2"> 
</td></tr>
<tr><td></td><td>Name</td><td>Description</td></tr>
<tr><td></td><td><code>Data</code></td><td>The data which was generated from the given material and the additional information.</td></tr>
<tr><td>Precondition</td><td colspan="2">The given invocation parameters must not be <code>null</code>.</td></tr>
<tr><td>Postcondition</td><td colspan="2">The data has been generated from the given invocation parameters.</td></tr>
<tr><td>Note</td><td colspan="2">General information:
The structure of the <code>Material</code> and <code>AdditionalInformation</code> objects depends on the application which implements this framework. It is advised to include information in the <code>AdditionalInformation</code> object to identify the creator of the <code>Data</code> object. Otherwise it is not possible to reconstruct after a data exchange from whom data was received.</td></tr>
</table>

5.1.2 saveData

Name	`saveData`	
Description	The `saveData` function saves data.	
Invocation parameters		
	Name	**Description**
	`Data`	The data which should be saved.
Return parameter		
Precondition	The given invocation parameter must not be `null`.	
Postcondition	The given data has been saved.	
Note	General information: The structure of the `Data` object depends on the application which implements this framework. Specific information for this function: This function uses the `addData` function provided by the *Storage-Component*. It abstracts from the function of the *Storage-Component*, which only works with the mechanism used to save data. Additional functionality can be implemented in this function, e.g. creating a log for saved data.	

5.1.3 updateData

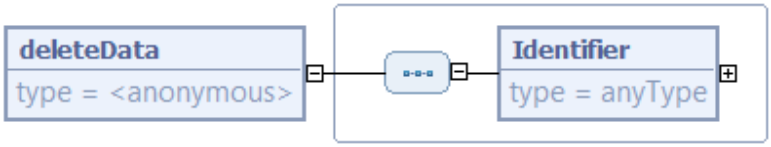
Name	`updateData`	
Description	The `updateData` function updates the additional information of the given data.	
Invocation parameters		
	Name	**Description**
	`Data`	The data whose additional information should be updated.
	`Additional Information`	Additional information that should be included in the data to provide further information.

<table>
<tr><td>Return parameter</td><td> 

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Data</td><td>The data which was updated with the given additional information.</td></tr>
</table></td></tr>
<tr><td>Precondition</td><td>The data must be currently stored.</td></tr>
<tr><td>Postcondition</td><td>The data has been updated with the given additional information.</td></tr>
<tr><td>Note</td><td><u>General information:</u>
The structure of the Material and AdditionalInformation objects depends on the application which implements this framework. It is advised to include information in the AdditionalInformation object to identify the creator of the Data object. Otherwise it is not possible to reconstruct after a data exchange from whom data was received.

<u>Specific information for this function:</u>
This function uses the getData function provided by the Storage-Component. The sole purpose of this function is to update the given Data object; therefore the saveData function has to be invoked if this updated data should be saved afterwards.</td></tr>
</table>

5.1.4 deleteData

<table>
<tr><td>Name</td><td>deleteData</td></tr>
<tr><td>Description</td><td>The deleteData function deletes data.</td></tr>
<tr><td>Invocation parameters</td><td> 

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Identifier</td><td>The identifier which is associated with the data that should be deleted.</td></tr>
</table></td></tr>
<tr><td>Return parameter</td><td></td></tr>
<tr><td>Precondition</td><td>Data associated with the given identifier is currently stored.</td></tr>
<tr><td>Postcondition</td><td>The data which was associated with the given identifier has been deleted.</td></tr>
</table>

Note	General information: The structure of the `Identifer` object depends on the application which implements this framework. Specific information for this function: This function uses the `removeData` function provided by the *Storage-Component*. It abstracts from the function of the *Storage-Component*, which only works with the mechanism used to save data. Additional functionality can be implemented in this function, e.g. creating a log for deleted data.

6 Overview of the Cryptography-Interface

6.1 Objective

The *Cryptography-Interface* provides functionality to secure a communication channel.

6.2 Functions

The *Cryptography-Interface* provides the following function group:

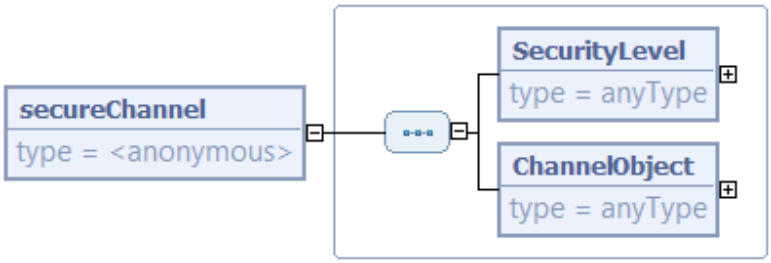
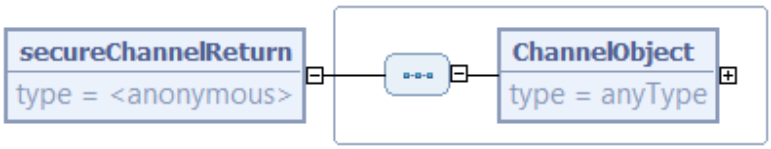
- **Function Group SecurityProvider**
  - **secureChannel**

    The `secureChannel` function is invoked to secure a communication channel.

7 Specification of the Cryptography-Interface

7.1 Function Group SecurityProvider

7.1.1 secureChannel

<table>
<tr><td>Name</td><td colspan="2">secureChannel</td></tr>
<tr><td>Description</td><td colspan="2">The secureChannel function secures an already established plain communication channel.</td></tr>
<tr><td>Invocation parameters</td><td colspan="2"> 
</td></tr>
<tr><td></td><td>Name</td><td>Description</td></tr>
<tr><td></td><td>SecurityLevel</td><td>The chosen level of security for the channel.</td></tr>
<tr><td></td><td>ChannelObject</td><td>All information necessary to use the plain communication channel.</td></tr>
<tr><td>Return parameter</td><td colspan="2"> 
</td></tr>
<tr><td></td><td>Name</td><td>Description</td></tr>
<tr><td></td><td>ChannelObject</td><td>All information necessary to use the secured communication channel.</td></tr>
<tr><td>Precondition</td><td colspan="2">A plain communication channel is established between the communication parties.</td></tr>
<tr><td>Postcondition</td><td colspan="2">A secure communication channel has been established between the communication parties according to the given parameters.</td></tr>
<tr><td>Note</td><td colspan="2">General information:
The structure of the SecurityLevel and ChannelObject objects depends on the application which implements this framework.

Specific information for this function:
This function should provide different means to secure an established communication channel.</td></tr>
</table>

8 Overview of the Exchange-Interface

8.1 Objective

The *Exchange-Interface* provides functionality to establish a channel between communication parties, which can be used to securely transmit data.

8.2 Functions

The *Exchange-Interface* provides the following function group:

- **Function Group ConnectionHandling**
  - **establishChannel**

    The `establishChannel` function is invoked to establish a channel between communication parties.
  - **dissolveChannel**

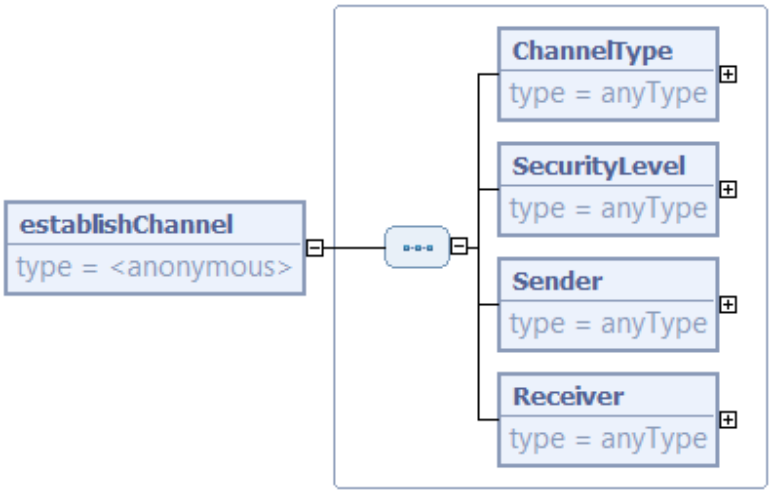
    The `dissolveChannel` function is invoked to dissolve an established channel.
- **Function Group TransmissionHandling**
  - **transmit**

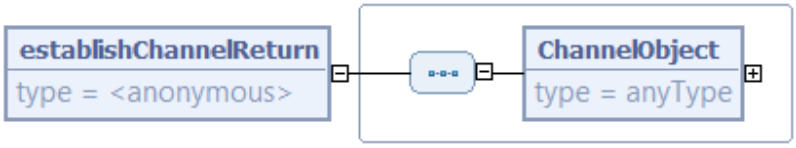
    The `transmit` function is invoked to transmit data over a communication channel.

9 Specification of the Exchange-Interface

9.1 Function Group ConnectionHandling

9.1.1 establishChannel

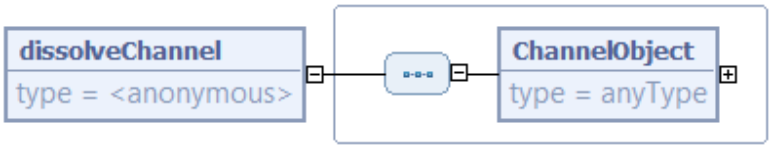
<table>
<tr><td>Name</td><td colspan="2">establishChannel</td></tr>
<tr><td>Description</td><td colspan="2">The establishChannel function establishes a communication channel between parties.</td></tr>
<tr><td>Invocation parameters</td><td colspan="2"> 

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ChannelType</td><td>The type of channel (plain or secure).</td></tr>
<tr><td>SecurityLevel</td><td>The chosen level of security for the communication channel. (only applies if the ChannelType is plain)</td></tr>
<tr><td>Sender</td><td>The sending party.</td></tr>
<tr><td>Receiver</td><td>The receiving party.</td></tr>
</table></td></tr>
<tr><td>Return parameter</td><td colspan="2"> 

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ChannelObject</td><td>All information necessary to use the established communication channel.</td></tr>
</table></td></tr>
<tr><td>Precondition</td><td colspan="2">The parties are not already connected through a communication channel.</td></tr>
<tr><td>Postcondition</td><td colspan="2">A communication channel has been established between the parties according to the given invocation parameters and the ChannelObject has been returned.</td></tr>
</table>

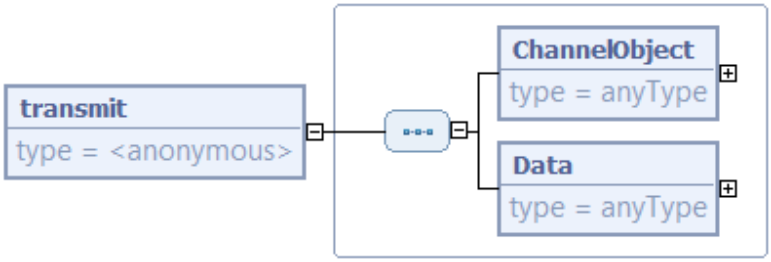
Note	General information: The structure of the `ChannelType`, `SecurityLevel`, `Sender`, `Receiver` and `ChannelObject` objects depends on the application which implements this framework. Specific information for this function: This function uses the `secureConnection` function from the *Cryptography-Component* to secure the communication channel.

9.1.2 dissolveChannel

Name	`dissolveChannel`	
Description	The `dissolveConncection` dissolves an established communication channel.	
Invocation parameters		
	Name	**Description**
	`ChannelObject`	All information necessary to use the established communication channel.
Return parameter		
Precondition	No transmission is in progress and no further transmissions are expected.	
Postcondition	The communication channel has been dissolved.	
Note	General information: The structure of the `ChannelType`, `SecurityLevel`, `Sender`, `Receiver` and `ChannelObject` objects depends on the application which implements this framework.	

9.2 Function Group TransmissionHandling

9.2.1 transmit

Name	`transmit`
Description	The `transmit` function transmits data over a communication channel.
Invocation parameters	

	Name	**Description**
	`ChannelObject`	All information necessary to use the established communication channel.
	`Data`	The data which should be transmitted from the sending to the receiving party.
Return parameter		
Precondition	A communication channel is established between the communication parties.	
Postcondition	The Data has been received by the receiving party.	
Note	General information: The structure of the `ChannelType`, `SecurityLevel`, `Sender`, `Receiver` and `ChannelObject` objects depends on the application which implements this framework.	

Anhang B

Schritte der Datenübertragung

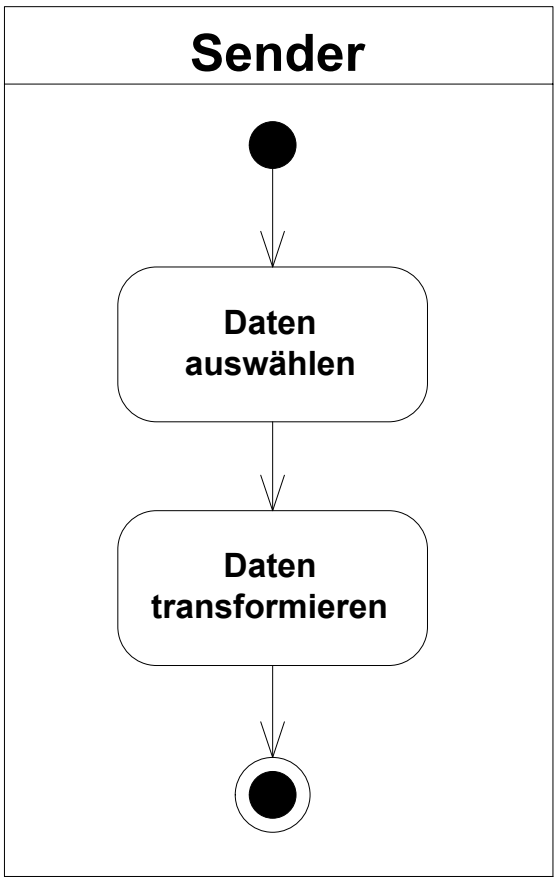


Abbildung B.1: Schritte: Vorbereitung

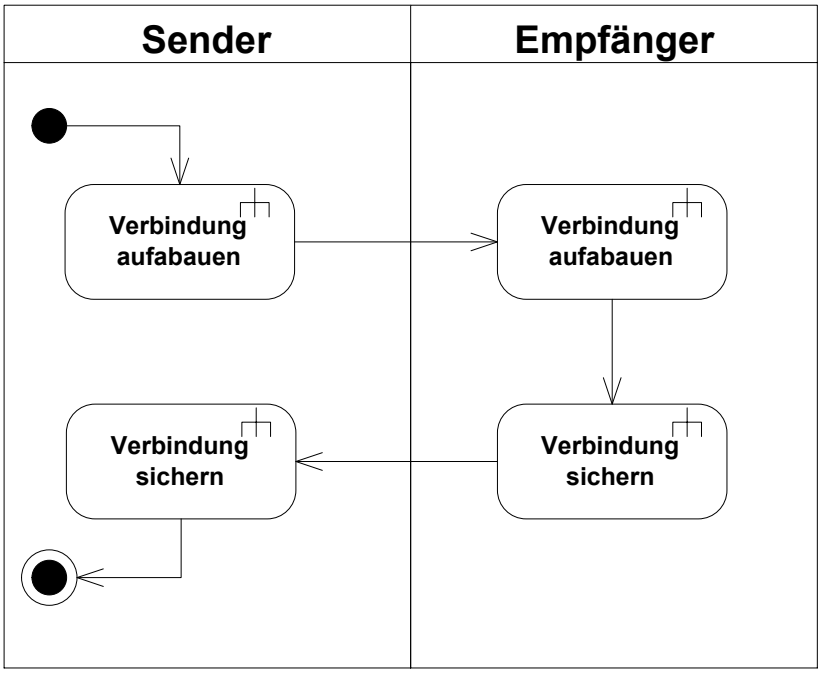


Abbildung B.2: Schritte: Verbindungsaufbau

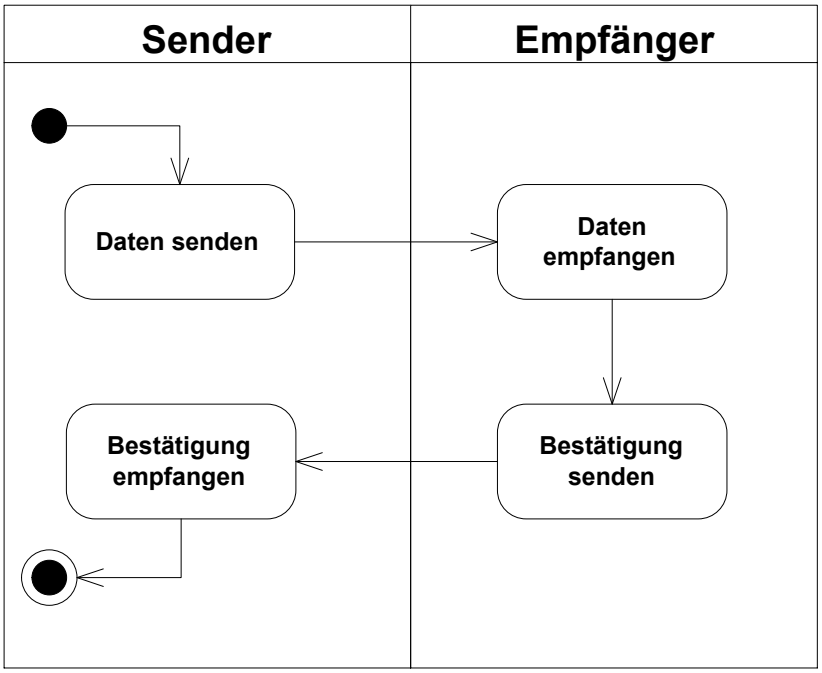


Abbildung B.3: Schritte: Übertragung

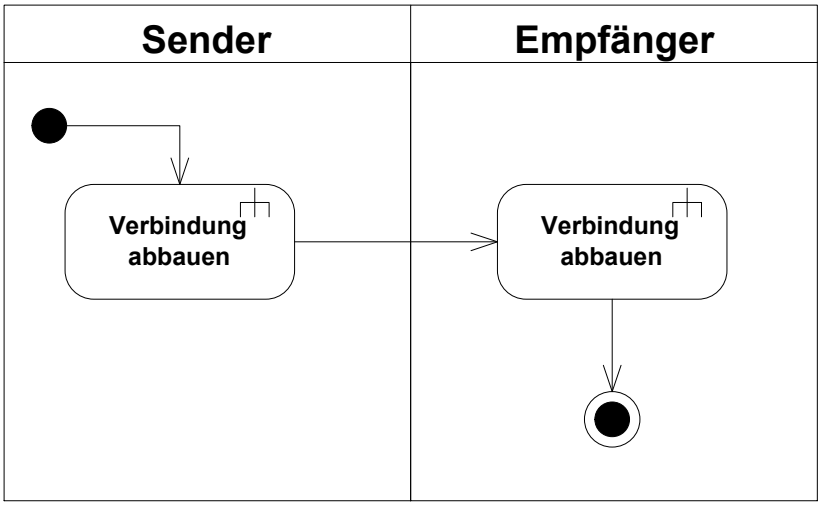


Abbildung B.4: Schritte: Verbindungsabbau

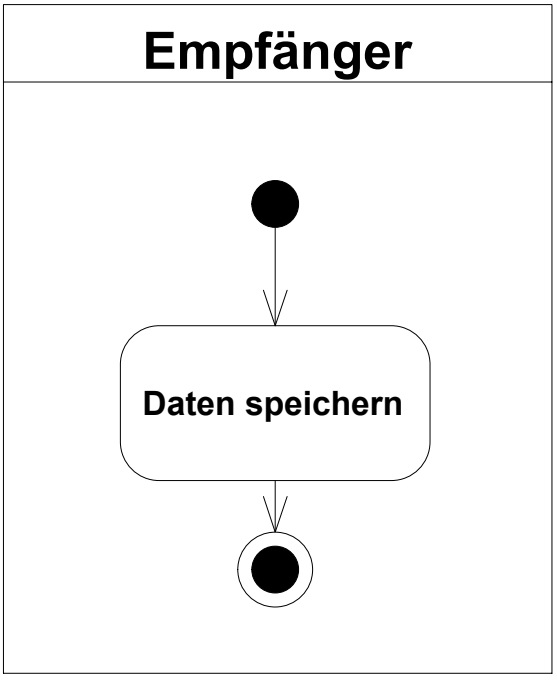


Abbildung B.5: Schritte: Nachbereitung