Diplomarbeit

Identitätsmanagement mit Hilfe von Javacards

von
Jan Peter Stotz

TECHNISCHE UNIVERSITÄT DARMSTADT

Fraunhofer Institut Sichere Informations-Technologie

Technische Universität Darmstadt
Fachbereich Informatik

Fraunhofer-Institut für Sichere Informationstechnologie

Betreuer:

Prof. Dr. Claudia Eckert
Dipl.-Inform. Mario Hoffmann

Darmstadt, Januar 2006

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 13. Januar 2006

Jan Peter Stotz

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

3DES	Triple **DES**
AES	**A**dvanced **E**ncryption **S**tandard
APDU	**A**pplication **P**rotocol **D**ata **U**nit
API	**A**pplication **P**rogramming **I**nterface
CHTML	**C**ompact **H**ypertext **M**arkup **L**anguage
COS	**C**ard **O**perating **S**ystem
CSS	**C**ascading **S**tyle **S**heets
CT-API	**C**ard **T**erminal **A**pplication **P**rogramming **I**nterface
DES	**D**ata **E**ncryption **S**tandard
EEPROM	**E**lectrically **E**raseable **P**rogrammable **R**ead **O**nly **M**emory
EJB	**E**nterprise **J**ava**B**ean
FQDN	**F**ully **Q**ualified **D**omain **N**ame
GPS	**G**lobal **P**ositioning **S**ystem
HBCI	**H**ome**b**anking **C**omputer **I**nterface
HMAC	Keyed-**H**ash **M**essage **A**uthentication **C**ode
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IDM	**Id**entitäts**m**anagement
IIS	**I**nternet **I**nformation **S**erver
J2EE	**J**ava **2** **E**nterprise **E**dition
J2ME	**J**ava **2** **M**icro **E**dition
J2SE	**J**ava **2** **S**tandard **E**dition
JAAS	**J**ava **A**uthentication and **A**uthorization **S**ervice
JC-RMI	**J**ava **C**ard **R**emote **M**ethod **I**nvocation
JSP	**J**ava **S**erver **P**ages
JVM	**J**ava **V**irtual **M**achine
LAP	**L**iberty **A**lliance **P**roject
LGPL	GNU **L**esser **G**eneral **P**ublic **L**icense
MACOS	**M**ulti **A**pplication **C**ard **O**perating **S**ystems
NAT	**N**etwork **A**ddress **T**ranslation
PIN	**P**ersonal **I**dentification **N**umber
PKCS	**P**ublic-**K**ey **C**ryptography **S**tandards
PKI	**P**ublic **K**ey **I**nfrastruktur
PTT	Ministére des **P**ostes, **T**élégraphe, **T**éléphone
RAM	**R**andom **A**ccess **M**emory
RMI	**R**emote **M**ethod **I**nvocation
ROM	**R**ead **O**nly **M**emory
SAML	**S**ecure **A**ssertion **M**arkup **L**anguage
SATSA	**S**ecurity **a**nd **T**rust **S**ervices **A**PI for J2ME

SeMoA ..... **Se**cure **Mo**bile **A**gent
SIM ..... **S**ubscriber **I**dentity **M**odule
SOAP ..... **S**imple **O**bject **A**ccess **P**rotocol
SSH ..... **S**ecure **Sh**ell
SSO ..... **S**ingle **S**ign-**O**n
TLS ..... **T**ransport **L**ayer **S**ecurity
TPM ..... **T**rusted **P**latform **M**odule
W3C ..... World Wide Web Consortium
WLAN ..... **W**ireless **L**ocal **A**rea **N**etwork
WML ..... **W**ireless **M**arkup **L**anguage
WS4EE ..... **W**eb**s**ervices **for** J2**EE**
WSDL ..... **W**eb **S**ervice **D**efinition **L**anguage
XHTML ..... E**x**tensible **H**ypertext **M**arkup **L**anguage
XML ..... E**x**tensible **M**arkup **L**anguage

1. Einleitung

Wer heutzutage durch das Internet surft, muss an vielen Stellen seine Anonymität opfern, um Zugriff auf die verschiedenen kostenlosen oder kostenpflichtigen Dienste zu erlangen. Egal, ob es um die Bestellung in einem Online-Shop geht oder um die Teilnahme in einem Webforum – ohne eine vorherige Authentifikation lassen sich nur noch sehr wenige Dienste im Internet nutzen. Da fast alle diese Dienste unabhängig voneinander sind, benötigt der Benutzer bei jedem Dienst ein eigenes Benutzerkonto. Mit jedem neuen Dienst muss sich der Benutzer zusätzliche Zugangsdaten, bestehend aus einem Benutzernamen und einem Passwort, einprägen. Gleichgültig wie viele Zugangsdaten sich ein Mensch merken kann – durch die Vielzahl der Dienste ist bei jedem Menschen irgendwann der Punkt erreicht, an dem er nicht mehr alle Zugangsdaten im Kopf behalten kann. Deshalb verwenden viele Benutzer nur wenige Kombinationen aus Benutzernamen und Passwörtern, die sie gleichzeitig für mehrere Dienste nutzen. Dadurch riskieren sie jedoch, dass unseriöse Dienstanbieter die Zugangsdaten nutzen, um sich mit der Identität des Benutzers Zugriff auf andere Dienste zu verschaffen. Andere Benutzer notieren sich alle Zugangsdaten außerhalb des Computers. Dieses Verfahren ist relativ sicher, sofern die Zugangsdaten gut geschützt vor fremden Blicken aufbewahrt werden, komfortabel ist es jedoch nicht. Wesentlich einfacher und komfortabler lassen sich Zugangsdaten mit Hilfe von sogenannten „Identitätsmanagement-Systemen“ direkt im Computer verwalten.

Aktuell gibt es eine Vielzahl von Systemen auf dem Markt, die das Identitätsmanagement vereinfachen sollen. Leider interpretiert jeder Hersteller den Begriff Identitätsmanagement unterschiedlich, sodass sich hinter einem Identitätsmanagement-System sowohl ein Authentifikationssystem, ein Zugriffskontrollsystem als auch ein System zum Verwalten von verschiedenen Identitäten eines Benutzers verbirgt. Auch die Experten sind sich nicht einig, was genau ein „Identitätsmanagement-System“ ist

und welche Funktionen es unterstützen soll. Nur bei den drei wichtigsten grundlegenden Eigenschaften, die ein Identitätsmanagement-System erfüllen sollte, sind sie sich halbwegs einig. Diese sind: „Schutz der Privatsphäre, Sicherheit und Benutzbarkeit“ [Unab03, S.v].

Auch wenn mehrere Ansichten darüber existieren, was ein Identitätsmanagement-System ausmacht, in dieser Arbeit bezeichnet ein Identitätsmanagement-System immer ein System, das einem Benutzer erlaubt, seine elektronischen Identitäten (z. B. die Zugangsdaten) zu verwalten. Eine elektronische Identität besteht dabei einmal aus den Zugangsdaten, die man bei einem Dienst verwendet hat und umfasst zusätzlich alle Daten, die man bei diesem Dienst über sich angegeben hat. Das könnte beispielsweise der echte (reale) Name des Benutzers oder ein imaginärer Name (Pseudonym) sein. Identitäten mit imaginären Namen werden oft als als virtuelle Identitäten bezeichnet.

Je nach Identitätsmanagement-System werden die Daten der verschiedenen Identitäten lokal auf dem Computer des Benutzers oder auf einem zentralen Server gespeichert. Die lokale Speicherung, beispielsweise im Webbrowser des Benutzers, hat den Vorteil, dass der Benutzer den Zugriff auf seine Daten einfach kontrollieren kann. Gleichzeitig hat sie jedoch den Nachteil, dass die gespeicherten Informationen nur auf einem einzelnen Computer zur Verfügung stehen. Die Speicherung der Daten auf einem zentralen Server kennt dieses Problem nicht. Hier kann der Benutzer von einem beliebigen Endgerät auf die eigenen gespeicherten Daten zugreifen. Dafür muss der Benutzer sich damit abfinden, dass er nicht mehr kontrollieren kann, wer auf seine Daten zugreift. Der Betreiber des Servers, auf dem die Daten gespeichert sind, kann beliebig Daten auslesen und kopieren, ohne dass der Benutzer etwas davon merkt. Damit dies nicht geschieht, haben viele Länder Gesetze und Verordnungen erlassen, die den Umgang mit „personenbezogenen Daten“ genau regeln, beispielsweise in Deutschland das Bundesdatenschutzgesetz [BDSG]. Trotzdem muss der Benutzer dem Betreiber eines Servers glauben, dass dieser sich an die bestehenden Datenschutzgesetze und Datenschutzverordungen hält. Durch menschliche oder technische Fehler passiert es immer wieder, dass Unberechtigte Zugriff auf die persönlichen Daten von Benutzern erhalten. Oftmals gehen Unternehmen mit den ihnen anvertrauten Benutzerdaten geradezu fahrlässig um. Im Jahr 2005 sind zwei Fälle bekannt geworden, bei denen einer Bank bzw. einer Handelskette Datenträger mit sensiblen Daten von tausenden von Benutzern gestohlen wurden [Blei05][Zieg05].

Angriffe auf die persönlichen Daten von Benutzern sind keine Seltenheit. Insbesondere mit Bankverbindungsdaten und Kreditkartennummern wird im Internet in einigen illegalen Foren ein reger Handel betrieben. Auch Informationen über die E-Mail Adressen von Millionen von Benutzern lassen sich in diesen Foren käuflich erwer-

ben. Außerhalb der Illegalität betreiben einige Firmen Handel mit den Informationen über ihre Benutzer. Die persönlichen Daten von Benutzern sind deshalb aus der Sicht Vieler zu einer Art „Handelsware“ geworden. Denkt man nun an Identitätsmanagement-Systeme, die an zentraler Stelle die persönlichen Daten von vielen Benutzern speichern, wird deutlich, dass bei diesen Systemen von vielen Seiten mit Angriffen auf die Benutzerdaten zu rechnen ist. Schon ein einziger krimineller Mitarbeiter reicht aus, um die Sicherheit der Benutzerdaten zu gefährden. Sehr eindrucksvoll hat das der Fall „Jason Smathers“ gezeigt. Smathers, ein ehemaliger AOL-Mitarbeiter, hat im Jahr 2004 über 92 Millionen E-Mail-Adressen von AOL-Kunden für mehrere tausend Dollar an einen Junk-Mail-Versender verkauft [Kuri04].

Vor diesem Hintergrund wird verständlich, wieso bisherige Versuche Identitätsmanagement-Systeme mit zentraler Speicherung der Daten zu etablieren, von den Benutzern nicht akzeptiert wurden (Beispiel: Microsoft Passport). Mit Hilfe von kryptographischen Verfahren lässt sich die Sicherheit der Daten in solchen Identitätsmanagement-Systemen entscheidend verbessern. Gleichzeitig kann dem Benutzer wieder mehr Kontrolle über seine Daten zurückgegeben werden. Der Entwurf und die Implementierung eines solchen Identitätsmanagement-Systems ist das Ziel dieser Arbeit.

1.1 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, ein Identitätsmanagement-System zu entwickeln, das es dem Benutzer erlaubt, trotz Speicherung auf einem zentralen Server, die Kontrolle über seine Daten zu behalten. Dies soll durch die Verschlüsselung der Benutzerdaten auf dem zentralen Server erreicht werden. Die für die Entschlüsselung notwendigen kryptographischen Schlüssel befinden sich unter der Kontrolle des Benutzers. Der Zugriff auf die Benutzerdaten auf dem Server durch den Betreiber oder durch einen Angreifer ist dadurch nicht möglich. Neben der Sicherheit der Benutzerdaten und der Zugriffskontrolle auf diese Daten soll das zu entwickelnde System auch möglichst einfach zu bedienen sein. Die Sicherheit der Benutzerdaten darf deshalb nicht zu Lasten der Benutzbarkeit des Systems gehen, da dies sich negativ auf die Benutzerakzeptanz auswirken kann.

Aus diesen Gründen erfolgt die Authentifikation des Benutzers gegenüber dem System nicht durch ein langes und kryptisches Passwort (wissensbasierte Authentifikation), sondern über eine Chipkarte (Authentifikation durch Besitz) und ein Karten-Passwort (PIN). Die Chipkarte verhindert Angriffe, bei denen versucht wird das Passwort zu erraten (Brute-Force-Angriff). Dadurch kann das Karten-Passwort wesentlich kürzer und einfacher sein ohne die Sicherheit zu beeinträchtigen. Ein einfacheres Passwort lässt sich zudem vom Benutzer leichter merken. Darüber hinaus lässt sich die Chipkarte als sicherer Speicher für die kryptographischen Schlüssel

verwenden. Ein weiterer Vorteil der Chipkarte ist ihre Fähigkeit, besonders sicherheitskritische Operationen innerhalb der Chipkarte und damit außerhalb der Reichweite eines Angreifers ausführen zu können. Bei der Entwicklung des Systems muss berücksichtigt werden, dass ein Benutzer seine Chipkarte verlieren bzw. diese einen Defekt aufweisen kann. Deshalb soll es jedem Benutzer möglich sein, mehrere identisch funktionierende Chipkarten zu besitzen. Im Fall des Verlustes einer Chipkarte muss der Benutzer in der Lage sein, die betreffende Chipkarte vom Zugriff auf die eigenen Daten auszuschliessen, um einen eventuellen Missbrauch zu verhindern.

Als Chipkarten kommen sogenannte Java-Karten („Java Card“[1]) zum Einsatz, eine Familie von programmierbaren Chipkarten mit eigenem Prozessor und der Fähigkeit Daten kryptographisch zu verschlüsseln und zu signieren.

1.2 Gliederung der Arbeit

Die weitere Arbeit ist in fünf Teile gegliedert, wobei jeder Teil einem Kapitel entspricht. Im Ersten Teil, dem zweiten Kapitel, werden einige bekannte Identitätsmanagement-Systeme untersucht sowie ihre Vor- und Nachteile verglichen. Außerdem befasst sich dieses Kapitel mit den notwendigen Techniken, die im Verlauf dieser Arbeit verwendet werden. Dazu gehört die Java-Karte, deren technischer Aufbau, Fähigkeiten und Sicherheitsfunktionen vorgestellt werden. Um den aktuellen Stand der Technik bei den Java-Karten besser verstehen zu können, enthält dieses Kapitel auch einen Abschnitt über die historische Entwicklung der Chipkarten. Am Ende des zweiten Kapitels wird das MOBILE-Projekt vorgestellt. Dieses ist für die weitere Arbeit von entscheidender Bedeutung, da das zu entwickelnde Identitätsmanagement-System eine Weiterentwicklung dieses Projektes darstellt. Um bei der Weiterentwicklung die Probleme und Schwachstellen des MOBILE-Projekt nicht zu übernehmen, werden diese analysiert. Das ermöglicht es, gezielt Lösungen für diese Probleme in dieser Arbeit zu finden und später zu implementieren.

Das dritte Kapitel befasst sich mit der Entwicklung des Designs des *Mobile*-Projektes (der Weiterentwicklung des MOBILE-Projektes). Nach der Definition der Anforderungen an das *Mobile*-Projekt und einem Überblick über das geplante System folgt die Entwicklung des Verschlüsselungssystems, das dem Benutzer die Kontrolle über seine Daten ermöglicht. Dieses wird in mehreren aufeinander aufbauenden Schritten entwickelt, bis am Ende alle vorher definierten Anforderungen erfüllt werden. Anschließend wird der Entwurf der Client-Software und der Anwendung für die Java-Karte skizziert. Die Client-Software wird auf dem Endgerät des Benutzers installiert und ermöglicht die Kommunikation zwischen dem zentralen Server und der

[1] `http://java.sun.com/products/javacard/index.jsp`

Java-Karte. Im letzten Abschnitt des dritten Kapitels wird die dazugehörige Server-Software vorgestellt.

Die Details der Implementierung und der verwendeten Software-Komponenten, sowie die bei der Implementierung aufgetretenen Probleme, behandelt das vierte Kapitel. Dieses gliedert sich in drei Abschnitte, von denen das Erste die Entwicklung der Java-Karten-Anwendung zum Thema hat. Außerdem beinhaltet dieser Abschnitt einen detailierten Einblick in die Klassen-Struktur der Karten-Anwendung sowie in die verwendeten Datenformate. Der zweite Abschnitt beschreibt die Implementierungsdetails der Client-Software und die bei der Realisierung aufgetretenen Probleme bei der Kommunikation mit der Java-Karten-Anwendung. Ergänzt wird dieser Abschnitt durch die Vorstellung einiger Bildschirmfotos (Screenshots) der Benutzungsoberfläche. Der letzte Abschnitt des vierten Kapitels behandelt die Server-Software inklusive der Datenbank, in der die Benutzerdaten gespeichert werden.

Im fünften Kapitel folgt dann die Analyse und Bewertung der im vierten Kapitel beschriebenen Implementierung. Die Analyse erfolgt an Hand der fünf Kriterien „Benutzbarkeit“, „Sicherheit“, „Datenschutz“, „Durchsetzung der Gesetze und Beweisbarkeit“ sowie „Vertrauenswürdigkeit“. Die Auswahl der Kriterien stammt aus der europäischen Vergleichsstudie von Identitätsmanagement-Systemen aus dem Jahr 2003 [Unab03].

Das letzte Kapitel dieser Arbeit präsentiert Anknüpfungspunkte für die Weiterentwicklung des vorgestellten Systems. Dazu gehören mögliche Erweiterungen mit bereits existierenden Technologien, die beispielsweise die Nutzung vereinfachen oder die Plattform um neue Funktionen und Anwendungsgebiete ergänzen. Außerdem werden verschiedene neue Technologien vorgestellt, die sich hervorragend für die Integration in das *Mobile*-Projekt eignen. Dazu gehört auch die Vorstellung eines neuen Identitätsmanagement-Systems, das sich aktuell noch in der Entwicklung befindet und voraussichtlich im nächsten Jahr (2006) auf den Markt kommen wird.

2. Stand der Technik

In diesem Kapitel werden Technologien und Projekte vorgestellt, die entweder bei der Entwicklung dieser Arbeit zum Einsatz kamen oder mit dem Thema dieser Arbeit verwandt sind. Das Kapitel ist unterteilt in drei Abschnitte. Im ersten Abschnitt wird das Thema „Identitätsmanagement-Systeme“ genauer untersucht. Dabei werden an Hand einiger aktueller Identitätsmanagement-Systeme deren Vor- und Nachteile herausgearbeitet und verglichen. Im zweiten Abschnitt wird dann die Entwicklung der Chipkarten bis hin zu den aktuellen Java-Karten beschrieben. Neben dem Aufbau von Chipkarten wird analysiert, wie sich die Chipkarten unterscheiden und wie die Kommunikation mit einer Java-Karte funktioniert. Außerdem wird untersucht, welche Sicherheitseigenschaften Java-Karten besitzen. Im letzten Abschnitt folgt die Vorstellung eines Projektes, das den Rahmen für das System bildet, welches in den beiden folgenden Kapiteln entwickelt wird.

2.1 Identitätsmanagement-Systeme

Hinter dem Begriff „Identitätsmanagement-System“ verbergen sich die verschiedensten Systeme, die es einem Benutzer erlauben, seine zahlreichen elektronischen Identitäten bei den verschiedenen Diensten und Anbietern zu verwalten. Solche Systeme lassen sich in drei Kategorien einteilen:

1. lokale, clientbasierte Identitätsmanagement-Systeme
2. zentralisierte, serverbasierte Identitätsmanagement-Systeme
3. dezentrale, verteilte Identitätsmanagement-Systeme

Zu den clientbasierten Identitätsmanagement-Systemen zählt man beispielsweise die Passwort- und Formular-Eingabehilfen, die viele aktuelle Webbrowser (Mozilla,

Firefox, Internet Explorer, Opera) enthalten. Diese können einmal auf Webseiten eingegebene Passwörter und Daten abspeichern. Besucht der Benutzer dieselbe Seite später wieder, kann er auf die gespeicherten Daten zurückgreifen und muss sie nicht noch einmal eingeben. Zudem bietet z. B. Mozilla die Möglichkeit, die auf den verschiedenen Webseiten eingegebenen Daten in Form einer Liste darzustellen. Mit Hilfe dieser Liste kann ein Benutzer genau nachvollziehen, bei welchem Dienst er welche Informationen über sich hinterlassen hat. Andere Webbrowser, wie der Internet Explorer oder der Webbrowser Opera, bieten die Möglichkeit, Benutzerdaten im Webbrowser zu hinterlegen. Diese werden beim Ausfüllen von Formularen auf beliebigen Webseiten im Internet angeboten. Die gesammelten Formular-, Passwort- und Profildaten werden von den Webbrowsern in der Standardeinstellung unverschlüsselt auf der Festplatte abgespeichert. Auf Computern, die nur von einer Person genutzt werden, besteht nur durch Schadsoftware wie Viren, Würmern oder Trojanischen Pferden Gefahr für die gesammelten Daten. In Mehrbenutzersystemen hingegen, wie sie in Firmen üblich sind, können diese Daten leicht in falsche Hände geraten. Um die gesammelten Daten besser zu schützen, können sie verschlüsselt werden. Die Verschlüsselung ist in vielen Webbrowsern optional und muss erst vom Benutzer aktiviert werden, sie wird deshalb eher selten verwendet. Ein weiterer Nachteil des browserbasierten Identitätsmanagements ist die Bindung der Benutzerdaten an eine Webbrowser-Konfiguration und damit im Normalfall auch an den verwendeten Computer. Verwendet man mehrere Computer, werden auf jedem andere Daten gespeichert. Eine automatische Synchronisation mehrerer Webbrowser-Konfigurationen auf unterschiedlichen Computern bietet kein aktuell verfügbarer Browser.

Reicht die Verschlüsselung der Daten im Webbrowser mit einem Passwort nicht aus, kann man zusätzliche Hardware in Form einer Chipkarte oder eines USB Tokens verwenden. Die Daten werden dabei in dem Hardware-Token selbst abgespeichert und sind damit vor unautorisiertem Zugriff geschützt und zusätzlich auf verschiedenen Computern verwendbar. Solche Hardware-Systeme beschränken sich durch den geringen Speicherplatz des Hardware-Tokens auf das Speichern von Passwörten. Durch die hohen Kosten für das Hardware-Token wird diese Variante des Identitätsmanagement nur von sehr wenigen Benutzern eingesetzt.

Auch die zentralen, serverbasierten Identitätsmanagement-Systeme haben sich bisher nicht durchsetzen können. Das bekannteste System in dieser Kategorie ist der Dienst „Microsoft Passport“ (die genaue Vorstellung dieses Systems folgt in Abschnitt 2.1.2). Andere zentrale Identitätsmanagement-Systeme wie das Projekt „DigitalMe“ von Novell sind mittlerweile wieder vom Markt verschwunden. Übrig geblieben sind hauptsächlich Systeme, deren primärer Einsatzzweck im Be-

reich der Authentifikation sowie der Autorisation und nicht in der Verwaltung der Identitäten eines Benutzers liegen. Es handelt sich dabei zumeist um Benutzer-Verzeichnisse (Directory-Services), wie das Active Directory von Microsoft oder das Novell eDirectory. Diese werden innerhalb von Unternehmen für die zentrale Verwaltung der Mitarbeiterdaten genutzt.

Die letzte Kategorie der Identitätsmanagement-Systeme besteht aus nur einem System, dem „Liberty Alliance Project“. Dieses System hat zum Ziel, die schon vorhandenen Authentifikationssysteme, Autorisierungssysteme und Identitätsmanagement-Systeme miteinander zu vernetzen. Dadurch entsteht ein Identitätsmanagement-System, das sich auf mehrere vorhandene Systeme verteilt. Die genaue Vorstellung des „LAP“ folgt in Abschnitt 2.1.3.

2.1.1 Single Sign-on

Das Prinzip des Single Sign-On (SSO) beschreibt ein System, bei dem der Benutzer davon befreit wird, sich für jede Identität die zur Anmeldung nötigen Daten, wie Benutzername und Passwort, zu merken. Stattdessen werden nur noch die Anmeldedaten des SSO-Systems benötigt. Wie ein SSO-System dieses Verhalten realisiert, oder welche Techniken dabei verwendet werden, ist nicht genau festgelegt.

2.1.2 Microsoft Passport

Mit dem System „Passport“ bietet Microsoft seit Mitte 1999 ein kostenloses und von jedem Internet-Nutzer verwendbares Identitätsmanagement-System an. Für die Verwendung von Passport ist auf der Benutzerseite keine zusätzliche Software notwendig, da Passport nur Techniken wie SSL, JavaScript und Cookies verwendet, die von aktuellen Webbrowsern unterstützt werden. Zum Anlegen eines Passport-Kontos ist nur eine gültige E-Mail Adresse notwendig. Pro Benutzerkonto kann man die Daten einer Identität oder eines Pseudonyms verwalten. Verwendet eine Webseite Passport als Authentifikationsdienst, wird der Benutzer zum Anmelden automatisch auf die Seite `www.passport.com` weitergeleitet. Nach korrekter Authentifikation setzt diese Seite ein verschlüsseltes Cookie, das die Benutzerkennung und das Passwort des Benutzers enthält. Ein zweites verschlüsseltes Cookie enthält die Profildaten des Benutzers und ist für die Webseite bestimmt, welche die Authentifikation veranlasst hat. Bei allen nachfolgenden Weiterleitungen auf die Passport-Seite wird der Benutzer automatisch über das verschlüsselte Cookie mit der Benutzerkennung und dem Passwort authentifiziert. Eine erneute Passworteingabe ist dadurch nicht mehr notwendig. Um mehrere Identitäten mit Passport zu verwalten, muss man für jede Identität ein eigenes Passport-Konto anlegen. Jedes dieser Konten benötigt je eine E-Mail-Adresse und ein Passwort. Man kann jedoch immer nur mit einem Passport-

Konto pro Webbrowser angemeldet sein. Mit mehreren Identitäten ist demnach kein Single Sign-On möglich.

Betreiber von Webseiten, die Passport als Authentifikations-System anbieten wollen, müssen zwingend den Internet Information Server (IIS) als Webserver verwenden. Nur für diesen bietet Microsoft ein Modul an, welches die Nutzung der Passport-Authentifikation ermöglicht. Alle Anbieter müssen sich außerdem für die Nutzung von Passport bei Microsoft registrieren.

Um die Sicherheit des Passport Dienstes war es von Anfang an eher schlecht bestellt. Neben einigen theoretischen Angriffsmöglichkeiten (Abfangen der Authentifikations-Cookies oder Man-in-the-Middle Angriffe durch Umleiten des Benutzers auf eine gefälschte Login-Seite [Ecke02, S.429]) wurden einige gravierende Sicherheitslücken gefunden, über die sensible Benutzer-Daten von Hackern ausgelesen werden konnten [Welt03]. Die damit verbundene negative Presse und das bei vielen Benutzern vorhandene Misstrauen gegenüber Microsoft sorgten dafür, dass sich der Passport-Dienst nicht durchsetzen konnte. Microsoft gibt zu, dass sie mit ihrem Ziel ein „Identitäts-Provider für das Internet zu sein“ gescheitert sind [Micr05b]. Selbst das große Online-Auktionshaus eBay hat Anfang 2005 die Möglichkeit eingestellt, sich über Microsoft Passport zu authentifizieren.

2.1.3 Liberty Alliance Project

Das Identitätsmanagement-System „Liberty Alliance Project“ (LAP)[1] wurde 2001 von Sun Microsystems entwickelt. Das Ziel war es, eine Alternative zum Microsoft Passport Dienst zu schaffen. Im Gegensatz zum proprietären Passport-System, dessen Schnittstellenbeschreibungen nicht öffentlich zugänglich sind, ist das Liberty Alliance Project ein offenes, verteiltes Identitätsmanagement-System, mit standardisierten Schnittstellen [Wiki05b].

Das Grundprinzip des LAP ist die Verknüpfung von mehreren verschiedenen Identitäten. Im Internet kann man sich bei vielen Dienstanbietern (service provider) registrieren und erhält dann ein Benutzerkonto bei diesem Dienst. Jedes Benutzerkonto stellt eine eigenständige, virtuelle Identität dar, die im LAP als „lokale Identität“ (local identity) bezeichnet wird. Ein Großteil der regelmäßigen Internet-Benutzer besitzen eine Vielzahl solcher Identitäten verstreut über das ganze Internet. Im LAP lassen sich diese Identitäten zu einer „vereinigten Identität“ (federated identity) verbinden. Dafür müssen mehrere der Dienstanbieter zusammenarbeiten und eine gemeinsame Authentifikationsdomäne bilden. Diese wird im LAP als „Circle Of Trust“ bezeichnet. Innerhalb dieser Domäne muss es mindestens einen Authentifikationsdienst (identity provider) geben, dem alle Dienstanbieter vertrauen. Das

[1] `http://www.projectliberty.org`

kann zum Beispiel einer der beteiligten Dienstanbieter sein. Dieser übernimmt die Authentifikation aller Benutzer für die beteiligten Dienste. Ein Benutzer muss sich nur einmal bei diesem Authentifikationsdienst anmelden und wird danach von allen Diensten in der Domäne automatisch als angemeldet akzeptiert. Die Verknüpfung der verschiedenen lokalen Identitäten mit der Identität des Authentifikationsdienstes wird jedoch nicht automatisch erstellt. Der Benutzer muss die Identitäten einmalig verknüpfen, erst danach funktioniert das Single Sign-On. Der Benutzer kann diese Verknüpfungen jederzeit wieder lösen [Sun 05].

Neben dem bereits vorgestellten Authentifikationsdienst (identity provider) und dem Dienstanbieter (service provider) kennt das LAP noch den Attributs-Dienst („attribute provider“). Dieser speichert Daten (Attribute) eines Benutzers, die dann von den Diensten der selben Authentifikationsdomäne abgefragt werden können. Über das Setzen von Zugriffsregeln kann der Benutzer bestimmen, welcher Dienstanbieter auf welche Daten zugreifen darf.

Durch die Nutzung mehrerer Authentifikationsdomänen kann ein Benutzer vollständig voneinander getrennte Profile anlegen. Denkbar ist z. B. ein berufliches Profil, bei dem der Authentifikationsdienst des Arbeitgebers verwendet wird. Dadurch wird die Nutzung der innerbetrieblichen Dienste ermöglicht. Daneben existiert ein berufliches Profil, bei dem die Bank des Benutzers den Authentifikationsdienst stellt. Auch eine Unterteilung der Profile in Sicherheitsstufen ist denkbar.

Auf der technischen Seite definiert das Liberty Alliance Project lediglich die Kommunikation zwischen den einzelnen Diensten, z. B. zwischen dem Authentifikationsdienst und dem Dienstanbieter. Diese übertragen die Authentifikationsdaten durch Dateien in der Secure Assertion Markup Language (SAML) über Webservices. Für die Benutzungsoberfläche gibt es im LAP hingegen lediglich Richtlinien und Empfehlungen, welche Techniken (z. B. Cookies) man bevorzugen oder vermeiden sollte. Die Wahl, welche Techniken letztendlich zum Einsatz kommen, bleibt dem Entwickler einer LAP-kompatiblen Software überlassen. Aktuell (Ende 2005) existieren bereits etliche Produkte, die erfolgreich auf ihre Konformität mit der LAP-Spezifikation getestet wurden. Eine Liste dieser Produkte befindet sich auf der Homepage des Liberty Alliance Project[2].

Auch wenn dem Liberty Alliance Project der große Durchbruch noch nicht gelungen ist, zeigt insbesondere die stetig wachsende Zahl an kompatiblen Software-Produkten, dass LAP sich zu einem Standard im Bereich des Identitätsmanagement entwickelt hat. Im Moment ist der Nutzen für die Benutzer noch gering, da sich erst wenige Anbieter zu „Circles of Trust“ zusammengeschlossen haben. Deshalb las-

[2] `http://www.projectliberty.org/activities/conformant_products.php`

sen sich in der Praxis die Vorteile von LAP, wie das Verknüpfen von verschiedenen Identitäten, bisher noch nicht ausnutzen. Das größte Problem von LAP liegt jedoch in der Komplexität des Systems. Ohne grundlegende Kenntnisse der hinter LAP stehenden Konzepte lässt sich nur ein geringer Teil der Funktionalität und damit auch der Vorteile nutzen. Durch eine Schulung lässt sich das Wissen der Benutzer über LAP zwar verbessern, die Komplexität dürfte jedoch trotzdem viele Benutzer überfordern und deshalb zu einer geringen Benutzerakzeptanz führen.

2.2 Chipkarten, Smart Cards und Java Cards

Chipkarten, kleine Plastikkarten mit einer goldenen Kontaktfläche, sind aus unserem Alltag nicht mehr wegzudenken. Sie dienen als „Telefonkarte“ im öffentlichen Fernsprecher oder im Mobiltelefon, als Krankenversichertenkarte beim Arzt oder als elektronische Geldbörse. Doch bei diesen Anwendungsfeldern wird es in Deutschland nicht bleiben. In Planung befindet sich schon die elektronische Gesundheitskarte (eGK), die „Arbeitnehmer-Chipkarte“ (JobCard), und es wird sogar über einen Chipkarten-Personalausweis nachgedacht.

Auch wenn die ersten Chipkarten erst seit etwa 1980 verfügbar waren, begann der Siegeszug der Plastikkarten und damit auch der Chipkarten schon wesentlich früher. Plastikkarten in „Scheckkartengröße“ (85,7mm $*$ 54mm) wurden erstmals 1950 in den USA ausgegeben. Es handelte sich dabei um die Kreditkarten des „Diners Club“, auf deren Oberfläche alle Informationen aufgedruckt waren. Später wurden die Informationen zusätzlich in das Plastik der Karte geprägt, was aber nichts daran änderte, dass nur Menschen die Informationen ablesen konnten. Maschinenlesbar wurden die Plastikkarten erst 1970, durch Einbettung eines Magnetstreifens. Das Problem der Magnetstreifenkarte ist, dass die Informationen auf dem Magnetstreifen sich nicht schützen lassen. Mit Hilfe eines passenden Schreib-/Lesegerätes kann jeder die Informationen auslesen und manipulieren. Damit ist der Magnetstreifen unbrauchbar für die Speicherung von sensiblen Daten, beispielsweise einer Personal Identification Number (PIN), mit der es möglich wäre, den Besitzer der Karte zu authentifizieren (wissensbasierte Authentifikation). Schon 1968, zwei Jahre vor Einführung der Magnetstreifenkarte, hatte der deutsche Erfinder Jürgen Dethloff die Idee, einen „integrierten Schaltkreis in eine Identifikationskarte einzubauen“ [RaEf02, S.3]. Die daraus resultierende Patentschrift „Elektronischer Identifikant“ gilt als die Geburtsstunde der Chipkarte.

2.2.1 Historie

Der nächste bedeutende Entwicklungsschritt kam durch den französichen Erfinder Roland Moreno, der 1972 ein Patent anmeldete, in dem er ein „*unabhängiges, elektro-*

nisches Objekt, entwickelt für die Speicherung von vertraulichen Daten“ beschreibt [More76]. Das Auslesen der vertraulichen Daten benötigt eine vorherige Authentifikation des rechtmäßigen Eigentümers durch einen „*geheimen Code*“, der innerhalb des Objektes geprüft wird. Fehlgeschlagene Authentifikations-Versuche werden in einem eigenen internen Speicher protokolliert. Damit ist Moreno der Erfinder der „Smart Card“, der intelligenten Chipkarte mit PIN-Authentifikation. Von der Idee bis zur Umsetzung war es noch ein langer Weg. Motorola lieferte 1979 den ersten Chipkarten-Prozessor, der nur aus einem einzigen Chip bestand. Die Integration aller benötigten Komponenten auf einem einzigen Chip erhöhte die Sicherheit der gespeicherten Daten, denn ein Abhören der Kommunikation zwischen den Komponenten wurde dadurch wesentlich erschwert.

Die erste Anwendung von Chipkarten kam 1982 in Form einer Telefonkarte. Das französische Ministerium für Post und Telekommunikation PTT führte in diesem Jahr einen Feldtest durch, um sowohl die mechanische Belastung im Alltagsgebrauch, als auch die Manipulationssicherheit des Guthabens auf der Karte zu testen. Der Test verlief so erfolgreich, dass die Telefonkarte 1983 offiziell eingeführt wurde. Die Verbreitung der neuen Technik übertraf alle Erwartungen. Nur zwei Jahre nach der Einführung der Telefonkarte waren bereits mehrere Millionen davon im Umlauf. In den nächsten Jahren kam eine Vielzahl weiterer Einsatzgebiete für die Chipkarten dazu, unter anderen 1991 als SIM in GSM-Mobiltelefonen, 1994 als deutsche Krankenversichertenkarte und 1996 als deutsche Geldkarte eingeführt.

Der Chipkartenhersteller Schlumberger veröffentlichte 1996 die Spezifikation einer Programmierschnittstelle (API), für eine Chipkarte, die in der Programmiersprache Java erstellte Programme ausführen kann (Java Card 1.0). Zusammen mit anderen Chipkartenherstellern und Sun Microsystems wurde das „Java Card Forum“[3] gegründet, das sich seit dieser Zeit um die Weiterentwicklung der Java-Karte kümmert. Die erste Veröffentlichung des Forums war die Spezifikation der „Java Card 2.0“ im Jahr 1997.

2.2.2 Typen

Innerhalb der Gruppe der Chipkarten unterscheidet man zwei grundlegende Typen. Zum einen die Speicher-Chipkarte („memory card“), die nur eine einfache Logik enthält, beispielsweise für den Speicherzugriff und das vorherige Prüfen der PIN. Bekannte Beispiele für Speicher-Chipkarten sind Telefonkarten und die deutsche Krankenversichertenkarte. Zum anderen die Prozessor-Chipkarte („microprocessor card“ oder „smart card“), die über einen zentralen Hauptprozessor verfügt. Beispiele für Prozessor-Chipkarten sind die GSM SIM-Card und die deutsche Geldkarte.

[3] `http://www.javacardforum.org`

2.2.3 Aufbau

Um die Kompatibilität der Chipkarten mit den Chipkarten-Lesegeräten (oft auch als „card terminal“ oder „card acceptance device - CAD“ bezeichnet) sicherzustellen, werden alle grundlegenden Eigenschaften der Chipkarte seit 1987 im Standard ISO 7816 beschrieben. Neben den Abmessungen der Karte und der mechanischen Belastbarkeit (ISO 7816-1) sind die genaue Position und die minimale Größe der Kontaktfächen auf der Chipkarte definiert (ISO 7816-2). Es existieren drei verschiedene Chipkartengrößen (siehe Abbildung 2.1), gebräuchlich sind jedoch nur die Größe ID-1 für normale Chipkarten und ID-000 für SIM-Karten in Mobiltelefonen.

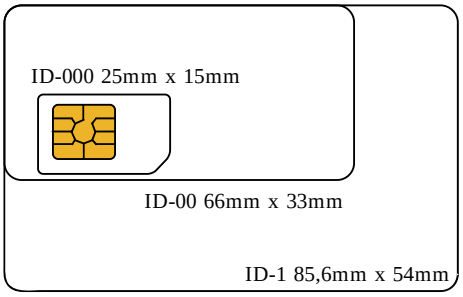

Abbildung 2.1: Chipkarte mit Abmessungen und Kontaktfläche nach ISO 7816

Das Herz der Chipkarte, der integrierte Schaltkreis hinter der Kontaktfläche, beinhaltet bei Speicher-Chipkarten einen wiederbeschreibbaren Speicher (EEPROM) und einen Lesespeicher (ROM). Prozessor-Chipkarten besitzen zusätzlich einen Hauptprozessor, der oft durch einen oder mehrere kryptographische Coprozessoren (z. B. für die RSA Ver- und Entschlüsselung) unterstützt wird. Ein kleiner flüchtiger Speicher (RAM) dient dazu die Ein-/Ausgabe-Pakete zwischenzuspeichern.

2.2.4 Chipkarten-Betriebssysteme

Jede Chipkarte besitzt ein mehr oder weniger einfaches Betriebssystem, das auch als Card Operating System (COS) bezeichnet wird. Dieses befindet sich im ROM der Karte. Die durch das Betriebssystem bereitgestellte Funktionalität hängt stark von der technischen Ausstattung der Chipkarte und dem geplanten Einsatzzweck ab. In den 80er Jahren war es üblich, Chipkarten mit einem speziell für die geplante Anwendung optimierten Betriebsystem auszustatten. Die fehlende Standardisierung zwischen den Betriebssystemen machte es unmöglich, eine Anwendung von einem Karten-Betriebsystem auf ein anderes zu portieren. Die Anwendungsentwickler waren an das einmal gewählte Betriebssystem und damit an seinen Hersteller gebunden. Auch waren diese Betriebssysteme nicht dafür ausgelegt mehrere unabhängige Anwendungen auf einer Chipkarte zu verwalten.

Heute geht der Trend zu Karten-Betriebsystemen, die mehrere Anwendungen auf einer Chipkarte erlauben. Diese sogenannten Multi Application Card Operating

Systems (MACOS) erlauben es, Anwendungen verschiedener Hersteller auf einer Karte auszuführen, ohne dass sich diese gegenseitig beeinflussen können. Außerdem wurden herstellerunabhängige Schnittstellen geschaffen, um Anwendungen ohne Anpassungen auf verschiedenen Karten-Betriebssystemen ausführen zu können. Die bekanntesten Vertreter dieser sogenannten „offenen Plattformen“ sind „MultOS“, ein offenes Hochsicherheits-Betriebsystem, und die „Java Card“, die es ermöglicht, Chipkartenanwendungen in der Programmiersprache Java zu schreiben. Auf Karten dieser Plattformen können zu einer beliebigen Zeit Anwendungen installiert oder gelöscht werden.

2.2.5 Java Card

Der Begriff „Java Card“ beschreibt eine Prozessor-Chipkarte, deren Betriebssystem eine Java Virtual Machine (JVM) für die Ausführung von Java-Programmen auf der Chipkarte enthält. Ein solches Programm wird Karten-Anwendung (card applet) genannt. Die Funktionalität, auf die eine Karten-Anwendung zurückgreifen kann, ist in der Java Card-Spezifikation beschrieben. Diese definiert das Application Programming Interface (API), das jeder Chipkarten-Hersteller in der JVM implementierten muss. Dadurch lässt sich ein Programm auf Java-Karten unterschiedlicher Hersteller ohne zusätzliche Anpassungen ausführen. Die Java-Karte ist ein MACOS, sie erlaubt die Installation mehrerer Anwendungen auf einer Chipkarte.

Karten-Anwendungen werden in der Programmiersprache Java erstellt. Ein Programmierer, der bereits Erfahrungen bei der Entwicklung von Java-Programmen für die Java 2 Standard Edition (J2SE) gesammelt hat, kann diese teilweise auf die Java-Karte übertragen. Allerdings muss bei der Programmierung der Java-Karte auf viele gewohnte Funktionen verzichtet werden. So stehen nur die primitiven Datentypen *boolean*, *byte* und *short* zur Verfügung. Der Datentyp *int* ist optional und wird deshalb nur von einigen wenigen Java-Karten unterstützt. Datentypen wie *long*, die Gleitkommazahlen *float* und *double*, sowie *char* und alle darauf aufbauenden Klassen wie *String* sind auf Java-Karten nicht verfügbar. Aus den verfügbaren primitiven Datentypen lassen sich eindimensionale Arrays erzeugen, mehrdimensionale Arrays sind nicht möglich. Auch das Klonen von Objekten wird nicht unterstützt. Objektorientierte Programmierung mit Hilfe von Klassen und Interfaces sowie deren Vererbung und Implementierung ist auch auf der Java-Karte möglich. Jedoch gibt es eine Beschränkung auf maximal 15 Interfaces pro Klasse (inklusive denen der Superklassen), und auch die Anzahl der Klassen pro Java-Package ist beschränkt. Ebenso beschränkt ist die maximale Anzahl an statischen Feldern und Methoden pro Klasse (jeweils maximal 255). Die vollständige Liste der Beschränkungen findet sich in der Spezifikation der JVM für Java-Karten [Sun 03, Abschnitt 2.2.4].

Wie im Abschnitt Aufbau (2.2.3) beschrieben, besteht der Speicher der Java-Karte zum großen Teil aus einem nicht flüchtigen Speicher. Dieser Speicher enthält den Programmcode aller auf der Karte installierten Anwendungen sowie deren Daten. Die Variablen und Objekte einer Anwendung behalten dadurch ihren Zustand, auch wenn die Java-Karte zwischenzeitlich aus dem Kartenlesegerät entfernt wurde. Über Transaktionen lassen sich mehrere Befehle zu einer atomaren Operation zusammenfassen. Transaktionen werden benötigt um inkonsistente Zustände zu verhindern, z. B. wenn der Benutzer die Ausführung einer Funktion durch Entfernen der Karte aus dem Kartenlesegerät abbricht.

Das automatische Freigeben von nicht mehr benötigten Speicherbereichen durch den „garbage collector“ ist eine weitere Funktion, die auf einer Java-Karte nicht zur Verfügung steht. Nicht mehr verwendeter Speicher wird erst wieder freigegeben, wenn diejenige Karten-Anwendung gelöscht wird, die den Speicher angefordert hat. Seit der Java Card Version 2.2 existiert eine optionale Erweiterung, die nicht mehr verwendeten Speicher freigeben kann. Durch das Fehlen von nebenläufigen Ausführungssträngen (Threads) auf der Java-Karte muss diese Funktion im Programmcode der Karten-Anwendung bei Bedarf aufgerufen werden.

2.2.6 APDU-Kommunikation mit einer Chipkarte

Der Datenaustausch zwischen einer Karten-Anwendung auf der Chipkarte und einem Anwendungsprogramm auf dem Computer erfolgt über kleine, maximal 256 Byte große Datenpakete. Diese Application Protocol Data Unit (APDU) genannten Pakete besitzen ein im Standard ISO 7816-4 vorgegebenes Format [7816-4, Abschnitt 5.1]. Eine APDU, die vom Anwendungsprogramm zur Chipkarte gesendet wird, nennt man Kommando-APDU. Auf jede Kommando-APDU sendet die Chipkarte eine Antwort-APDU zurück. Der komplette Aufbau einer Kommando-APDU zeigt Abbildung 2.2. Das Klassifikations-Byte (CLA) gibt an, ob es sich um ein im ISO-Standard definiertes Kommando handelt, beispielsweise für den Zugriff auf Speicher- oder Signaturkarten. Neben solchen speziellen Kommandos gibt es zwei Wertebereiche (`0x80` bin `9F` und `B0` bis `CF`) die von Karten-Anwendungen frei verwendet werden dürfen.

Das Befehls-Byte (INS) codiert, welche Funktion in der Karten-Anwendung ausgeführt werden soll. Über die beiden Parameter-Bytes (P1 und P2) können zusätzlich zwei Funktionsparameter angegeben werden. Die Zuordnung von Werten und Parametern zu bestimmten Funktionen kann frei vom Programmierer der Karten-Anwendung gewählt werden. Im optionalen Daten-Teil des APDU lassen sich Datenblöcke von bis zu 247 Byte Länge auf die Karte übertragen. Die Länge des Datenblocks wird im Lc-Feld (Lc = „Length command“) vermerkt. Soll ein Block

übertragen werden, der mehr als 247 Byte umfasst, müssen die Daten auf mehrerer APDUs aufgeteilt werden und auf der Karte wieder zusammengesetzt werden. Im Le-Feld (Le = „length expected“) wird vermerkt, wie groß die erwartete Länge einer Antwort-APDU sein muss. Eine solche Antwort-APDU enthält neben den Antwort-Daten das sogenannte Status-Wort (SW), das aus zwei Bytes besteht. Es gibt an, ob der durch die Kommando-APDU gewählte Befehl vollständig ausgeführt wurde (SW = `0x9000`) oder welcher Fehler dabei auftrat. Die Liste mit möglichen Fehlerwerten ist im Standard ISO 7816-4 definiert [7816-4, Abschnitt 5.1.3].

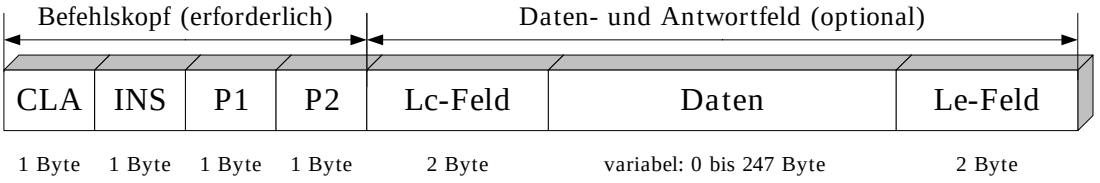

Abbildung 2.2: Datenformat einer Kommando-APDU nach ISO 7816-4

2.2.7 RMI-Kommunikation mit einer Java-Karte

Seit der Einführung der Java Card 2.2 gibt es eine einfachere Möglichkeit, Befehle und Daten zwischen Karten-Anwendung und Anwendungsprogramm auszutauschen. Dabei wird das Konzept des entfernten Methodenaufrufs verwendet, das in Java als Remote Method Invocation (RMI) bekannt ist. Dieses aus der Netzwerkprogrammierung stammende Konzept wurde auf die Java-Karte übertragen und angepasst.

2.2.7.1 Remote Method Invocation

Die Remote Method Invocation basiert auf dem Client - Server Konzept. Der Server stellt eine bestimmte Funktionalität zur Verfügung, indem er eine vorher definierte Schnittstellenbeschreibung (*Interface*) in einer Klasse implementiert. Eine Registrierungs-Instanz auf dem Server verwaltet eine Liste aller über RMI verwendbaren Klassen. Dem Client ist die Schnittstellenbeschreibung und die Netzwerkadresse bekannt, unter der die Registrierungs-Instanz erreichbar ist. Um Funktionen auf dem Server aufzurufen, generiert RMI auf dem Client automatisch ein Verbindungs-Objekt (auch Proxy oder Stub genannt), das aus der Sicht des Clients die Schnittstellenbeschreibung des Servers implementiert. Ruft der Client eine Funktion dieses Objektes auf, werden alle für den Aufruf nötigen Daten in ein übertragbares Format konvertiert. Dazu werden Funktions-Parameter wie Zahlen oder die Daten von Java-Objekten automatisch in einen Byte-Strom umgewandelt (serialisiert). Der serialisierte Funktionsaufruf wird über das Netzwerk an den Server gesendet, auf dem er von einem automatisch generierten Objekt (Skeleton) wieder deserialisiert und ausgeführt wird. Damit übertragene Objekte wieder deserialisiert werden können, müssen die Klassen der Objekte sowohl auf dem Client wie auch auf dem Server

vorhanden sein. Das Ergebnis der ausgeführten Funktion wird analog zum Aufruf vom Server serialisiert, an den Client übertragen, dort deserialisiert und dem Client-Programm übergeben, das die Server-Funktion aufgerufen hat. Der große Vorteil von RMI ist die Transparenz für den Programmierer. Aus Sicht des Client-Programmes besteht kein Unterschied zwischen einem lokalen Aufruf und dem Aufruf einer Server-Funktion.

2.2.7.2 Java Card Remote Method Invocation

Überträgt man das Konzept des entfernten Methodenaufrufs auf die Java-Karte, stellt das Anwendungsprogramm den RMI-Client dar, der Funktionen auf der Java-Karte (Server) aufruft. Beim Java Card Remote Method Invocation (JC-RMI) genannten Verfahren erfolgt die Übertragung der Befehle statt über ein Netzwerk über die APDUs. Im Gegensatz zum normalen RMI gibt es kaum Gemeinsamkeiten zwischen den Ausführungsumgebungen auf dem Client und dem Server. Einzig die primitiven Datentypen, die in Abschnitt 2.2.5 vorgestellt wurden, existieren unverändert in beiden Umgebungen. Es existieren keine Klassen, die identisch auf der Java-Karte und im Anwendungsprogramm vorliegen. Ohne identische Klassen funktioniert jedoch die aus RMI bekannte automatische Serialisierung und Deserialisierung von übertragenen Objekten nicht. Will man Objekte auf die Karte übertragen oder von ihr zurückgeben, muss man deshalb alle benötigten Daten selbst in einen Byte-Strom umwandeln (serialisieren). Um beispielsweise einen öffentlichen RSA-Schlüssel zu übertragen, muss man den öffentlichen Exponenten und den RSA-Modulus auslesen und in einen Bytestrom umwandeln. Auf der Empfängerseite muss die verwendete Kodierung der Serialisierung bekannt sein, um Exponent und Modulus aus dem Bytestrom zu extrahieren und damit das passende RSA-Schlüssel-Objekt zu erzeugen.

2.2.8 Sicherheit der Java Card

Die Verbesserung der Sicherheit auf Chipkarten war eines der Hauptziele bei der Entwicklung der Java-Karte. Die Programmiersprache Java besitzt bereits einige Eigenschaften, die sich positiv auf die Sicherheit der damit entwickelten Anwendungen auswirken. Dazu zählt das Fehlen von direkten Speicherzugriffen, z. B. über Zeiger (Pointer), und die Eigenschaft, dass jedes Java-Objekt einem bestimmten Datentyp angehört (strenge Typisierung). Darüber lassen sich fehlerhafte Zuweisungen abfangen, bei denen die Datentypen von Quelle und Ziel inkompatibel sind (Typensicherheit). Außerdem prüft Java die Grenzen von Speicherbereichen und verhindert so Buffer-Overflow-Angriffe.

Diese positiven Eigenschaften gelten auch für die Java-Karte. Allerdings ist es nicht möglich, die Überprüfung des Java Bytecodes zur Laufzeit durch den Bytecode-Verifier auf der Java-Karte vorzunehmen, wie das bei J2SE der Fall ist. Der

Bytecode-Verifier überprüft dort vor der Ausführung den gesamten Java Bytecode auf Fehler in der Typisierung und fehlerhafte Speicherreferenzen. Die für diese Codeanalyse nötigen Algorithmen lassen sich mit dem auf der Java-Karte zur Verfügung stehenden Speicher nicht durchführen. Die Überprüfung des Bytecodes erfolgt deshalb außerhalb der Java-Karte. Es gibt erste Ansätze, die eine vollständige oder zumindest teilweise Überprüfung des Bytecodes auf der Java-Karte ermöglichen [Lero01].

Die Java-Karte ist eine Multi-Anwendungs-Chipkarte (MACOS). Sie erlaubt es, mehrere Karten-Anwendungen gleichzeitig auf einer Karte zu installieren. Um die Datenintegrität und Vertraulichkeit der einzelnen Anwendungen zu gewährleisten, existiert eine Anwendungs-Firewall, die den Zugriff auf Daten und Programme anderer Karten-Anwendungen verhindert. Jede Anwendung erhält so einen privaten Speicherbereich, auf den nur sie zugreifen kann. Deshalb kann man von außerhalb einer Karten-Anwendung nur solche Daten lesen oder schreiben, für die es Funktionen zum Auslesen oder Ändern gibt. Damit hängt die Sicherheit der Daten einer Karten-Anwendung nur von der Programmierung ab. Einige Daten sind so sensibel, dass es keine Funktionen geben darf, diese auszulesen. Für andere kann z. B. die Karten-Anwendung vor dem Auslesen eine Authentifikation des Benutzers über seine PIN fordern.

Neben dieser anwendungsinternen Sicherheit muss auch noch die anwendungsübergreifende Sicherheit gewährleistet werden. Das betrifft insbesondere das Schutzziel Verfügbarkeit. Es muss z. B. verhindert werden, dass ein Angreifer eine wichtige Anwendung und deren gespeicherte Daten auf der Java-Karte löscht. Die Java Card-Spezifikation verweist dazu auf den „GlobalPlatform Standard“[4]. Dieser, ehemals als „Visa OpenPlatform“ bekannte Standard, umfasst die gesamte Verwaltung der Karte und aller Anwendungen, die auf ihr installiert sind. Jede GlobalPlatform kompatible Chipkarte enthält eine vorinstallierte Karten-Anwendung, den *CardManager*, der es erlaubt Karten-Anwendungen auf die Karte zu übertragen oder bestehende zu löschen. Der Zugriff auf den CardManager ist erst nach einer erfolgreichen Authentifikation möglich. Dies kann durch Besitz eines symmetrischen Schlüssels (DES, 3DES, AES) oder eines privaten asymmetrischen Schlüssels (RSA, DSA) erfolgen [HaNS02, S.59f]. Nach mehreren fehlgeschlagenen Authentifikationsversuchen wird der CardManager dauerhaft gesperrt. Andere Karten-Anwendungen, die sich bereits auf der Chipkarte befinden, sind nicht davon betroffen. Durch die Sperrung der CardManager-Anwendung ist es jedoch nie mehr möglich, Karten-Anwendungen auf dieser Chipkarte zu installieren oder zu löschen.

[4] `http://www.globalplatform.org`

2.3 MOBILE-Projekt

Das Projekt „MOBILE - Sichere Dienste für mobile Bürger“ wurde in den Jahren 2001 bis 2003 am Fraunhofer-Institut für Sichere Informations-Technologie (SIT) in Zusammenarbeit mit dem Fraunhofer-Institut für Graphische Datenverarbeitung (IGD) und dem Fraunhofer-Institut für Integrierte Publikations- und Informationssysteme (IPSI) entwickelt. Ziel des MOBILE-Projektes war es, eine Plattform zu entwickeln, „die ein mehrseitig sicheres, dynamisches und individuelles Angebot an orts-, zeit- und kontextabhängigen Diensten im täglichen Einsatz ermöglicht“. [Hoff04, S.9]

Im MOBILE-Projekt besitzt jeder Benutzer einen persönlichen Assistenten, der auf einem zentralen System, der sogenannten „Homebase“ ausgeführt wird. Der Assistent verfügt über einen Datenspeicher, in dem die verschiedensten Daten über seinen „Besitzer“ gespeichert sind. Der Benutzer kann dem Assistenten Aufgaben zuweisen, die dieser dann unter Verwendung verschiedener Dienste ausführt. Neue Aufgaben oder die Ergebnisse vergangener Aufgaben kann der Benutzer über eine Weboberfläche abrufen. Diese wird dynamisch, unter Berücksichtigung der Bildschirmgröße und anderer Eigenschaften des verwendeten Endgerätes erzeugt.

Das MOBILE-Projekt bildet den Rahmen für das Identitätsmanagement-System, das in dieser Arbeit entwickelt wird. Deshalb folgt eine genauere Vorstellung des Designs des MOBILE-Projektes. Außerdem wird die existierende prototypische Implementierung, der „Demonstrator“, auf Probleme untersucht.

Die Plattform untergliedert sich in drei Bereiche: Der erste Bereich beinhaltet das mobile Endgerät des Nutzers. Dieses ist verantwortlich für die Darstellung der Benutzungsoberfläche und bietet die Möglichkeit über geeignete Ortungstechnik die gegenwärtige Position des Gerätes zu ermitteln. Der zweite Bereich umfasst die Homebase, die über eine permanente Internetverbindung verfügt. Der letzte Bereich besteht aus einem zur Plattform gehörenden Ontologie-Dienst sowie weiteren externen Diensten, die von den Assistenten der Benutzer verwendet werden.

Jeder persönliche Assistent bearbeitet Aufgaben für seinen Benutzer. Dafür stehen dem Assistenten eine große Zahl von unterschiedlichen externen Diensten zur Verfügung. Je nach Komplexität der Aufgabe verwendet der Assistent ein oder mehrere Dienste, um die ihm gestellte Aufgabe auszuführen. Oftmals existieren mehrere Dienste, die ähnliche oder gleiche Dienstleistungen anbieten. Übergibt der Benutzer dem Assistenten beispielsweise die Aufgabe, eine Dienstreise zu buchen, hat der Assistent viele Möglichkeiten, diese Aufgabe umzusetzen, angefangen mit der Art des Transportmittels (Zug, PKW, Flugzeug usw.) bis hin zur Wahl eines passenden Hotels. Um dem Assistenten die Bewertung und den Vergleich der verschiedenen

Dienste zu ermöglichen, kann der Benutzer verschiedene Angaben über seine Vorlieben machen. Die so gesammelten Daten bleiben auch nach Beendigung der Aufgabe in der Homebase gespeichert. Der Assistent kann so im Laufe der Zeit auf ein immer genaueres Benutzerprofil zurückgreifen und der Benutzer muss die benötigten Daten nicht immer wieder für jede Aufgabe neu eingeben.

Durch die Einbindung der externen Dienste ist das MOBILE-Projekt sehr flexibel in der Art und dem Umfang der Aufgaben, die das System ausführen kann. Diese Flexibilität bringt aber auch ein großes Problem mit sich. Es existieren keine Standards, für die von den verschiedenen Diensten genutzte Semantik. Jeder Dienst besitzt in der Regel ein eigenes „Vokabular, mit dem die Objekte einer Domäne, d.h. ihre Begriffe und Individuen, bezeichnet werden“ [GHHM+04, S.8]. Will man nun verschiedene Dienste miteinander vergleichen oder kombiniert verwenden, benötigt man einen Übersetzer, der die Unterschiede zwischen den verschiedenen Diensten kompensiert. Dies ist die Aufgabe des Ontologie-Dienstes. Dieser ermöglicht es verschiedene Begriffe mit identischer Bedeutung zu erkennen und so die Ergebnisse der verschiedenen Dienste zu interpretieren und zu vergleichen.

Um ortsabhängige Dienste, die sogenannten „Location-based Services“ zu ermöglichen, können die Endgeräte der Homebase ihre aktuelle Position mitteilen. Diese wird von der Homebase genutzt, um in der näheren Umgebung für den Benutzer interessante Orte wie Restaurants, touristische Ziele oder Haltestellen zu ermitteln. Der Benutzer kann sich von der Homebase Kartenausschnitte erzeugen lassen, auf denen solch interessante Orte markiert sind. Das ermöglicht dem Benutzer, sich auch in einer unbekannten Umgebung schnell einen Überblick zu verschaffen.

Vom MOBILE-Projekt existiert ein Demonstrator[5], der nur einige wenige der beschriebenen Funktionen umsetzt. So fehlt beispielsweise die Anbindung an externe Dienste. Außerdem werden alle Daten der Homebase in einer unverschlüsselten XML-Datei gespeichert. Diese zentrale und ungesicherte Verwaltung der persönlichen Benutzerdaten bietet ein lohnenswertes Ziel für einen Angreifer. Bedenklich ist auch die gewählte Realisierungsform für die Übermittlung der Positionsdaten der Endgeräte an die Homebase. Auf dem Endgerät wird dafür eine Software installiert, die in regelmäßigen Abständen die aktuelle Position ermittelt und an die Homebase sendet. Dadurch ist die Homebase in der Lage Bewegungsprofile aller Benutzer zu erzeugen. Sinnvoller wäre eine Übermittlung der Koordinaten nur bei Bedarf und mit einer Genauigkeit in Abhängigkeit vom geplanten Einsatz der Positionsdaten.

Die Homebase des Demonstrators basiert auf der Plattform Secure Mobile Agent (SeMoA). Diese ermöglicht die Nutzung von sicheren und mobilen Agenten. Dahinter

[5] `https://pc-nixdorf.sit.fhg.de:50001/mobile`

steht die Idee, nicht die Daten zum Programm zu transferieren sondern die Programme (Agenten) zu den Daten. Nur die bearbeiteten Ergebnisse des Programms werden am Ende zu einem Empfänger gesendet. Der Einsatz von Agenten ermöglicht es, flexibel große Mengen von Daten zu bearbeiten. Der lokale Zugriff durch einen Agenten spart das Übertragen der „Rohdaten“, die sehr umfangreich ausfallen können. Zudem benötigt der lokale Zugriff auf Daten gegenüber einem entfernten Zugriff über ein Netzwerk wesentlich weniger Zeit, da die Verzögerung durch die Netzwerkübertragung wegfällt. Dem gegenüber stand von Anfang an das Problem der Sicherheit des mobile Codes der Agenten und der Umgebung, in der er ausgeführt wird. Genaueres zu den Sicherheitsbedrohungen findet sich unter [Ecke02, S.51ff]. Wegen dieser Sicherheitsprobleme konnten sich Agentensysteme bisher nicht durchsetzen.

3. Design

Der erste Schritt bei der Entwicklung einer neuen Anwendung oder der Erweiterung einer bestehenden Anwendung ist die Untersuchung der benötigten Technologien und der Vergleich mit verwandten Produkten. Dieses wurde im vorausgehenden Kapitel durchgeführt. Der nächste Schritt ist der Entwurf eines Software-Designs unter Berücksichtigung der im ersten Schritt gewonnenen Erkenntnisse.

Das im Folgenden dargestellte Software-Design basiert auf dem in Abschnitt 2.3 vorgestellten MOBILE-Projekt. Dieses wird in diesem Kapitel an vielen Stellen überarbeitet und erweitert. Das primäre Ziel bei der Überarbeitung ist es, dem Benutzer die Kontrolle über seine Daten zu ermöglichen, obwohl sie in der Homebase gespeichert sind. Dies geschieht durch Verschlüsseln der Daten. Die dabei verwendeten kryptographischen Schlüssel werden auf Java-Karten gespeichert, die dem Benutzer gehören. Der Benutzer kontrolliert die Java-Karten und die auf diesen gespeicherten kryptographischen Schlüssel und kann damit den Zugriff auf seine Daten kontrollieren.

Ein weiteres Ziel bei der Überarbeitung ist die Vermeidung der Schwachstellen und Probleme, die der Demonstrator aufgezeigt hat. So wird die Kommunikation zwischen dem Endgerät und der Homebase komplett neu entwickelt. Dies ist auch wegen der Integration der Java-Karte in das MOBILE-Projekt notwendig. Zusätzlich wird das Design an aktuelle technische Entwicklungen angepasst. Dazu gehört z. B. das Ersetzen der Agenten-Plattform SeMoA durch einen Anwendungsserver nach dem Standard Java 2 Enterprise Edition (J2EE). Außerdem nutzt der Assistent zum Ausführen der Aufgaben jetzt Webservices statt mobiler Agenten.

Um eine Verwechselung zwischen dem „alten MOBILE-Projekt“ und dem „neuen überarbeiteten *Mobile*-Projekt“ auszuschließen, verwendet diese Arbeit neue Be-

zeichnungen für die verschiedenen Komponenten. Aus der „Homebase“ wird so z. B. der *Mobile* Server.

Dieses Kapitel gliedert sich in in fünf Teile. Im ersten Teil werden die Anforderungen, die das neue *Mobile*-Projekt erfüllen soll, dargestellt. Im zweiten Teil folgt die Vorstellung des überarbeiteten Designs der *Mobile*-Plattform. Das entwickelte Verschlüsselungssystem, das dem Benutzer die Kontrolle über seine Daten ermöglicht, wird im dritten Teil vorgestellt. Im vierten Teil wird das Design der Softwarekomponenten vorgestellt, die auf auf dem Endgerät des Benutzers und der Java-Karte ihren Dienst verrichten. Im vierten und letzten Teil folgt dann die Vorstellung der Softwarekomponenten, die den *Mobile* Server bilden.

3.1 Anforderungen an das neue Design

Bei der Weiterentwicklung des MOBILE-Projektes zum „*Mobile*-Projekt“ wurden zwei Schwerpunkte gesetzt:

Der erste Schwerpunkt ist die Korrektur der im MOBILE-Demonstrator gefundenen Datenschutz- und Sicherheitsprobleme. Im MOBILE-Projekt werden alle Benutzerdaten unverschlüsselt in der Homebase gespeichert. Dies erfordert ein sehr großes Vertrauen des Benutzers sowohl in den Betreiber der Homebase, als auch in die Sicherheit der Homebase. Der Betreiber hat beispielsweise vollständigen Zugriff auf alle in der Homebase gespeicherten Benutzerdaten. Die Eigentümer dieser Daten können nicht kontrollieren, was der Betreiber mit ihren Daten macht. Durch fahrlässigen Umgang mit den Benutzerdaten, kriminelle Mitarbeiter oder Sicherheitslücken in der verwendeten Software gelangen immer wieder Benutzerdaten in falsche Hände. Die erste und wichtigste Anforderung betrifft deshalb die auf dem *Mobile* Server gespeicherten persönlichen Daten der Benutzer. Diese sollen so verschlüsselt werden, dass nur der Benutzer sie entschlüsseln kann. Dies schützt die Daten vor unberechtigtem Auslesen. Die Benutzer greifen, wie im MOBILE-Projekt, über eine webbasierte Benutzungsschnittstelle auf die *Mobile*-Plattform zu. Für die Authentifikation der Benutzer gegenüber der Plattform sollen Java-Karten verwendet werden, die als sicherer Speicher z. B. für geheime Schlüssel und als sichere Ausführungsumgebung für ein Programm dienen. Damit die Benutzer komfortabel mit mehreren Geräten auf die *Mobile*-Plattform zugreifen können, soll jeder Benutzer mehrere Java-Karten besitzen dürfen. Dadurch kann der Benutzer zum Einen pro Endgerät eine Java-Karte verwenden und ist nicht gezwungen, die eine Java-Karte ständig zwischen den verschiedenen Geräten zu wechseln. Zum Anderen sind nach dem Verlust einer Java-Karte durch Diebstahl oder im Falle einer defekten Java-Karte alle verschlüsselt auf dem *Mobile* Server gespeicherten Daten über eine zweite Java-Karte weiterhin zugreifbar. Bei nur eine Java-Karte pro Benutzer wären die Daten in einem solchen

Fall verloren, da sie nicht mehr entschlüsselt werden könnten. Hat ein Benutzer nun mehrere Java-Karten und verliert eine davon, soll es möglich sein die verlorene Karte zu sperren, sodass es nicht mehr möglich ist über diese Karte Zugriff auf die Benutzerdaten zu erhalten. Zusätzlich soll der Assistent auf dem *Mobile* Server, wie im MOBILE-Projekt, verschiedene Aufgaben ausführen, wobei er Zugriff auf bestimmte Benutzerdaten benötigt. Dies darf nur nach vorheriger Autorisierung durch den Benutzer möglich sein, da ansonsten der Benutzer wiederum keine Kontrolle über die Verwendung seiner Daten hat. Die zu entwickelnde *Mobile*-Plattform soll außerdem möglichst einfach zu benutzen sein, denn ein System, bei dem die Sicherheit zu Lasten der Benutzbarkeit geht, wird von den Benutzern nicht akzeptiert und damit auch nicht verwendet.

Der zweite Schwerpunkt bei der Weiterentwicklung zum *Mobile*-Projekt ist die Aktualisierung der verwendeten Technologien. Multi-Agentensysteme, wie das im MOBILE-Projekt verwendete SeMoA, haben aktuell (2005) immer noch keine nennenswerte Verbreitung erfahren. Die Technologie der „Webservices“ hingegen hat sich in den vergangenen Jahren stark verbreitet. Ein Webservice ist ein plattformunabhängiger Dienst, der von anderen Anwendungen im einem Netzwerk (z. B. dem Internet) genutzt werden kann. Neben den bekannten Anbietern von Webservices wie Amazon[1], Google[2] und eBay[3] gibt es tausende von Anbietern, welche die verschiedensten Dienste als Webservices zur Verfügung stellen. Auf der Clientseite soll die *Mobile*-Plattform, wie im MOBILE-Projekt auch, von mobilen Endgeräten genutzt werden können, beispielsweise mit einem PDA oder einem Mobiltelefon. Es soll dazu eine Software entwickelt werden, die auf dem mobilen Endgerät die Nutzung der Java-Karte ermöglicht. Außerdem soll sich die Software leicht erweitern lassen, um die Positionsinformationen des Endgerätes, die z. B. von einem angeschlossenen oder integrierten GPS-Gerät stammen, in der *Mobile*-Plattform nutzen zu können.

3.2 Gesamtsystem

Nachdem die Anforderungen an das *Mobile*-Projekt im vorangegangenen Abschnitt definiert wurden, folgt nun der Überblick über den Entwurf des Gesamtsystems, dass diese Anforderungen erfüllt. Das in diesem Abschnitt dargestellte System wurde vom Autor in Zusammenarbeit mit Dipl.-Inform. Mario Hoffmann entwickelt. In Abbildung 3.1 ist das Ergebnis dieser gemeinsamen Entwicklung zu sehen. Die Abbildung zeigt die in drei Bereiche aufgeteilte *Mobile*-Plattform: In der linken Hälfte der „Trust Domain“, befindet sich der Benutzer mit der Java-Karte, seinem Endgerät und der Client-Software (hier „Client-Proxy“ genannt). Im rechten Teil befindet

[1] `http://www.amazon.com/webservices`
[2] `http://www.google.com/apis`
[3] `http://developer.ebay.com/soap/`

sich der Server mit dem Assistenten und mehreren Datenbanken, in denen die Benutzerdaten gespeichert werden. Rechts, außerhalb der „Trust Domain“, befinden sich die externen Webservices, deren Dienste der Assistent für die Ausführung der Aufgaben in Anspruch nimmt.

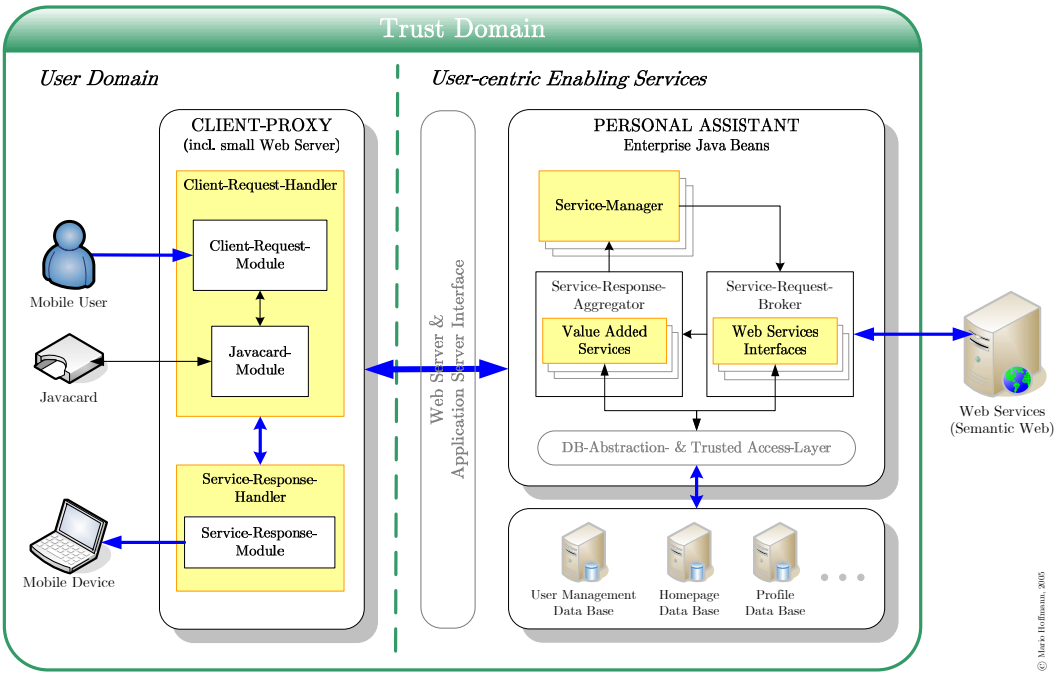

Abbildung 3.1: Überblick über das *Mobile*-Projekt (Quelle: Mario Hoffmann)

Da das detaillierte Design und die nachfolgende Implementierung aller Komponenten, die in der Abbildung dargestellt sind, den Umfang einer Diplomarbeit bei Weitem übersteigt, konzentriert sich diese Arbeit auf folgende Komponenten:

1. Die Software für die Endgeräte der Benutzer („Client-Proxy“), die im weiteren Verlauf dieser Arbeit als *Mobile* Client bezeichnet wird.
2. Die *Mobile* Java-Karte, die mit dem *Mobile* Client kommuniziert.
3. Den *Mobile* Server, der die Datenbank mit den verschlüsselten Benutzerdaten enthält und auf dem die Assistenten aller Benutzer ausgeführt werden. Der Assistent und die von ihm genutzten Webservices werden dabei nicht genauer betrachtet.

Neben der Entwicklung der einzelnen Komponenten befasst sich ein großer Teil dieser Arbeit mit der Kommunikation zwischen den Komponenten. Insbesondere die Kommunikation zwischen dem *Mobile* Client und dem *Mobile* Server ist problematisch, da es beim Zugriff mit mobilen Endgeräten einige technische Einschränkungen zu beachten gibt: Die Verbindung mit dem Internet erfolgt bei diesen Geräten entweder über das Mobilfunknetz eines Mobilfunkbetreibers oder über einen WLAN-Hotspot. In beiden Fällen bekommt der *Mobile* Client keinen direkten Zugang zum Internet, sondern befindet sich in einem privaten Netzwerk, das über ein Gateway (Mobil-

funk) bzw. einen NAT-Router (WLAN) mit dem Internet verbunden ist. Das hat zur Folge, dass vom mobilen Endgerät ohne Probleme eine Verbindung zu einem beliebigen Computer im Internet (z. B. dem *Mobile* Server) aufgebaut werden kann, umgekehrt jedoch der *Mobile* Server keine Verbindung zum Endgerät und damit zum *Mobile* Client herstellen kann. Dies gilt wohlgemerkt nur für den Verbindungsaufbau. Über eine einmal hergestellte Verbindung zwischen *Mobile* Client und *Mobile* Server können Daten in beide Richtungen übertragen werden. Bricht diese Verbindung zusammen, muss sie vom *Mobile* Client auf dem Endgerät wieder hergestellt werden.

Eine Übersicht aller in dieser Arbeit relevanten Komponenten der *Mobile*-Plattform sowie die Kommunikationswege zwischen den Komponenten zeigt Abbildung 3.2. Auf der linken Seite ist der Benutzer zu sehen, der über einen beliebigen Webbrowser auf wie webbasierte Benutzungsoberfläche zugreift. Diese besteht aus Webseiten, die vom *Mobile* Server erzeugt werden. Der im *Mobile* Client integrierte Proxy leitet diese an den Browser des Benutzers weiter und ergänzt sie um einige, im *Mobile* Client, erzeugte Webseiten. Diese im Client erzeugten Webseiten ermöglichen beispielsweise die Eingabe und clientseitige Verschlüsselung von Benutzerdaten. Erst die verschlüsselten Daten werden dann über die Webservice-Schnittstelle an den *Mobile* Server übertragen. Über diese Schnittstelle kann der *Mobile* Client Funktionen auf dem *Mobile* Server ausführen und so beispielsweise Daten aus der Datenbank auslesen oder in ihr abspeichern.

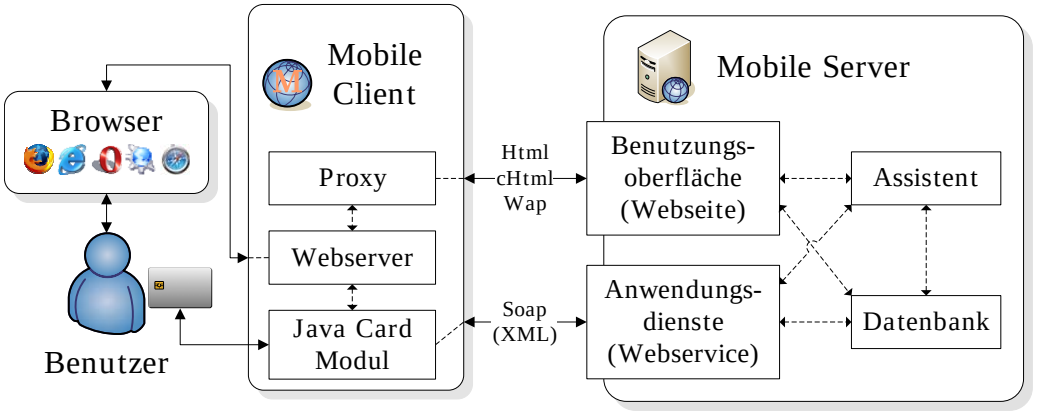

Abbildung 3.2: Kommunikationswege zwischen Benutzer, Client und Server

Nachdem das Design des Gesamtsystems ausgearbeitet ist, fehlt nun noch die wichtigste Neuerung des *Mobile*-Projektes, die Verschlüsselung der Benutzerdaten, die dem Benutzer die Kontrolle über seine Benutzerdaten ermöglicht. Der nun folgende Abschnitt beschäftigt sich mit der Entwicklung eines passenden Verschlüsselungssystems.

3.3 Das Verschlüsselungssystem

Um die Daten der Benutzer auf dem *Mobile* Server vor unberechtigtem Zugriff zu schützen, sollen diese verschlüsselt abgespeichert werden. Die dabei gestellten Anforderungen wie mehrere Java-Karten pro Benutzer, die vollständige Kontrolle des Benutzers über seine Daten und die zusätzliche Möglichkeit, dass der Assistent auf dem *Mobile* Server Zugriff auf bestimmte Benutzerdaten erhält, werden von keinem, dem Autor bekannten, existierenden Verschlüsselungssystem erfüllt. Es ist deshalb notwendig, ein eigenes Verschlüsselungssystem mit den gewünschten Eigenschaften zu entwickeln.

Die Entwicklung des Verschlüsselungssystems erfolgt in mehreren Schritten. Angefangen mit einer sehr einfachen Lösung mit nur einer Java-Karte werden in jedem Schritt mehr Anforderungen berücksichtigt. Diese iterative Vorgehensweise hat den Vorteil, dass man bei der Entwicklung des Systems nach dem Prinzip „Keep it simple“ vorgehen kann. In jedem Entwicklungsschritt werden dazu die zu erreichenden Ziele mit möglichst geringem Aufwand umgesetzt. Ein geringer Aufwand bedeutet ein einfacheres und überschaubareres System, in dem sich Fehler und Schwachstellen leichter finden lasen. Dadurch erhöht sich die erreichte Sicherheit des entwickelten Systems.

Die bei der Entwicklung des Verschlüsselungssystems für das *Mobile*-Projekt durchlaufenen Schritte werden in den folgenden Abschnitten beschrieben.

3.3.1 Verschlüsselung von Daten mit Java-Karten

Für die Verschlüsselung von Daten beliebiger Größe wird üblicherweise ein symmetrischer kryptographischer Algorithmus verwendet, beispielsweise Triple DES (3DES) oder der Advanced Encryption Standard (AES). Diese ver- und entschlüsseln Daten wesentlich schneller als asymmetrische kryptographische Algorithmen wie RSA. Das einfachste Verschlüsselungssystem besteht deshalb aus der Java-Karte, in der die symmetrischen Schlüssel erzeugt und gespeichert werden. Die Verschlüsselung und Entschlüsselung der Daten übernimmt die Java-Karte, damit die Schlüssel nie die sicherere Umgebung der Java-Karte verlassen müssen. Die verschlüsselten Daten können dann beliebig außerhalb der Java-Karte gespeichert werden, beispielsweise in einer Datenbank auf dem *Mobile* Server.

Ein wenig mehr Aufwand muss getrieben werden, wenn jeder Benutzer mehr als nur eine Java-Karte besitzen können soll. Dazu wird ein symmetrischer Master-Schlüssel eingeführt, der auf jeder Java-Karte gespeichert ist. Die bereits verwendeten symmetrischen „Daten-Verschlüsselungs-Schlüssel“ (Datenschlüssel) bleiben bestehen. Sie werden jedoch mit dem neuen Master-Schlüssel verschlüsselt. Dadurch können die

verschlüsselten Datenschlüssel zusammen mit den verschlüsselten Daten außerhalb der Karte gespeichert werden. Die Ver- und Entschlüsselung ist damit zweistufig: Zuerst wird der verschlüsselte Datenschlüssel in die Java-Karte geladen und entschlüsselt. Dann kann die Java-Karte wie bisher genutzt werden um Daten zu ver- oder entschlüsseln.

Jetzt kann jeder Benutzer zwar mehrere Karten verwenden, aber es ist nur mit großem Aufwand möglich, einzelne Java-Karten nachträglich zu sperren. Jede Java-Karte, in der das Master-Passwort gespeichert ist, kann die verschlüsselten Datenschlüssel entschlüsseln und dadurch auch die Benutzerdaten entschlüsseln. Verliert der Benutzer eine Karte und will diese vom Zugriff auf seine Daten aussperren, bleibt nur die Möglichkeit den Master-Schlüssel zu ändern. Alle verbleibenden Java-Karten des Benutzers müssen dafür mit dem neuen Master-Schlüssel aktualisiert werden und ebenso alle Datenschlüssel. Für die Aktualisierung der Datenschlüssel werden diese mit dem alten Master-Schlüssel entschlüsselt und mit dem neuen Master-Schlüssel wieder verschlüsselt. Der Aufwand für den Benutzer bei einer solchen Änderung des Master-Schlüssels steigt mit der Anzahl der Java-Karten, die er besitzt. Der Aufwand für die Neuverschlüsselung der Datenschlüssel hingegen hängt von der Anzahl der verwendeten Datenschlüssel ab. Durch die relativ geringe Geschwindigkeit, mit der eine Java-Karte kryptographische Operationen durchführt, kann diese Neuverschlüsselung der Datenschlüssel mehrere Minuten dauern, je nach dem, wie viele Datenschlüssel bearbeitet werden müssen. Wegen dem hohen Aufwand für den Benutzer ist zu befürchten, dass viele Benutzer verlorene Karten nicht sperren, da ihnen das Verfahren zu umständlich und zeitraubend ist. Deshalb kommt im *Mobile*-Projekt ein für den Benutzer einfacheres Verfahren zu Einsatz:

Der symmetrische Master-Schlüssel, der identisch in jeder Java-Karte eines Benutzers gespeichert ist, wird durch asymmetrische Schlüsselpaare ersetzt, für jede Java-Karte eines. Wegen der Bindung von Java-Karte und Schlüsselpaar werden die einzelnen Schlüssel des Schlüsselpaares in dieser Arbeit als „Kartenschlüssel“ bezeichnet. Die Kombination aus einem symmetrischen und einem asymmetrischen Verschlüsselungsalgorithmus ist nicht unüblich und wird als „hybride Verschlüsselung“ bezeichnet. Der private Kartenschlüssel wird auf der Java-Karte generiert und verlässt diese nie. Der öffentliche Kartenschlüssel hingegen wird allen anderen Java-Karten eines Benutzer bekannt gemacht. Jeder Datenschlüssel wird, statt wie bisher mit dem Master-Schlüssel, für jede Java-Karte mit deren öffentlichem Kartenschlüssel verschlüsselt. Es entstehen dadurch für jeden Datenschlüssel genau so viele verschlüsselte Kopien, wie der Benutzer Java-Karten besitzt. Das Sperren einer Java-Karte lässt sich jetzt in zwei Schritten durchführen. Zuerst wird der öffentliche Kartenschlüssel der zu sperrenden Karte möglichst aus allen Java-Karten des Benutzers

gelöscht. Danach können alle Datenschlüssel gelöscht werden, die mit dem öffentlichen Kartenschlüssel der Karte verschlüsselt sind, die gesperrt werden soll. Der Aufwand ist verglichen mit der Änderung des Master-Schlüssels ein wenig geringer, da nicht alle Schlüssel neu verschlüsselt werden müssen. Zusätzlich ist der Benutzer in diesem Fall nicht gezwungen alle Karten sofort zu aktualisieren, da diese auch weiterhin funktionieren. Eine wesentlich genauere Beschreibung, welche Schritte für das Sperren eine Java-Karte notwendig sind, folgt in Abschnitt 3.3.4.2.

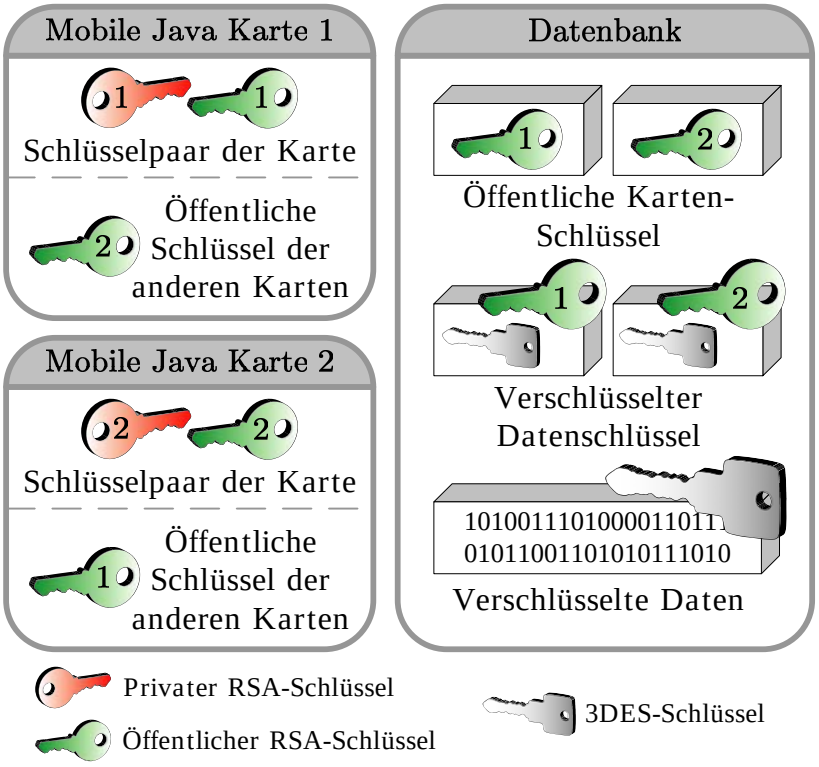

Abbildung 3.3: Schlüsselverteilung für einen Benutzer mit zwei aktiven Karten

Ein Beispiel für die Verteilung der verschiedenen Schlüssel eines Benutzers mit zwei *Mobile* Java-Karten zeigt die Abbildung 3.3. Das Beispiel verwendet als symmetrischen kryptographischen Algorithmus den Verschlüsselungsalgorithmus „Triple DES“ (auch unter den namen 3DES oder DESede bekannt) und als asymmetrischen kryptographischen Algorithmus den Verschlüsselungsalgorithmus „RSA“. Im rechten Teil befindet sich die Datenbank, die auf dem *Mobile* Server liegt. Sie enthält einen mit einem Datenschlüssel verschlüsselten Datenblock (unten). Zwei Kopien des verwendeten Datenschlüssels befinden sich darüber (Mitte). Diese sind mit den öffentlichen Schlüsseln der *Mobile* Java-Karte 1 bzw. 2 verschlüsselt. Erzeugt werden die Datenschlüssel von den *Mobile* Java-Karten. Die Verschlüsselung des Datenschlüssels erfolgt auch auf der Karte. Dazu besitzt die Karte eine Liste aller öffentlichen Schlüssel, der aktiven Karten eines Benutzers. In diesem Beispiel enthält die *Mobile* Java-Karte 1, neben ihrem eigenen Schlüsselpaar, eine Kopie des öffentlichen Schlüssels der *Mobile* Java-Karte 2. In der Datenbank ist zusätzlich ei-

ne Liste aller öffentlichen Schlüssel hinterlegt. Wird eine Karte gesperrt, oder eine neue Karte dem System hinzugefügt, können alle Karten ihre internen Kopien der öffentlichen Schlüssel über diese Liste in der Datenbank aktualisieren.

3.3.2 Authentizität der öffentlichen Kartenschlüssel

Bei der Verwendung eines asymmetrischen kryptographischen Systems, wie dem RSA-Kryptosystem, gibt es zwei sicherheitskritische Punkte, die beachtet werden müssen. Zuerst muss sichergestellt werden, dass der private Schlüssel so abgespeichert wird, dass dieser nur nach erfolgreicher Authentifikation des Eigentümers verwendet werden kann. Dies wird im *Mobile* System erreicht, indem das RSA-Schlüsselpaar auf der Java-Karte erzeugt wird und der private Schlüssel diese nie verlässt. Zudem ist der Zugriff auf die Java-Karte erst nach einer erfolgreichen Authentifikation des Benutzers möglich. Der zweite, ebenso wichtige Punkt ist die Sicherstellung der Authentizität des öffentlichen Schlüssels. Üblicherweise wird dafür eine Public Key Infrastruktur (PKI) verwendet, die über Zertifikate die Authentizität eines öffentlichen Schlüssels sicher stellt. Dabei kommt ein hierarchisches Vertrauensmodell zum Einsatz. In diesem Fall ist eine PKI ungeeignet für die Sicherstellung der Authentizität der öffentlichen Schlüssel. Der *Mobile* Server müsste dafür um eine eigene PKI erweitert werden. Diese PKI könnte über Zertifikate die Authentizität der öffentlichen Schlüssel nachprüfbar machen. Damit würde der Benutzer jedoch wieder einen Teil seiner Selbstbestimmung verlieren, denn die Authentizität der öffentlichen Schlüssel würde auf dem Vertrauen in diese PKI basieren. Auch die Möglichkeit eine von der *Mobile*-Plattform unabhängige PKI zu verwenden erscheint unpraktikabel, denn Vorgänge, wie die Revokation von Zertifikaten, müssten vom Benutzer von Hand durchgeführt werden. Als dritte Möglichkeit könnte jeder Benutzer eine eigene private PKI verwenden. Der damit verbundene Aufwand überwiegt jedoch den Nutzen, selbst wenn man auf einen umfassenden Schutz der Certificate Authority (CA) verzichtet. Zudem erfordert der Betrieb einer eigenen PKI ein hohes Maß an Fachwissen, was von einem normalen Benutzer nicht vorausgesetzt werden kann.

Deshalb kommt ein eigenes Verfahren zum Einsatz, das es erlaubt, die Authentizität der öffentlichen Schlüssel der *Mobile* Java-Karten zu prüfen. Dieses Verfahren benötigt weder die Installation einer zusätzlichen Software noch das Vertrauen in ein nicht vom Benutzer kontrolliertes System. Dazu wird eine „symmetrische Signatur“ verwendet, die aus einer parametrisierten, kryptographischen Prüfsumme (HMAC) besteht. Der Parameter, ohne den sich die Prüfsumme weder erzeugen noch verifizieren lässt, wird aus einem Passwortsatz gebildet, den der Benutzer bei der Initialisierung der ersten *Mobile* Java-Karte wählt. Dieser Passwortsatz wird auf

jeder *Mobile* Java-Karte gespeichert, sodass dieser nur zum Initialisieren einer neuen Java-Karte benötigt wird.

3.3.3 Der Offline-Zugriff über Tickets

Das vorgestellte Verschlüsselungssystem benötigt für jede Verschlüsselung oder Entschlüsselung die *Mobile* Java-Karte. Diese befindet sich in dem mobilen Endgerät des Benutzers. Trotz der fast flächendeckenden Verfügbarkeit der Mobilfunknetze gibt es Situationen, in denen eine zuverlässige Kommunikation nicht möglich ist (z. B. durch Funklöcher). Eine dauerhafte Verbindung zwischen dem *Mobile* Server mit der Datenbank und dem Endgerät kann deshalb nicht vorausgesetzt werden. Da der Assistent auf dem Server autonom die gestellten Aufgaben durchführt, kann es passieren, dass zu der Zeit, wenn der Zugriff auf Benutzerdaten benötigt wird, die Verbindung zum Endgerät des Benutzers unterbrochen ist. Der Assistent kann in einer solchen Situation die ihm gestellte Aufgabe nicht durchführen, da er ohne Kommunikation mit dem Endgerät und der *Mobile* Java-Karte keinen Zugriff auf die Benutzerdaten hat. Um diesen Fall zu verhindern muss das bisherige System so erweitert werden, dass der Assistent auf alle für eine Aufgabe benötigten Daten zugreifen kann, auch wenn keine Verbindung zu einer der Java-Karten des Benutzers besteht (der Benutzer „offline“ ist).

Eine Voraussetzung für eine solche Erweiterung ist, dass es möglich ist, vor der Ausführung einer Aufgabe zu ermitteln, welche Benutzerdaten für die Ausführung benötigt werden. Alle von der *Mobile*-Plattform angebotenen Aufgaben müssen diese Voraussetzung erfüllen, ansonsten lassen sich diese Aufgaben nicht offline durchführen. Idealerweise erfolgt die Ermittlung der benötigten Daten direkt nachdem der Benutzer die Aufgabe an seinen Assistenten übergeben hat. Nachdem die benötigten Daten ermittelt wurden, gibt es zwei Möglichkeiten, diese für den Assistenten zugreifbar zu machen.

Entweder man übergibt dem Assistenten direkt die entschlüsselten Benutzerdaten, oder man ermöglicht stattdessen dem Assistenten die Entschlüsselung der benötigten Benutzerdaten zum Zeitpunkt der Ausführung einer Aufgabe. Die Übergabe der entschlüsselten Daten an den Assistenten hat den Nachteil, dass unverschlüsselte Kopien der eigentlichen Daten entstehen. Werden die ursprünglichen, verschlüsselten Daten in der Datenbank geändert, müssen alle Kopien aktualisiert werden. Darüber hinaus kostet die mehrfache Abspeicherung von identischen Daten unnötig Speicherplatz.

Bei der zweiten Möglichkeit, bleiben die Benutzerdaten verschlüsselt abgespeichert in der Datenbank, bis die Aufgabe ausgeführt wird. Zwischenzeitliche Änderungen an den Daten sind somit kein Problem. Deshalb wird diese Variante verwendet, um dem Assistenten den Offlinezugriff zu ermöglichen. Der Benutzer erstellt dafür über

die *Mobile* Java-Karte eine Zugriffsberechtigung, die als Ticket bezeichnet wird. Ein solches Ticket enthält ein oder mehrere Datenschlüssel, die mit dem öffentlichen Schlüssel der Datenbank verschlüsselt sind. Auf welche Daten mit Hilfe eines Tickets zugegriffen werden kann, wird durch die darin enthaltenen Datenschlüssel bestimmt. Das setzt voraus, dass jeder Datenschlüssel nur für die Verschlüsselung von einem Eintrag in der Datenbank verwendet wurde. Der für die Entschlüsselung eines Tickets notwendige private Schlüssel ist nur der Datenbank-Zugriffsschicht, einem speziellen Modul auf dem *Mobile* Server bekannt. Neben den Datenschlüsseln enthält das Ticket noch Informationen, mit denen die Verwendung eingeschränkt werden kann. Es lässt sich z. B. ein Zeitpunkt festlegen, nach dem das Ticket ungültig wird. Die Überprüfung geschieht durch die Datenbank-Zugriffsschicht, die ungültige Tickets abweist. So lange ein Ticket gültig ist, kann es beliebig oft verwendet werden. Ursprünglich war angedacht, jedes Ticket zusätzlich mit einer maximalen Verwendungsanzahl zu versehen. Dies setzt jedoch voraus, dass sich jedes Ticket eindeutig identifizieren lässt. Zusätzlich muss die Datenbank-Zugriffsschicht für jedes Ticket die Anzahl der Verwendungen protokollieren. Diesem hohen Aufwand steht nur eine geringe Verbesserung der Sicherheit gegenüber, weshalb auf die Möglichkeit, die Verwendungsanzahl einschränken zu können, verzichtet wurde.

3.3.3.1 Verschlüsselungs- und Entschlüsselungs-Tickets

Neben den Benutzerdaten werden auch die von den Webservices zurückgegebenen Informationen durch den Assistenten in der Datenbank abgespeichert. Auf diese Ergebnisdaten benötigt der Assistent schreibenden Zugriff, auf die Benutzerdaten hingegen nur einen lesenden Zugriff. Aus diesem Grund lassen sich Tickets erzeugen, die entweder nur das Lesen oder nur das Schreiben von Daten aus der Datenbank erlauben. Wie beim Ablaufzeitpunkt übernimmt die Datenbank-Zugriffsschicht die Prüfung und Durchsetzung dieser Beschränkungen. Die Notwendigkeit zur Einführung von Tickets, die nur zum Schreiben und damit zum Verschlüsseln von Daten verwendet werden können, ergibt sich aus der symmetrischen Signatur der öffentlichen Karten-Schlüssel. Bei der normalen hybriden Verschlüsselung wird ein zufälliger Sitzungs-Schlüssel (entspricht dem Datenschlüssel) erst dann erzeugt, wenn Daten verschlüsselt werden sollen. Dieser Schlüssel wird mit ein oder mehreren vertrauenswürdigen öffentlichen Schlüsseln verschlüsselt. Übertragen auf das *Mobile*-System, müsste die Datenbank-Zugriffsschicht beim Abspeichern von neuen Daten einen Datenschlüssel erzeugen und diesen, mit allen vertrauenswürdigen öffentlichen Kartenschlüsseln des Benutzers verschlüsselt, in der Datenbank abspeichern. Das Problem liegt in der Prüfung der Vertrauenswürdigkeit der öffentlichen Kartenschlüssel, denn dafür wird der Passwortsatz benötigt. Dieser ist jedoch nur dem Benutzer und seinen *Mobile* Java-Karten bekannt. Über die Verschlüsselungstickets bereitet deshalb

die *Mobile* Java-Karte ein oder mehrere Datenschlüssel vor, die dann durch die Datenbank-Zugriffsschicht bei Bedarf zum Verschlüsseln der Ergebnisdaten genutzt werden können.

3.3.3.2 Erstellung eines Tickets

Der gesamte Ablauf von der Aufgabe, die der Benutzer stellt, bis zur Ausstellung eines Tickets, ist in Abbildung 3.4 dargestellt. Nachdem der Benutzer eine Aufgabe gestellt hat (1), wird diese vom *Mobile* Client zum Assistenten auf dem *Mobile* Server weitergereicht (2). Dieser ermittelt die zur Lösung der Aufgabe benötigten Benutzerdaten. Danach wird eine Anfrage für ein Ticket erstellt, über das der Assistent Zugriff auf die benötigten Benutzerdaten erhält (3). Die Anfrage wird dem Benutzer angezeigt um ihn zu informieren, welche Daten benötigt werden (4). Ist dieser mit der Freigabe der Daten einverstanden, autorisiert er die Ticketerstellung (5). Der *Mobile* Client erstellt daraufhin über die *Mobile* Java-Karte ein Ticket für die angefragten Daten (7) und leitet das ausgestellte Ticket an den Assistenten auf dem *Mobile* Server. Dieser speichert das Ticket zusammen mit den Informationen über die Aufgabe und löst es ein, wenn die Aufgabe ausgeführt wird.

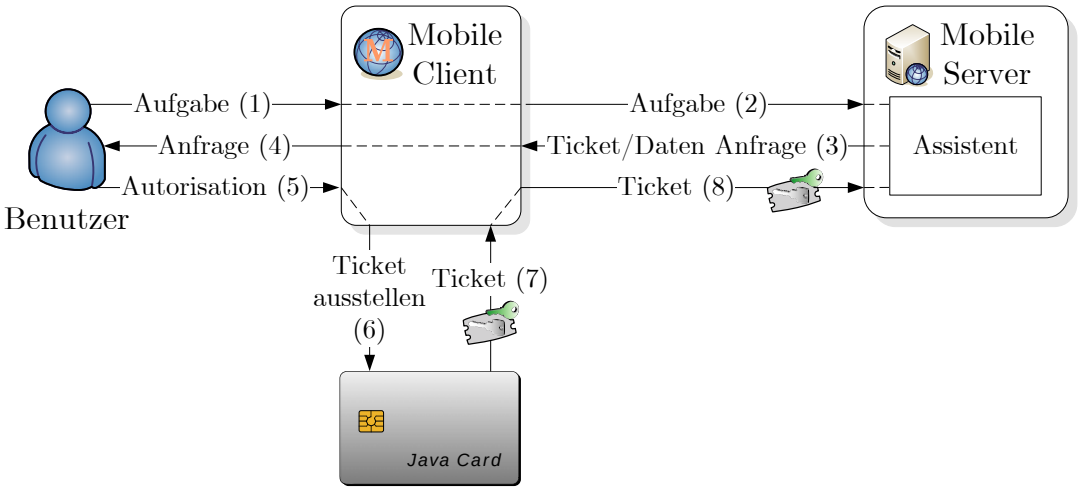

Abbildung 3.4: Schritte von der Aufgabenstellung zum Ticket

3.3.3.3 Einlösen eines Tickets

Benötigt der Assistent Zugriff auf Benutzerdaten um eine Aufgabe auszuführen, kann er über ein passendes Ticket eine unverschlüsselte Kopie der Daten von der Datenbank-Zugriffsschicht anfordern. Dies ist in Schritt (1) in Abbildung 3.5 dargestellt. Nach der Entschlüsselung des Tickets prüft die Datenbank-Zugriffsschicht, ob alle benötigten Datenschlüssel enthalten sind, und ob die Einschränkungen des Tickets erfüllt sind. Ist beides der Fall, werden die verschlüsselten Benutzerdaten aus der Datenbank ausgelesen (2) und jeweils mit dem passenden Datenschlüssel aus dem Ticket entschlüsselt. Nach der Übergabe der entschlüsselten Daten an den

Assistenten besteht kein Bedarf mehr für die entschlüsselten Benutzerdaten und die Datenschlüssel aus dem Ticket, sodass diese sofort gelöscht werden.

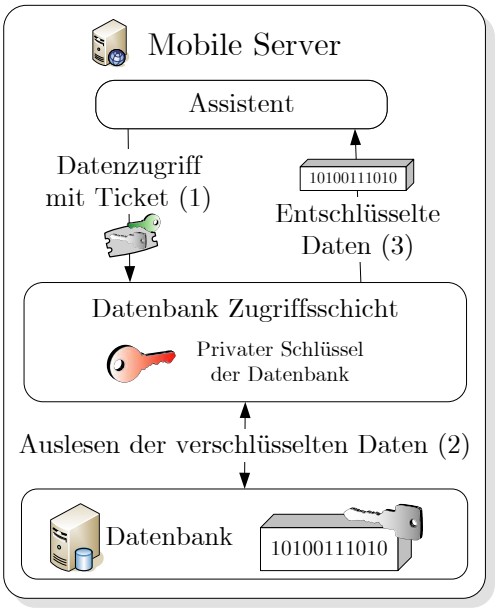

Abbildung 3.5: Zugriff auf Daten durch Einlösen eines Tickets

3.3.4 Karten Verwaltung

Im Gegensatz zu herkömmlichen Identitätsmanagement-Systemen, bei denen der Benutzer nur eine Chipkarte oder ein USB-Token zur Identifikation besitzt, erlaubt das *Mobile*-System jedem Benutzer mehrere Java-Karten zu verwenden. Die Funktionalität jeder einzelnen Karte ist aus Sicht des Benutzers identisch. Einzig die Karten-PIN, mit der sich der Benutzer gegenüber der Karte authentifiziert, kann bei jeder Karte anders gesetzt werden.

3.3.4.1 Hinzufügen einer neuen *Mobile* Java-Karte

Will der Benutzer dem System eine weitere *Mobile* Java-Karte hinzufügen, benötigt er nur seinen Benutzernamen, das dazu gehörige Passwort und den Passwortsatz. Benutzername und Passwort hat der Benutzer bei der Anmeldung für die Nutzung der *Mobile*-Plattform erhalten. Der Passwortsatz wird vom Benutzer beim Hinzufügen der ersten *Mobile* Java-Karte gewählt (siehe Abschnitt 3.3.2).

Den vereinfachten Ablauf, wenn eine neue *Mobile* Java-Karte dem System hinzugefügt wird, zeigt die Abbildung 3.6. Die erforderlichen Daten für eine neue Karte werden vom Benutzer in (1), zusammen mit der PIN für die neue Karte an den *Mobile* Client übermittelt. Dieser authentifiziert sich gegenüber dem *Mobile* Server und fordert eine Identifikations-Nummer für eine neue Karte an. Außerdem werden der öffentliche Schlüssel der Datenbank und die der anderen *Mobile* Java-Karten des

Benutzers ausgelesen (3). Bevor diese Daten an die Java-Karte weitergegeben werden, muss erst die *Mobile* Karten-Anwendung installiert werden. Diese wird mit den gesammelten Daten und der vom Benutzer gewählten Karten-PIN initialisiert. Vor der Installation der öffentlichen Schlüssel der anderen Karten prüft die *Mobile* Java-Karte mit Hilfe des Passwortsatzes die symmetrischen Signaturen der öffentlichen Kartenschlüssel. Nach erfolgreicher Prüfung und Installation aller Schlüssel ist die Initialisierung abgeschlossen und die Karten-Anwendung gibt den eigenen signierten, öffentlichen Schlüssel an den *Mobile* Client zurück. Dieser fügt ihn der Liste der öffentlichen Schlüssel in der Datenbank an und schließt damit die Initialisierung ab.

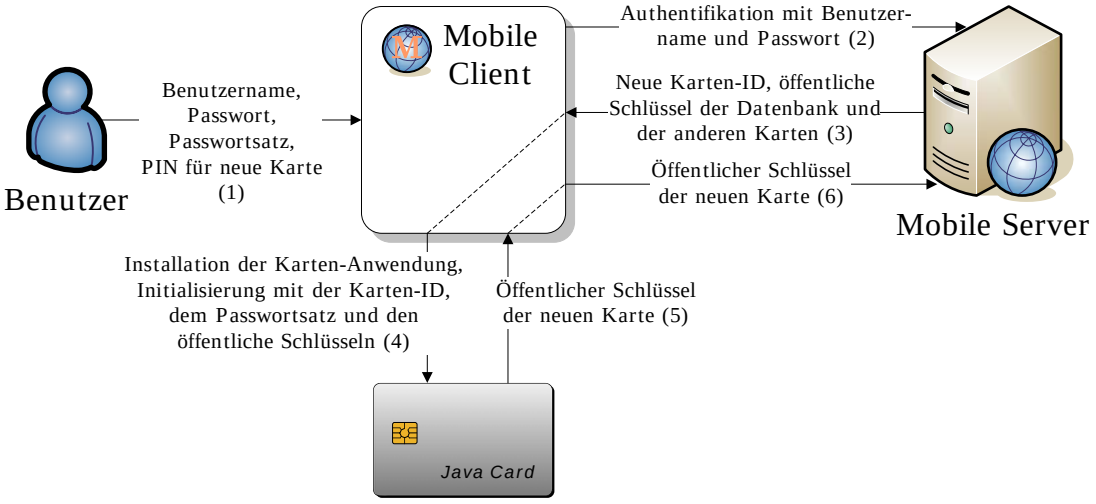

Abbildung 3.6: Ablauf wie dem System eine neue Java-Karte hinzugefügt wird

Nach der erfolgreichen Initialisierung einer neuen Karte muss noch die Liste der verschlüsselten Datenschlüssel in der Datenbank aktualisiert werden. Vorher kann die neue Karte nicht verwendet werden, da alle Datenschlüssel nur für die bisherigen Karten des Benutzers verschlüsselt sind. Deshalb ist es notwendig, dass sich der Benutzer nach dem Hinzufügen einer neuen Karte mit einer der anderen Karten anmeldet. Bei jeder Anmeldung wird die Datenbank auf neue und gesperrte Karten untersucht. Wurde zwischenzeitlich eine neue Karte hinzugefügt, erweitert die Karte ihre interne Liste der öffentlichen Schlüssel um den der neuen Karte. Dabei wird, wie bei jedem Import eines öffentlichen Schlüssels, die Signatur des Schlüssels überprüft. Bei einer fehlgeschlagenen Prüfung wird der Benutzer gewarnt, dass die Datenbank fehlerhafte Schlüssel enthält und eventuell eine Manipulation vorliegt. Nach dem erfolgreichen Import des öffentlichen Schlüssels der neuen Karte folgt die Aktualisierung aller in der Datenbank vorhandenen Datenschlüssel. Jeder einzelne wird dazu auf die Karte geladen und dort mit dem privaten Schlüssel der Karte entschlüsselt. Der Datenschlüssel wird mit dem öffentlichen Schlüssel der neuen Karte verschlüsselt und der Datenbank hinzugefügt. Abhängig von der Anzahl der Datenschlüssel in der Datenbank kann dieser Vorgang länger dauern. Nach jedem Hinzufügen ei-

ner neuen Karte ist diese Aktualisierung der verschlüsselten Datenschlüssel nur ein Mal notwendig, deshalb darf der Zeitbedarf dafür etwas höher ausfallen ohne die Benutzbarkeit des *Mobile*-Systems zu beeinträchtigen.

3.3.4.2 Sperren einer *Mobile* Java-Karte

Geht eine Karte verloren, wird gestohlen oder funktioniert nicht mehr, sollte sie sicherheitshalber gesperrt werden. Sofern es sich bei der Karte nicht um die letzte aktive Karte des Benutzers handelt, kann diese durch eine neuerstellte ersetzt werden. Hatte der Benutzer nur diese eine aktive Karte, sind damit auch alle Daten in der Datenbank des *Mobile* Servers nicht mehr zugreifbar, da es keine Möglichkeit mehr gibt diese zu entschlüsseln. Aus diesem Grund sollte jeder Benutzer immer mindestens über zwei aktive Karten verfügen.

Die sichere Umsetzung einer Kartensperrung ist erheblich aufwendiger als das Hinzufügen einer neuen Karte, da Manipulationen in der Datenbank wesentlich schwieriger zu erkennen sind. Man muss dabei gar nicht unbedingt von einer gezielten Manipulation der Datenbank ausgehen. Nehmen wir an, zum Sperren eine Karte wird nur deren öffentlicher Schlüssel in der Datenbank gelöscht und die Datenbank muss wegen eines Hardware- oder Software-Fehlers durch ein Backup ersetzt werden, das vor der Sperrung angelegt wurde, dann ist die vorher gesperrte Karte wieder aktiv, ohne dass der Benutzer etwas davon mitbekommt. Dieser geht immer noch davon aus, dass die Karte gesperrt ist. Gleiches gilt natürlich, wenn die Datenbank absichtlich manipuliert wurde. Die einzige Möglichkeit, einen solchen Angriff zu verhindern, ist auf jeder Karte eine Liste der gesperrten Karten zu führen. Diese Liste wird bei jeder Anmeldung aktualisiert und dabei um neue gesperrte Karten ergänzt. Wird dabei eine als gesperrt bekannte Karte gefunden, die in der Datenbank als nicht gesperrt markiert ist, wird der Benutzer gewarnt, dass eine Manipulation der Datenbank vorliegt. Eine solche Aktualisierung der karteninternen Daten kann nur stattfinden, wenn der Benutzer die Karte verwendet. Deshalb ist es unerlässlich, dass der Benutzer nach der Sperrung einer Karte alle verbleibenden Karten mindestens einmal verwendet um sich am *Mobile*-System anzumelden. Die dabei ausgeführte Aktualisierung sorgt dafür, dass alle Karten über die Sperrung informiert werden.

Da zwischen einer Manipulation der Datenbank und einem möglichen Entsperren einer Karte durch den Benutzers nicht unterschieden werden kann (z. B. weil eine Karte wiedergefunden wurde), ist die Sperrung eine Karte unwiderruflich. Ein Entsperren ist nicht möglich. Sperrt ein Benutzer eine vermeintlich verlorene Karte und findet sie später wieder, kann er sie löschen und durch eine neue Initialisierung wieder dem System hinzufügen.

3.4 Benutzer-Seite

Aus Sicht eines Benutzers besteht die *Mobile*-Plattform aus drei Teilen:

1. Der Software, die er auf seinem Gerät installieren muss
2. Den Chipkarten, als sichere Speicher für die beteiligten privaten Schlüssel
3. Dem *Mobile* Server, auf dem alle Daten verschlüsselt abgelegt werden und der die gestellten Aufgaben ausführt

Auf allen Geräten, mit denen auf den *Mobile* Server zugegriffen werden soll, muss eine spezielle Software installiert werden. Diese Software wird als *Mobile* Client bezeichnet. Sie ist nötig, um lokal am oder im Endgerät verfügbare Hardware wie GPS-Empfänger oder einen Chipkartenleser anzusprechen. Letzerer ist Voraussetzung um Chipkarten und damit auch die benötigte Java-Karte verwenden zu können.

3.4.1 *Mobile* Client

Der *Mobile* Client dient haupsächlich dazu, die Kommunikation zwischen der *Mobile* Java-Karte und dem *Mobile* Server herzustellen. Neben der Anbindung der Java-Karte bietet der *Mobile* Client die Möglichkeit Module einzubinden, über die andere lokale Hardware z. B. zur Positionsbestimmung in die *Mobile*-Plattform eingebunden werden kann. Durch diese Anforderungen ist es nicht möglich, den *Mobile* Client in Form eines Java-Applets umzusetzen. Ein Java-Applet wird in eine Webseite eingebettet und vom Webbrowser in einer „Sandbox“ ausgeführt. Diese verhindert aus Sicherheitsgründen den Zugriff auf alle lokalen Ressourcen, wie ein Chipkartenlesegerät[4]. Deshalb erfolgt die Umsetzung des *Mobile* Clients als eigenständiges Programm, das auf dem Endgerät des Benutzers installiert wird. Programme dieser Art werden nicht in einer „Sandbox“ ausgeführt, der Zugriff auf die Java-Karte ist so möglich. Für die Interaktion mit dem Benutzer stellt der *Mobile* Client eine webbasierte Benutzungsoberfläche zur Verfügung.

Durch die große Vielfalt an möglichen Endgeräten, ist es nicht möglich, einen *Mobile* Client zu entwickeln, der auf allen Endgeräten funktionsfähig ist. Vergleicht man die verfügbare Hardware, wie die Bildschirmgöße, den verfügbaren Speicher oder die Geschwindigkeit des Prozessors der verschiedenen Endgeräte, lassen sich diese in drei Klassen einteilen:

1. Personal Computer, stationär (Desktop) oder mobil (Notebook), mit dem Betriebssystem Windows, Linux oder einem anderen verwandten Betriebssystem
2. Handheld-Computer, Persönliche digitale Assistenten (PDA) mit der Möglichkeit eine Internetverbindung aufzubauen

[4]What Applets Can and Can't Do: `http://java.sun.com/docs/books/tutorial/applet/overview/security.html`

3. Java-fähige Mobiltelefone mit einer Java-Karten-SIM/USIM oder einem zusätzlichen integrierten Chipkartenlesegerät

Um innerhalb der Geräteklassen nicht auch noch auf die unterschiedlichen Betriebssysteme eingehen zu müssen, bietet sich die Programmiersprache Java an. Diese ermöglicht es den *Mobile* Client unabhängig vom eingesetzten Betriebssystem zu entwickeln. Zudem existiert eine passende Java-Plattform für jede der drei Geräteklassen, die es ermöglicht die Funktionen des Gerätes optimal zu nutzen. Die Wahl der Programmiersprache und die Betrachtung der verschiedenen Geräteklassen ist eigentlich eine Teilaufgabe der Implementierung. Die frühzeitige Analyse dieser Aspekte schon beim Design ermöglicht es jedoch, Rücksicht auf die zur Verfügung stehende Funktionalität zu nehmen. Dies verhindert den Entwurf eines Systems, das sich bei seiner späteren Implementierung als nicht oder nur eingeschränkt umsetzbar herausstellt.

Bei der Erstellung der webbasierten Benutzungsoberfläche arbeiten der *Mobile* Client und der *Mobile* Server zusammen. Um Geräte mit geringen Ressourcen, wie Mobiltelefone, nicht zu überfordern, werden die Webseiten der Benutzungsoberfläche wenn möglich vom *Mobile* Server erzeugt. Der Client ergänzt diese durch lokal erzeugte Webseiten, die den Zugriff auf die Java-Karte und die sichere Eingabe von Benutzerdaten ermöglichen. Um die serverseitig erzeugten Webseiten besser an die Fähigkeiten des verwendeten Endgerätes anpassen zu können, ermittelt der *Mobile* Client einige der Geräteeigenschaften wie die Bildschirmauflösung und sendet sie zum *Mobile* Server.

Für den Zugriff auf diese webbasierte Benutzungsoberfläche gibt es zwei unterschiedliche Möglichkeiten der Umsetzung:

1. Ein speziell angepasster Webbrowser, in den der *Mobile* Client integriert ist. Die vom *Mobile* Server erzeugten Webseiten werden direkt dargestellt, die clientseitigen Webseiten hingegen durch Erweiterungen im Browser erzeugt.
2. Ein unabhängiger Dienst, der als lokaler Webserver arbeitet. Auf diesen kann der Benutzer mit dem gewohnten Webbrowser zugreifen. Der lokale Webserver generiert einen Teil der Benutzungsoberfläche, ein anderer Teil wird über einen integrierten Proxy vom *Mobile* Server bezogen.

Im Folgenden wird überprüft, für welche Geräteklasse sich diese Möglichkeiten umsetzten lassen und welche Vor- und Nachteile dadurch entstehen.

Auf normalen Desktop oder Notebook-Computern sind beide Varianten realisierbar. Die heutigen Betriebssysteme, die auf diesen Geräten laufen, enthalten bereits eine Vielzahl von Hintergrundprogrammen. Darüber hinaus installieren immer mehr Anwendungen zusätzliche Dienste und Hintergrundprogramme, die immer aktiv sind,

aber nur selten genutzt werden. Diese verlangsamen das System und verschlechtern so die Benutzbarkeit. Aus diesem Grund bevorzugen viele Benutzer Programme, die man nur bei Bedarf startet und später wieder beendet, wenn man sie nicht mehr benötigt. Damit würde sich die Variante mit dem angepassten Webbrowser anbieten. Zudem existieren bereits einige Java-Webbrowser, in die man den *Mobile* Client integrieren könnte (Beispiele: JBrowser[5] oder Jazilla[6]). Eine Integration des Clients in einen Browser hätte zudem den Vorteil, dass die Kommunikation zwischen Browser und Client nur innerhalb eines Prozesses erfolgt und damit keine Möglichkeiten für einen Angriff bietet. Andererseits lehnen es viele Benutzer ab, einen anderen Webbrowser, als den gewohnten und favorisierten zu verwenden und sei es nur temporär. Dies würde zu einer geringeren Akzeptanz der *Mobile*-Plattform durch die Benutzer führen. Deshalb sollte der *Mobile* Client als eigenständiger Dienst für diese Plattform umgesetzt werden.

Auch auf aktuellen Handheld-Computern ist es möglich, Programme im Hintergrund auszuführen. Da die Resourcen, wie der verfügbare Hauptspeicher und die Geschwindigkeit des Prozessors, in dieser Geräteklasse sehr begrenzt sind, sollte man jedoch sehr sparsam damit umgehen, was gegen die Realisierung als Dienst spricht. Die Variante mit dem modifizierten Browser ist deshalb für diese Plattform zu bevorzugen.

Ganz anders sieht es auf Java-fähigen Mobiltelefonen aus. Dort ist es nicht möglich, Programme als Dienst im Hintergrund zu starten, um dann mit dem Webbrowser darauf zuzugreifen. Die Variante mit dem unabhängigen Dienst ist somit für Mobiltelefone nicht realisierbar. Damit bleibt nur die Möglichkeit den *Mobile* Client in einen Webbrowser zu integrieren. Webbrowser für die Plattform Java 2 Micro Edition (J2ME) werden, wegen ihrer geringen Anforderungen an die Hardware, oft als „Microbrowser“ oder „Picobrowser“ bezeichnet. Zudem beschränken sich diese Browser auf die Darstellung nur eine der Webseiten-Beschreibungssprachen Wireless Markup Language (WML), Compact Hypertext Markup Language (CHTML) oder Extensible Hypertext Markup Language (XHTML) . Leider lassen sich diese Browser nicht um eine eigene Funktionalität erweitern, sodass es eventuell nötig ist, zusätzlich zum *Mobile* Client noch einen passenden Browser zu entwickeln.

Damit zeigt sich, dass beide Client-Varianten benötigt werden, um alle drei Geräteklassen unterstützen zu können. Auf den Personal-Computern sollte wegen der Benutzerakzeptanz die Lösung des unabhängigen Dienstes mit integriertem Proxy gewählt werden. Für die Mobiltelefone muss hingegen die Variante des Webbrowsers mit integriertem *Mobile* Client gewählt werden, da sich nur diese Variante in dieser Geräteklasse realisieren lässt. Die Handheld-Computer können prinzipiell mit beiden

[5] `http://www.bysoft.se/sureshot/jbrowser`
[6] `http://jazilla.sourceforge.net`

Varianten umgehen. Es bietet sich an, eine der Umsetzungen zu verwenden, die bereits für eine der anderen Geräteplattformen entworfen wurde. Dadurch lässt sich die spätere Implementierung auf nur zwei Lösungen reduzieren. Wegen der begrenzen Ressourcen auf den Handheld-Computern dürfte es leichter sein, den Browser-Client anzupassen. Dieser ist für den Einsatz auf Mobiltelefonen zugeschnitten, auf denen die Ressourcen noch knapper sind. Die Alternative wäre die Proxy-Lösung von der Geräteklasse der Personal-Computer auf die der Handheld-Computer zu portieren. Wegen der großen Unterschiede zwischen den beiden Geräteklassen ist diese Portierung jedoch sehr aufwendig.

3.4.2 *Mobile* Java-Karte

Die *Mobile* Java-Karte dient dem Benutzer als sicherer Speicher für die verschieden Daten, die für das Verschlüsselungssystem benötigt werden. Zudem verhindert die Karte die Manipulation der gespeicherten Daten oder des Programmcodes der Karten-Anwendung. Eine Auflistung aller Daten, die auf der Karte gespeichert werden, zeigt Abbildung 3.7. Benutzer- und Karten-Identifikationsnummer werden von der Datenbank vorgegeben. Damit wird verhindert, dass Identifikationsnummern mehrfach verwendet werden und so Konflikte zwischen Benutzern oder Karten entstehen. Das RSA-Schlüsselpaar der Karte wird direkt auf der Karte generiert. Der dabei erzeugte private Schlüssel verlässt nie die Karte. Des Weiteren befinden sich auf der Karte alle aus der Datenbank geladenen und von der Karte überprüften öffentlichen Schlüssel der anderen aktiven *Mobile* Java-Karten des Benutzers.

Abbildung 3.7: Daten auf der *Mobile* Java Card

3.5 *Mobile* Server

Der *Mobile* Server besteht aus ein oder mehreren Servern, die sich im Rechenzentrum eines Dienstanbieters oder einer Firma befinden. Damit ist der *Mobile* Server

der einzige Teil der *Mobile*-Plattform, der sich nicht unter direkter Kontrolle des Benutzers befindet. Zudem verarbeitet dieser die Daten aller Nutzer des Systems, was besondere Anforderungen an den Schutz dieser Daten stellt. Eine der Aufgaben des Servers ist die persistente Speicherung aller anfallenden Daten. Es bietet sich an, dafür eine Datenbank zu verwenden. Gespeichert werden Verwaltungsdaten, bestehend aus dem Benutzernamen und den dazugehörigen Passwörtern, die öffentlichen Kartenschlüssel aller Benutzer und die verschlüsselten Benutzerdaten. Außerdem werden die für die Ausführung der verschiedenen Aufgaben benötigten Daten sowie die Ergebnisse der Aufgaben gespeichert. Abgesehen von den Verwaltungsdaten und den öffentlichen Schlüsseln sind all diese Daten mit dem Verfahren verschlüsselt, das in Abschnitt 3.3 vorgestellt wurde.

3.5.1 Schnittstellen zum *Mobile* Client

Der Datenaustausch zwischen dem *Mobile* Client und dem Server erfolgt über das Internet, mit dem der Server dauerhaft verbunden ist. Der Client verwendet dabei zwei Schnittstellen, die vom *Mobile* Server zur Verfügung gestellt werden. Für die Nutzung dieser Schnittstellen ist eine Authentifikation erforderlich, weshalb nur dem System bekannte Benutzer die Dienste dieser Schnittstellen nutzen können. Für Zugriffe des *Mobile* Clients auf die Datenbank und den Assistenten ist die erste Schnittstelle als Webservice ausgeführt. Über die Webservice-Schnittstelle kann der *Mobile* Client Funktionen auf dem Server aufzurufen und so z. B. verschlüsselte Daten in der Datenbank ablegen. Ein Webservice bietet gegenüber anderen Varianten, wie dem Zugriff über RMI, mehrere Vorteile. So ist die Nutzung von Webservices sowohl unabhängig von der Plattform, als auch von der verwendeten Programmiersprache. RMI hingegen ist auf Java beschränkt. Damit ist es möglich den *Mobile* Client später in einer anderen Programmiersprache als Java umzusetzen. Gleiches gilt für den Server. Ein weiterer Vorteil ist die immer größere Verbreitung der Webservices, was zur Folge hat, das fertige Webservice-Bibliotheken für sehr viele Plattformen verfügbar sind. Selbst für Mobiltelefone mit J2ME existiert eine optionale Erweiterung, die es erlaubt Webservices zu nutzen[7].

Die Erstellung und Abfrage, der vom Benutzer an das *Mobile* System gestellten Aufgaben, ermöglicht die zweite Schnittstelle. Diese bietet den Zugriff auf eine webbasierte Benutzungsoberfläche, für die der *Mobile* Server dynamische Webseiten generiert. Je nach den Fähigkeiten des Endgerätes werden die Webseiten in einer der Beschreibungssprachen HTML, XHTML, WML oder CHTML ausgeliefert. Die Ermittlung der Gerätefähigkeiten und deren Übermittlung an den *Mobile* Server übernimmt der *Mobile* Client.

[7] JSR172: `http://www.jcp.org/en/jsr/detail?id=172`

3.5.2 Assistent

Der Assistent agiert als Stellvertreter des Benutzers auf dem Server und verwaltet dabei die Aufgaben und Tickets. Um eine gestellte Aufgabe auszuführen, sendet er Anfragen an Webservices außerhalb der *Mobile* Plattform. Stehen mehrere Webservice-Anbieter zur Verfügung, die einen vergleichbaren Dienst erbringen, kann der Assistent auf Grund von Kriterien, wie dem Preis oder den Vorlieben des Benutzers, eigenständig eine Auswahl treffen. Der Assistent ist die zentrale Komponente der *Mobile*-Plattform, die dafür verantwortlich ist, welche Funktionen und Aufgaben über die Plattform genutzt werden können. Dies wird durch die Anzahl und Diversität der angebundenen Webservices bestimmt. Außerdem ist von Bedeutung, wie flexibel der Assistent die verschiedenen Webservices bei der Ausführung der Aufgaben verknüpfen kann.

Das vorgestellte Design des Assistenten stammt vollständig aus dem MOBILE-Projekt (siehe Abschnitt 2.3). Die vorliegende Arbeit hat nicht zum Ziel das Design des Assistenten zu erweitern oder zu implementieren. Auf Grund der zentralen Bedeutung für die *Mobile*-Plattform konnte aber nicht auf eine Vorstellung verzichtet werden. Außerdem ist die Arbeitsweise des Assistenten der Grund, für die Einführung des Ticket-Systems und damit mitverantwortlich für das Design des Verschlüsselungssystems.

3.5.3 Datenbank-Zugriffsschicht

Der *Mobile* Server befindet sich aus der Sicht des Benutzers unter fremder Kontrolle. Was innerhalb des Systems mit seinen Daten passiert, ist für den Benutzer nicht kontrollierbar. Die Verschlüsselung der Benutzerdaten durch das vorgestellte Verschlüsselungssystem sorgt dafür, dass der Benutzer die Kontrolle auf den Zugriff seiner Daten behält, obwohl sich diese auf dem *Mobile* Server befinden. Das Ticket-System ermöglicht es zudem, den Zugriff auf Teile der Benutzerdaten zu delegieren. Zum Einlösen der Tickets wird eine Instanz auf dem *Mobile* Server benötigt, die stellvertretend für den Benutzer die Nutzung der Tickets kontrolliert. Diese Aufgabe übernimmt die Datenbank-Zugriffsschicht. Damit sie aus Sicht der Benutzer vertrauenswürdig ist, muss gewährleistet sein, dass sie die in den Tickets angegebenen Einschränkungen immer durchsetzt. Damit diese Überprüfung nicht umgangen werden kann, ist die Datenbank-Zugriffsschicht als einziger Teil der *Mobile*-Plattform in der Lage, die von den Benutzern ausgestellten Tickets einzulösen. Dies wird über den privaten Schlüssel erreicht, mit dessen Hilfe die Tickets entschlüsselt werden können. Dieser ist nur der Datenbank-Zugriffsschicht bekannt. Ohne den passenden Schlüssel gibt es keine Möglichkeit, Zugriff auf die in den Tickets enthaltenen Datenschlüssel zu erhalten, und ohne diese lassen sich die Benutzerdaten nicht entschlüsseln. Da die

Datenbank-Zugriffsschicht von allen Benutzern der *Mobile*-Plattform genutzt wird, muss sichergestellt werden, dass es nicht zur Vermischung von Daten verschiedener Benutzer kommen kann. Jede Einlösung eines Tickets durch den Assistenten muss getrennt von anderen nachfolgenden oder gleichzeitigen Zugriffen erfolgen.

3.5.4 Datenbank

Die Datenbank dient der dauerhaften Speicherung von Verwaltungs- und Benutzerdaten. Zu den Verwaltungsdaten gehören die Benutzernamen und Passwörter aller Benutzer der *Mobile*-Plattform. Ebenso dient die Datenbank als Speicherort für die öffentlichen Kartenschlüssel aller Benutzer und deren verschlüsselte Benutzerdaten. Durch die Verschlüsselung der Benutzerdaten sind sie in der Datenbank gegen unberechtigtes Auslesen gut geschützt. Gleiches gilt für die öffentlichen Kartenschlüssel, die durch ihre Signatur vor Manipulation geschützt sind.

4. Implementierung

Nach dem Entwurf des Designs des *Mobile*-Projektes im letzten Kapitel folgt nun die Implementierung. Durch die Beachtung einiger Einschränkungen der Implementierung schon während der Entwicklung des Designs, ist die Umsetzbarkeit des Designs eigentlich kein großes Problem mehr. In der Praxis jedoch tauchen schnell Probleme auf, beispielsweise, wenn es darum geht, ein Mobiltelefon zu finden, das alle benötigten Funktionen unterstützt. Insbesonders die Kommunikation mit der Java-Karte auf einem Handheld-Computer oder Mobiltelefon stellt sich aktuell (2005) als noch nicht durchführbar heraus. Deshalb ist es nur möglich den *Mobile* Client für die Geräteklasse der Personal Computer zu implementieren. Ohne die Geräteklasse der Mobiltelefone und der Handheld-Computer ist die Bereitstellung der webbasierten Benutzungsoberfläche für mehrere Geräteklassen in dieser Implementierung nicht mehr notwendig.

Als besondere Hindernisse bei der Implementierung haben sich die verwendeten Software-Komponenten, wie die Webserver und die Entwicklung der J2EE-Anwendungsmodule, ergeben. Die Integration von bereits fertigen Software-Komponenten und der Einsatz von Frameworks erspart viel Arbeit. Die Komplexität dieser Komponenten und Frameworks macht jedoch eine gründliche und zeitaufwendige Einarbeitung notwendig, was die Arbeitsersparnis stark verringert.

Die Implementierung des *Mobile*-Projektes erfolgt in drei Schritten. Im ersten Schritt wird die Java-Karten-Anwendung für die *Mobile* Java-Karte implementiert. Danach folgt im zweiten Schritt die Implementierung des *Mobile* Clients und im letzten Schritt werden die J2EE-Komponenten für den *Mobile* Server implementiert. Der Grund, die Implementierung für die drei Plattformen Java-Karte, mobiles Endgerät und J2EE-Anwendungsserver in dieser Reihenfolge durchzuführen, liegt in den Beschränkungen der jeweiligen Plattformen. Die Java-Karte besitzt die stärksten

Einschränkungen unter den drei Plattformen. Dadurch beeinflusst sie auch die Implementierung der beiden anderen Teile. Deshalb wird die Java-Karten-Anwendung als erstes implementiert. Wegen der direkten Kommunikation zwischen Java-Karte und *Mobile* Client ist es empfehlenswert im nächsten Schritt die Client-Software umzusetzen. Als letzter Schritt bleiben dann noch die J2EE-Komponenten für den *Mobile* Server übrig.

Die vollständigen Quelltexte, die für die Implementierung der *Mobile*-Plattform in dieser Arbeit erstellt wurden, finden sich auf der CD-ROM im Anhang.

4.1 *Mobile* Java-Karte

Die Anforderungen, die das *Mobile*-Projekt an die Java-Karte stellt, sind sehr hoch. Da der Benutzer zu jeder Zeit Karten sperren oder dem System neue Karten hinzufügen kann, ist der Speicherverbrauch für die Liste der öffentlichen Schlüssel der anderen Karten nicht vorhersagbar. Wird eine Karte gesperrt, wird der dazugehörige öffentliche Schlüssel auf allen anderen Karten gelöscht. Der Speicher von gelöschten Objekten steht den Karten-Anwendungen jedoch nicht wieder als freier Speicher zur Verfügung. Erst Java-Karten nach der neuen Java-Karten-Spezifikation 2.2.1 können den Speicher von gelöschten Objekten wieder freigeben. Zusätzlich können Java-Karten in dieser neuen Version über JC-RMI mit der Clientanwendung auf dem Computer kommunizieren (siehe Abschnitt 2.2.7.2). Beide Funktionen, die Freigabe von gelöschten Objekten und der Zugriff über JC-RMI, ermöglichen eine sehr flexible Implementierung, die zudem wesentlich einfacher um neue Funktionen erweitert werden kann. Zusätzlich vereinfacht es die Implementierung sowohl des Programmteiles auf der Java-Karte als auch die des *Mobile* Clients. Diese positiven Eigenschaften führten recht schnell zu der Entscheidung, die Karten-Anwendung für die Java Card 2.2.1-Plattform zu entwickeln.

Auch wenn sich auf einer Java-Karte mehrere unterschiedliche Karten-Anwendungen installieren lassen, sollte die *Mobile* Karten-Anwendung möglichst alleine auf einer Java-Karte installiert werden. Ansonsten könnte es vorkommen, dass der freie Speicher beim Hinzufügen einer neuen *Mobile* Java-Karte nicht mehr ausreicht. In einem solchen Fall wäre die Karte dann nicht mehr in der Lage, alle notwendigen öffentlichen Schlüssel als Kopie auf der Karte bereitzustellen, und würde nicht mehr korrekt funktionieren.

4.1.1 Entwicklung der Karten-Anwendung

Zum Entwickeln und Testen von Java-Karten-Anwendungen bietet Sun Microsystems das „Java Card 2.2.1 Development Kit“[1] an. Dieses enthält auch einen Emu-

[1] `http://java.sun.com/products/javacard/dev_kit.html`

lator, mit dem sich Karten-Anwendungen für Java-Karten testen lassen. Ein Emulator ist oftmals die einzige Möglichkeit bestimmte Fehler zu finden, da im Falle eines Fehlers von der Karten-Anwendung nur eine Fehlerkonstante zurückgegeben wird (siehe auch „Status-Wort“ in Abschnitt 2.2.6). Leider fehlen dem Java-Karten-Emulator von Sun einige wichtige Implementierungen kryptographischer Verfahren, z. B. Triple-DES und ein kryptographisch sicherer Zufallsgenerator (secure random). Deswegen verwendet man besser gleich einen alternativen Java-Karten-Emulator von einem der Chipkartenhersteller wie Giesecke & Devrient, Gemplus oder IBM. Diese Emulatoren sind meist Teil einer Entwicklungsumgebung, die sich die Hersteller teuer bezahlen lassen (zur Zeit 1000 Euro und mehr pro Lizenz). Einzig von IBM gibt es ein Plugin[2] für die bekannte Entwicklungsumgebung Eclipse[3], die mit umgerechnet zur Zeit 50 Euro sehr günstig ist. Mit inbegriffen ist eine Java-Karte vom Typ JCOP41v2.2, die erste öffentlich verfügbare Java-Karte nach der Spezifikation 2.2.1. Karten nach dieser Spezifikation sind aktuell (Ende 2005) immer noch recht selten. Alle Chipkartenhersteller haben aber entsprechende Produkte angekündigt. Obwohl die JCOP41v2.2 Java-Karte erst Mitte 2005 auf den Markt gekommenen ist, unterstützt sie nicht den Rijndael-Algorithmus, einen symmetrischen Verschlüsselungsalgorithmus, der auch als AES bekannt ist. Dieser Algorithmus lässt sich sehr effizient sowohl in Hardware als auch in Software implementieren, der alternative Algorithmus 3DES hingegen nur in Hardware. Außerdem bietet der AES-Algorithmus bei gleicher Schlüssellänge eine höhere Sicherheit als 3DES. Mangels alternativer Java-Karten nach der Spezifikation 2.2.1, die den AES-Algorithmus unterstützen, ist die vorliegende Implementierung für die Java-Karte JCOP41v2.2 entwickelt worden. Für die symmetrische Verschlüsselung wird der 3DES-Algorithmus verwendet. Bei der Implementierung wurde aber darauf geachtet, dass sich dieser sehr einfach durch AES oder einen anderen kompatiblen symmetrischen Verschlüsselungsalgorithmus ersetzen lässt.

4.1.2 Interne Klassenstruktur

Die Menge der Klassen und Interfaces, die auf der Java-Karte die Karten-Anwendung bilden, sind in Abbildung 4.1 dargestellt. Von zentraler Bedeutung ist dabei das Interface *IMobile*. Nur Methoden dieses Interfaces sind von außerhalb der Java-Karte über JC-RMI ausführbar. Die eigentliche Funktionalität ist in der Klasse *IMobileImpl* enthalten. Die drei anderen Interfaces, die auch außerhalb der Java-Karte verwendet werden, definieren Konstanten, wie beispielsweise die maximale Länge der Karten-PIN oder die Nummer für einen bestimmten, auf der Karte aufgetretenen, Fehler.

[2] JCOP Tools: `http://www.zurich.ibm.com/jcop/products/tools.html`
[3] `http://www.eclipse.org`

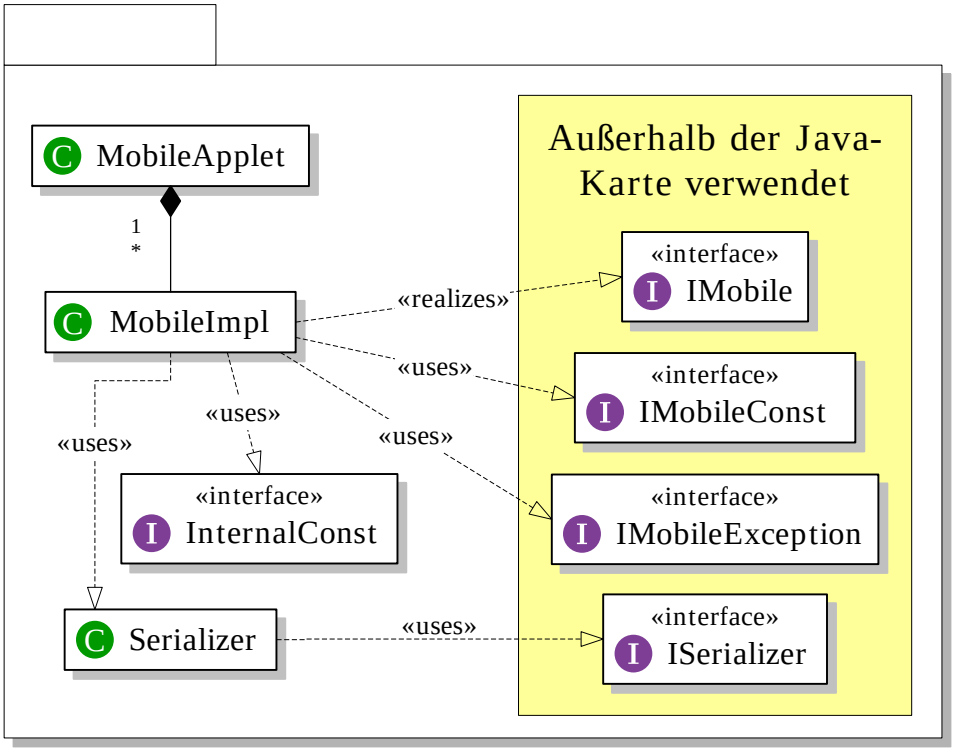

Abbildung 4.1: UML Klassendiagramm der Java-Karte (*mobile.jc.oncard*)

Um öffentliche Schlüssel zwischen Java-Karte und *Mobile* Client zu transferieren, müssen diese in ein übertragbares Format gebracht werden. Im Kontext der Remote Method Invocation spricht man von der Serialisierung eines Objektes. Die Deserialisierung rekonstruiert das Objekt wieder aus dem übertragbaren Format. Auf der Java-Karte übernimmt diese Aufgabe die Klasse *Serializer*. Bei der Serialisierung und Deserialisierung werden Konstanten verwendet, die angeben, wie lang oder von welchem Typ der entsprechende Schlüssel ist. Diese Konstanten sind im Interface *ISerializer* zu finden.

4.1.3 Bearbeitung von Datenblöcken

Auf Grund der geringen maximalen Länge einer APDU (siehe Abschnitt 2.2.6) kommt es oft vor, dass Daten nicht in einem Stück auf die Java-Karte übertragen werden können. In diesen Fällen kann versucht werden, die Bearbeitung blockweise durchzuführen. Dabei werden die zu verarbeitenden Daten in mehrere Blöcke aufgespalten, die klein genug sind, um in einem Stück auf die *Mobile* Java-Karte übertragen zu werden. Auf der Karte werden die Teildatenblöcke sofort bearbeitet und das Ergebnis als Antwort zurückgegeben. Dieses Verfahren hat den Vorteil, dass auf der Java-Karte kein Speicher benötigt wird, um die Daten oder das Ergebnis zwischenzuspeichern, es wird z. B. beim Verschlüsseln und Entschlüsseln von Daten über die *Mobile* Java-Karte verwendet. In vielen Fällen ist eine solche Vorgehensweise jedoch nicht möglich, beispielsweise, wenn die Daten nicht linear bearbeitet werden kön-

nen. Dann bleibt nur die Alternative, die zu bearbeitenden Daten komplett in den Speicher der Java-Karte zu übertragen und sie erst dann zu bearbeiten. Der freie Speicher auf der Java-Karte bestimmt in diesem Fall die maximale Länge der bearbeitbaren Daten. Die Übertragung der Daten auf die Karte und das Auslesen nach der Bearbeitung muss wiederum in mehreren Schritten durch die Übertragung von kleinen Blöcken geschehen. Es werden jedoch im Gegensatz zur direkten blockweisen Bearbeitung doppelt so viele Schritte benötigt. Zur Anwendung kommt dieses Verfahren beim Import des öffentlichen Schlüssels der Datenbank oder einer anderen *Mobile* Java-Karte.

4.1.4 Datenformate

Der Datenaustausch zwischen den verschiedenen *Mobile* Java-Karten eines Benutzers untereinander und mit dem Assistenten auf dem *Mobile* Server erfolgt über binäre Datenblöcke. Diese enthalten mehrere Informationen, die zusammengehören, beispielsweise beim öffentlichen Kartenschlüssel unter anderem den öffentlichen Schlüssel und die HMAC-Signatur. Die Definition, welcher Teil eines binären Datenblockes welche Informationen enthält, nennt man Datenformat. Innerhalb des *Mobile*-Projektes kommen drei unterschiedliche Datenformate zum Einsatz, die in den folgenden Abschnitten genauer betrachtet werden. Dabei wird auch analysiert, wie der Schutz der Daten in den verschiedenen Datenformaten bezüglich deren Integrität und Vertraulichkeit umgesetzt wurde. Zusätzlich zu diesen drei Datenformaten existiert noch ein weiteres Datenformat, das bei der Serialisierung von öffentlichen RSA-Schlüsseln verwendet wird. Dieses enthält jedoch keinerlei Schutzmechanismen und ist weder für das Verständnis dieser Arbeit, noch für die spätere Analyse und Bewertung relevant. Deshalb wird auf eine Vorstellung dieses Datenformates verzichtet.

4.1.4.1 Der signierte öffentliche Kartenschlüssel

Ein signierter öffentlicher Schlüssel besteht aus vier Teilen: Der Benutzer-Identifikationsnummer, der Karten-Identifikationsnummer, den serialisierten Schlüsseldaten des öffentlichen Schlüssels und der HMAC-Signatur. Die Länge des gesamten Gebildes ist abhängig von der gewählten Schlüssellänge des RSA-Schlüssels. Kommt ein RSA-Schlüssel mit 1024 Bit zum Einsatz, beträgt die Gesamtlänge 327 Byte, bei einem 2048 Bit RSA-Schlüssel 583 Byte. Um Angriffe zu verhindern, bei denen die Benutzer- oder Karten-Identifikationsnummer (*userID* und *cardID*) gefälscht ist, werden diese in die HMAC-Signatur mit einbezogen (siehe Abbildung 4.2). Außerhalb der Java-Karte wird der signierte öffentliche Schlüssel als binärer Datenblock behandelt und als solcher übertragen oder in der Datenbank abgespeichert.

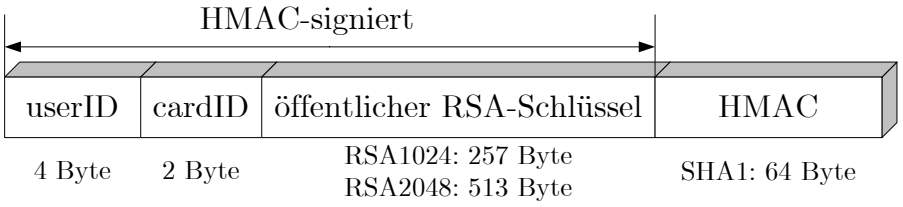

Abbildung 4.2: Aufbau des HMAC-signierten öffentlichen Schlüssels

4.1.4.2 Verschlüsselte Datenschlüssel

Der Aufbau der verschlüsselten Datenschlüssel ist sehr einfach gehalten (siehe Abbildung 4.3). Wie beim signierten öffentlichen Schlüssel beginnt ein Datenschlüssel mit der Benutzer- und Karten-Identifikationsnummer (*userID* und *cardID*), gefolgt von der Identifikationsnummer des Datenschlüssels. Am Ende befindet sich der RSA-verschlüsselte 3DES-Schlüssel. Da die Blocklänge des RSA-Algorithmus mit 128 Byte bzw. 256 Byte (bei 1024 Bit bzw. 2048 Bit Schlüssellänge) wesentlich größer ist als die Schlüssellänge des 3DES-Schlüssels mit 21 Byte (168 Bit), wird der verbleibende Platz nach dem Padding-Verfahren aufgefüllt, das im PKCS#1 v1.5 (Public-Key Cryptography Standards) beschrieben ist. Genau wie die signierten öffentlichen Kartenschlüssel werden auch die verschlüsselten Datenschlüssel nur auf den Java-Karten erzeugt und interpretiert.

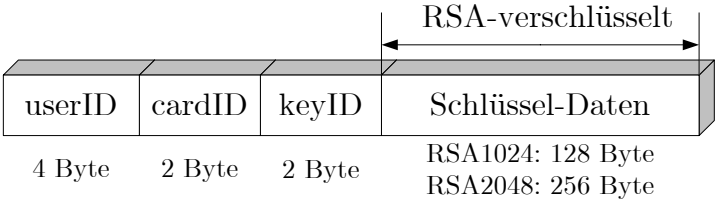

Abbildung 4.3: Aufbau eines verschlüsselten Datenschlüssels

4.1.4.3 Ticket

Alle Daten eines Tickets sind mit dem öffentlichen Schlüssel der Datenbank-Zugriffsschicht verschlüsselt. Die Tickets nehmen deshalb unter den Datenformaten eine Sonderstellung ein, da sie als einzige auf den Java-Karten erzeugt werden, aber nicht dort interpretiert werden. Die interne Struktur eines Tickets ist in Abbildung 4.4 dargestellt. Damit die Datenbank-Zugriffsschicht beim Einlösen das Ticket einem Benutzer zuordnen kann, enthalten die ersten vier Byte die Benutzer-Identifikationsnummer (*userID*).

Da ein Ticket das Auslesen von sensiblen Informationen ermöglicht, gibt es die Möglichkeit, die Gültigkeit jedes Tickets zeitlich zu beschränken. Das Ablaufdatum wird im Format „Millisekunden seit dem 1.1.1970 00:00“ im Ticket gespeichert. Dieses

Format benötigt 8 Byte und verbraucht damit relativ viel Speicherplatz. Zudem erlaubt es die Angabe eines Zeitpunktes mit der Genauigkeit einer Millisekunde und ist damit genauer als benötigt. Das maximal verarbeitbare Datum mit diesem Format liegt weit jenseits des Jahres 3000, was in jedem Fall ausreichend für diese Implementierung ist. Der Grund für die Wahl dieses Datumsformats liegt darin, dass es sich um das Standardformat in Java handelt. Bedenkt man die Vielzahl bereits im Einsatz befindlicher Varianten einen Zeitpunkt darzustellen, erscheint es dem Autor angebracht, ein bereits etabliertes Datumsformat zu verwenden, anstatt der langen Liste ein neues hinzuzufügen.

Vor der Ticketerstellung müssen alle Datenschlüssel auf die *Mobile* Java-Karte transferiert worden sein. Die vorliegende Implementierung beschränkt sich darauf, Tickets mit maximal vier Datenschlüsseln zu erzeugen. Diese Grenze wird bestimmt von der maximalen Länge eines Datenpaketes, das von der Java-Karte ausgegeben werden kann. Werden mehr als diese vier Schlüssel benötigt, müssen diese auf mehrere Tickets verteilt werden. Das Datenformat des Tickets hingegen ist nicht auf vier Schlüssel beschränkt. Theoretisch sind bis zu 32767 Datenschlüssel in einem Ticket möglich. Jeder der enthaltenen Schlüssel besteht aus der Identifikationsnummer (*keyID*) und dem dazugehörigen Datenschlüssel.

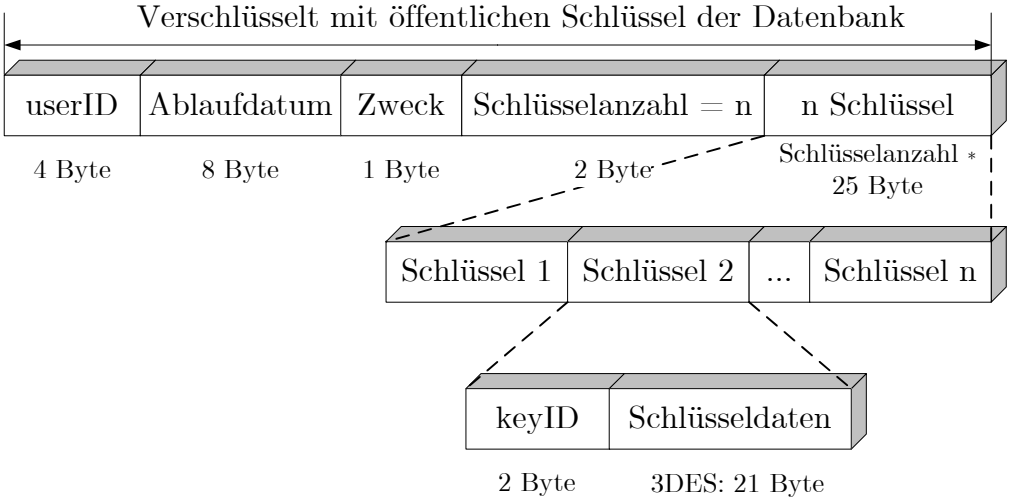

Abbildung 4.4: Aufbau eines Tickets

4.2 *Mobile* Client

Der *Mobile* Client ist eine Software, die auf dem Endgerät des Benutzers installiert wird. Sie hat vier Aufgaben zu erfüllen: Erstens die Authentifikation des Benutzers gegenüber der *Mobile* Java-Karte, zweitens die Weiterleitung von Anfragen an den *Mobile* Server (Proxy-Funktionalität), drittens das Bereitstellen der webbasierten Benutzungsoberfläche, und viertens regelt er die Kommunikation mit lokal an den

Computer angeschlossenen Geräten wie dem Chipkartenlesegerät oder einem Gerät zur Positionsbestimmung (z. B. GPS-Empfänger).

Um möglichst viele verschiedene Endgeräte mit der *Mobile*-Plattform nutzen zu können, muss für jede der drei im Abschnitt 3.4.1 vorgestellten Geräteklassen eine angepasste Implementierung des *Mobile* Client erfolgen. Diese Arbeit beschränkt sich auf die Implementierung des *Mobile* Client für die Geräteklasse der Personal-Computer. Für die beiden anderen Geräteklassen wird nur ermittelt, wie eine Implementierung erfolgen könnte.

Die Implementierung des *Mobile* Clients für die Geräteklasse der Personal-Computer erfolgt auf Basis der „Java 2 Standard Edition“ (J2SE). Eine Implementierung dieser Laufzeitumgebung (JVM) für Java-Programme steht für die verschiedensten Betriebssysteme zur Verfügung. Neben den Implementierungen, die Sun Microsystems für die Betriebssysteme Windows, Linux und Solaris anbietet[4], kann unter Linux auch alternativ „Blackdown-Java“[5] von IBM genutzt werden.

Auf Java-fähigen Mobiltelefonen und vielen Handheld-Computern steht eine JVM für die „Java 2 Micro Edition“ (J2ME) zur Verfügung. Die Plattform J2ME wurde für Geräte wie eingebettete Systeme (in einem Chip zusammengefasste Mini-Computer), Mobiltelefone, Handheld-Computer und Set-Top-Boxen entwickelt. Da die Unterschiede zwischen diesen Geräten bezüglich der Hardware sehr groß sind, definiert J2ME mehrere „Profile“ mit unterschiedlichem Funktionsumfang. Angefangen vom Profil „CLDC“ (Connected Limited Device Configuration), das einfache Funktionen und Standard-Datentypen umfasst, über zum „MIDP“ (Mobile Information Device Profile), das unter anderem Klassen für eine graphische Benutzungsoberfläche enthält, bis zu speziellen Profilen wie das Location API, mit dem aus J2ME-Programmen heraus die aktuelle Positionsinformation ausgelesen werden kann, beispielsweise aus einem im Gerät integrierten GPS-Empfänger. Die Funktionalität, die einem J2ME-Programm zur Verfügung steht, ist deshalb abhängig von den unterstützten Profilen durch die Laufzeitumgebung, die JVM. In Mobiltelefonen ist diese JVM ein fester Teil des Betriebssystems und kann nicht vom Benutzer ersetzt oder erweitert werden, um zusätzliche Profile nutzen zu können. Auf PDAs und Smartphones hingegen kann der Benutzer eine andere Laufzeitumgebung installieren oder die vorhandene ersetzen. Die am weitesten verbreitete JVM für PDAs ist die „J9 VM“ von IBM, die es sowohl für die Betriebssysteme PalmOS, WindowsCE und Windows Mobile 2003 gibt. Einige Betriebssysteme wie „Symbian OS“ enthalten sogar bereits eine J2ME-VM.

[4] `http://java.sun.com/j2se/downloads/index.html`
[5] `http://www.blackdown.org`

In den folgenden Abschnitten werden zuerst die Probleme beim Zugriff auf das Chipkartenlesegerät unter den verschiedenen Java-Versionen untersucht. Danach geht es um die Entscheidung, welcher Webserver für die Bereitstellung der webbasierten Benutzungsoberfläche in den *Mobile* Client integriert wird. Nach einem genaueren Einblick in den modularen Aufbau des *Mobile* Clients und die Zusammenarbeit der Module mit dem Webserver folgt die Vorstellung der Abläufe bei der Benutzer-Authentifikation. Der darauf folgende Abschnitt geht genauer auf die Schritte ein, die für die Installation des *Mobile* Clients auf dem Endgerät des Benutzers notwendig sind. Der letzte Abschnitt über den *Mobile* Client zeigt einige Bildschirmfotos der webbasierten Benutzungsoberfläche und erklärt die Funktionen der darauf sichtbaren Eingabeelemente.

4.2.1 Kommunikation mit der Java-Karte

Eine der Schwierigkeiten bei der Implementierung des *Mobile* Clients ist die Kommunikation mit der *Mobile* Java-Karte. Nur wenige aktuelle Personal-Computer verfügen standardmäßig über ein Chipkartenlesegerät. Zum Nachrüsten existiert eine Vielzahl von externen Chipkartenlesegeräten, die über die USB- oder die serielle Schnittstelle an den Computer angeschlossen werden. Der Anschluss erfordert kein Fachwissen und lässt sich auch von ungeübten Anwendern durchführen. Die Ansteuerung des Chipkartenlesegerätes aus dem *Mobile* Client heraus ist das eigentliche Problem. Für die Plattform J2SE existiert bisher keine einheitliche Schnittstelle, um mit einem angeschlossenen Chipkartenlesegerät zu kommunizieren. Alternativ gibt es mehrere Bibliotheken, die den Zugriff über betriebssystemspezifische Funktionen ermöglichen, unter anderem das Open Card Framework (OCF)[6] und JPCSC[7]. Beide Bibliotheken lassen sich sowohl unter Windows, wie auch unter Linux nutzen. Dabei verwenden sie die vom Betriebssystem zur Verfügung gestellte PC/SC-Schnittstelle, um das Chipkartenlesegerät anzusprechen. Diese von der PC/SC Workgroup[8] standardisierte Schnittstelle ermöglicht es Daten mit beliebigen Prozessor-Chipkarten auszutauschen. Neben der PC/SC-Schnittstelle gib es noch das Card Terminal Application Programming Interface (CT-API)[DGTT98], das wesentlich hardwarenaher und damit schwieriger zu programmieren ist. Dafür bietet es die Möglichkeit, die erweiterten Funktionen von Chipkartenlesegeräten, wie ein PIN-Eingabefeld oder ein Display, zu nutzen. Außerdem kann das CT-API auch mit Speicher-Chipkarten (ohne Prozessor) umgehen. Wegen der einfacheren Programmierung und der besseren Unterstützung der verschiedenen Chipkartenlesegeräte unter Windows und Linux kommt im *Mobile* Client für die Plattform J2SE die PC/SC-Schnittstelle zum Einsatz. Der Zugriff erfolgt über die JPCSC-Bibliothek, da der Quellcode dieser

[6] `http://www.opencard.org/index-downloads.shtml`
[7] `http://www.linuxnet.com/middleware/files/jpcsc-0.8.0-src.zip`
[8] `http://www.pcscworkgroup.com`

Bibliothek verfügbar ist und sie im Gegensatz zum OCF wesentlich schlanker und damit leichter in den *Mobile* Client integrierbar ist.

Im Gegensatz zu den Personal-Computern verfügt jedes Mobiltelefon über ein eingebautes Chipkartenlesegerät, in dem sich die SIM-Karte befindet. Bei den J2ME-Programmierern kam schnell der Wunsch auf, diese ohnehin vorhandene Hardware für eigene Anwendungen zu nutzen. Dazu wurde eine einheitliche Programmierschnittstelle definiert, über die sich sowohl Java-Karten, als auch beliebige Chipkarten ansprechen lassen[9]. Zusammen mit anderen Funktionen aus dem Bereich der Kryptographie wurde diese Schnittstelle 2004 als Profil Security and Trust Services API for J2ME (SATSA) zusammengefasst. Aktuell (November 2005) ist noch kein Mobiltelefon erhältlich, das diese Erweiterung unterstützt. Erste Modelle, die SATSA unterstützen, werden für Anfang 2006 erwartet. Da der *Mobile* Client nicht ohne den Zugriff auf ein Chipkartenlesegerät (und damit auf die *Mobile* Java-Karte) funktioniert, ist es im Moment noch nicht möglich einen *Mobile* Client für die Plattform J2ME zu implementieren und auf einem realen Gerät zu testen.

Abgesehen von Smartphones, einer Kombination aus PDA und Mobiltelefon, verfügen die Geräte der Handheld-Klasse nicht über ein eingebautes Chipkartenlesegerät. Für einige Modelle gibt es die Möglichkeit, ein Chipkartenlesegerät nachzurüsten. Die Verbindung erfolgt dabei entweder über eine vorhandene CompactFlash-Schnittstelle[10], über die SecureDigitalIO-Schnittstelle (SDIO)[11] oder über die PCMCIA-Schnittstelle, für die es eine große Auswahl an Chipkartenlesegeräten gibt. Hat man den Handheld-Computer mit dem erforderlichen Chipkartenlesegerät erweitert, bleibt noch das Problem, dieses aus einem Java-Programm heraus anzusprechen. Einige der aktuellen Handheld-Computer bringen bereits eine Ausführungsumgebung für J2ME-Programm mit, bisher allerdings noch ohne die Erweiterung SATSA. Für die Implementierung des *Mobile* Client auf Handheld-Computern bleibt im Moment nur die Entwicklung einer eigenen Bibliothek für jedes unterstützte Betriebssystem, um auf ein Chipkartenlesegerät zugreifen zu können.

4.2.2 Der integrierte Webserver

Für die Bereitstellung der webbasierten Benutzungsoberfläche ist es notwendig, den *Mobile* Client mit einem Webserver auszustatten. Die Implementierung eines eigenen Webservers ist nicht notwendig, da es mehrere in Java geschriebene Webserver gibt, die für eine Integration in den *Mobile* Client in Frage kommen. Zudem sind einige dieser Webserver in der Lage auch als Proxy zu arbeiten, womit auch diese Funktionalität nicht von Grund auf neu programmiert werden muss.

[9] JSR177: `http://jcp.org/en/jsr/detail?id=177`
[10] z. B. SpringCard-CF: `http://www.pro-active.fr/products/sc-cf`
[11] z. B. SpringSIM-SD: `http://www.pro-active.fr/products/ss-sd`

Auf eine mögliche Integration untersucht, wurden die Webserver:

- Apache Tomcat[12] Version 5.5.9
- Jetty[13] Version 5.1.3
- Simple HTTP Engine[14] Version 2.5.4
- Tiny Java Web Server (Tjws)[15] Version 1.7a

Bis auf „Tomcat“ sind alle genannten Webserver explizit für die Integration in eine Anwendung entwickelt worden. Das umfasst sowohl die Beanspruchung von Speicher und Prozessor zur Laufzeit, als auch die Datei-Größe der Bibliothek, in der die Webserver-Funktionalität implementiert ist. Beide Eigenschaften sind wünschenswert, aber nicht notwendig für die Implementierung des *Mobile* Clients. Deshalb kann „Tomcat“ noch nicht ausgeschlossen werden, auch wenn die minimale Konfiguration mindestens 2 Megabyte Festplattenplatz benötigt und damit bis zu 20 mal mehr als die anderen Webserver. Ein Ausschlusskriterium hingegen ist die fehlende Möglichkeit des „Tiny Java Web Server“, bestimmte Bereiche des Webservers erst nach einer erfolgreichen Authentifikation zugänglich zu machen. Ähnliches gilt für die „Simple HTTP Engine“. Diese enthält nur rudimentäre Unterstützung für die Implementierung einer Authentifikation. Die beiden verbleibenden Webserver „Tomcat“ und „Jetty“ bieten umfangreiche Unterstützung für die Implementierung einer eigenen Authentifikation und eignen sich damit ideal für eine Integration in den *Mobile* Client. Die Wahl zwischen diesen beiden wurde letztendlich zu Gunsten von „Jetty“ entschieden, da dieser über das Modul *ProxyHandler* verfügt, welches die Funktionalität eines Proxy anbietet. Auf Basis dieses Moduls ist es sehr leicht, die benötigte Weiterleitung von Anfragen vom *Mobile* Client an den *Mobile* Server zu realisieren.

4.2.3 Aufbau des *Mobile* Clients

Der Aufbau des *Mobile* Clients unterteilt sich in zwei Bereiche (siehe Abbildung 4.5). Der erste (obere) Bereich umfasst die Module, in denen die Client-Funktionalität implementiert ist, die unabhängig von der Benutzungsoberfläche ist. Dazu gehören: Das Modul für den Zugriff auf die Java-Karte und das Webservice-Modul, über das Funktionen auf dem *Mobile* Server aufgerufen werden können. Der zweite (untere) Bereich enthält den integrierten Webserver Jetty, der die webbasierte Benutzungsoberfläche erzeugt. Innerhalb des Webservers arbeiten zu diesem Zweck verschiedene Module für die Authentifikation des Benutzers, ein Proxy-Modul für die Weiterleitung von Anfragen an den *Mobile* Server und einige Servlets, dynamisch generierte Websei-

[12] `http://tomcat.apache.org`
[13] `http://jetty.mortbay.org/jetty`
[14] `http://simpleweb.sourceforge.net`
[15] `http://tjws.sourceforge.net`

ten, die bestimmte Funktionen über die webbasierte Benutzungsoberfläche anbieten. Ergänzt wird diese Oberfläche durch statische Ressourcen wie HTML-Seiten, Bilder und Cascading Style Sheets (CSS)-Dateien, in denen Formatierungsanweisungen für alle Webseiten definiert sind.

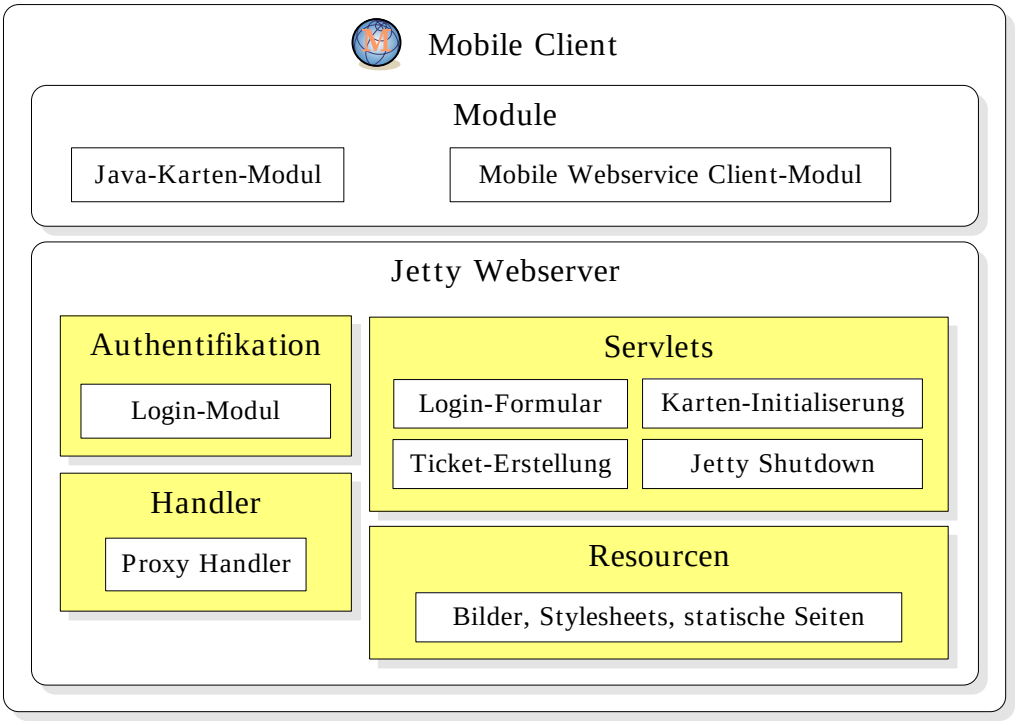

Abbildung 4.5: Aufbau des *Mobile* Clients

Einige der Servlets lassen sich ohne vorherige Authentifizierung nutzen, beispielsweise das Login-Formular, in das der Benutzer die Karten-PIN für die Authentifikation eingibt, oder das Servlet für die Karten-Initialisierung, mit dem sich neue *Mobile* Java-Karten initialisieren und dem System hinzufügen lassen. Auch ohne Authentifikation nutzbar ist das „Shutdown-Servlet“, das es erlaubt den *Mobile* Client direkt über die webbasierte Benutzungsoberfläche zu beenden. Außerdem sind alle statischen Ressourcen frei zugänglich, da sie sonst nicht auf allen Webseiten verwendet werden können. Alle übrigen Servlets, sowie der Proxy-Handler, sind nur nach einer erfolgreichen Authentifikation nutzbar.

4.2.3.1 Das Webservice Client-Modul

Das Webservice-Modul ermöglicht die Nutzung der Funktionen, die vom *Mobile* Server über die Webservice-Schnittstelle zur Verfügung gestellt werden. Um welche Funktionen es sich dabei genau handelt, wird im Abschnitt 4.3.1.1 vorgestellt. Der Aufruf einer Funktion über einen Webservice erfolgt ähnlich wie bei der Remote Method Invocation (siehe Abschnitt 2.2.7.1). Statt eines binären Byte-Stroms wie bei RMI, werden die Daten bei Webservices durch Textdateien in der Beschreibungssprache Extensible Markup Language (XML) übertragen. Die Struktur dieser

XML-Dateien ist durch das im Simple Object Access Protocol (SOAP) festgelegte XML-Schema bestimmt. Die Übersetzung des Java-Aufrufs in eine solche SOAP-Nachricht und umgekehrt erfolgt durch Axis[16], der Webservice-Implementierung des Apache Web Services Projektes. Der Transport der SOAP-Nachrichten erfolgt in dieser Implementierung über das Hypertext Transfer Protocol (HTTP). Um die Vertrauchlichkeit und die Integrität der übertragenen Daten sicherzustellen, wird das Protokoll Transport Layer Security (TLS) verwendet. Die Authentizität des *Mobile* Servers wird dabei über ein X.509-Zertifikat gesichert, der Client authentifiziert sich über den Benutzernamen und das dazugehörige Passwort.

4.2.3.2 Das Java-Karten-Modul

Über das Java-Karten-Modul stellt der *Mobile* Client eine Verbindung mit der *Mobile* Java-Karte her, um deren Funktionalität nutzen zu können. Durch die Einschränkungen der Java-Karten-Plattform (z. B. beim Bearbeiten größerer Datenblöcke wie in Abschnitt 4.1.3 dargestellt) war es nicht möglich, die durch die *Mobile* Java-Karte bereitgestellte Funktionalität im Hinblick auf einfache Nutzbarkeit aus dem *Mobile* Client zu entwerfen. In vielen Fällen müssen mehrere Funktionen der Java-Karte in der richtigen Reihenfolge aufgerufen werden, um ein bestimmtes Ergebnis zu erhalten. Die Klasse *MobileJavaCard* wurde entwickelt, um diese Komplexität vor dem *Mobile* Client zu verstecken und so die *Mobile* Java-Karte einfacher in den anderen Modulen nutzen zu können. Diese Klasse kapselt die Funktionalität der *Mobile* Java-Karte und fasst sie zu einfach verwendbaren Funktionen zusammen. Dazu gehört die Zerteilung und das blockweise Übertragen der Daten auf die *Mobile* Java-Karte. In Zusammenarbeit mit der *Serializer*-Klasse werden zudem alle zu übertragenden Daten in ein geeignetes, von der Java-Karte unterstütztes, Format konvertiert. Auf der *Mobile* Java-Karte werden die Objekte wieder deserialisiert, wie es in Abschnitt 4.1.2 vorgestellt wurde.

Einige Funktionen des Java-Karten-Moduls benötigen sowohl eine Verbindung zur Java-Karte als auch eine Verbindung zum *Mobile* Server über das Webservice Client-Modul. Soll beispielsweise ein Ticket erzeugt werden, wie es im Abschnitt 3.3.3.2 beschrieben wird, sind zum Ausstellen des Tickets mehrere Schritte notwendig, die in der Abbildung 3.4 zusammengefasst wurden. Dazu gehört das Laden der benötigten verschlüsselten Datenschlüssel aus der Datenbank und der Transfer auf die Java-Karte. Dort werden sie entschlüsselt und können danach zu einem Ticket zusammengestellt und verschlüsselt werden. Im letzten Schritt wird dann das Ticket an den *Mobile* Server übertragen. Diese Abfolge von Schritten fasst das Java-Karten Modul in der Funktion *createTicketAndSendToAssistant(..)* zusammen. Weitere Funktionen für die Aktualisierung der Liste der verschlüsselten Datenschlüssel auf dem *Mobi-*

[16] `http://ws.apache.org/axis`

le Server oder das Laden eines Datenschlüssels aus der Datenbank auf die *Mobile* Java-Karte stehen ebenso zur Verfügung.

4.2.3.3 Zusammenarbeit der Module

Einen Überblick über die beiden vorgestellten Module zeigt Abbildung 4.6 in Form eines UML-Diagrammes. Die Klassen und Interfaces in der oberen Hälfte gehören zum Java-Karten-Modul, die in der unteren zum *Mobile* Webservice Client-Modul. Die Schnittstelle der *Mobile* Java-Karte (in der Abbildung zusammengefasst zum „Mobile Java-Karten Interface“) ist bereits bekannt aus der Vorstellung der Klassenstruktur in Abschnitt 4.1.2. Diese wird verwendet von der Klasse *MobileJavaCard*, die den größten Teil der Funktionalität des Java-Karten-Moduls implementiert. Diese Klasse wird beerbt von der Klasse *MobileJavaCardIssuance*, die zusätzliche Funktionen für die Initialisierung einer neuen *Mobile* Java-Karte bereitstellt. Im *Mobile* Webservice Client-Modul verwaltet die Klasse *ServerConnection* die Verbindung zu den zwei Webservice-Implementierungen, welche die Kommunikation mit der Datenbank bzw. dem Assistenten auf dem *Mobile* Server ermöglichen.

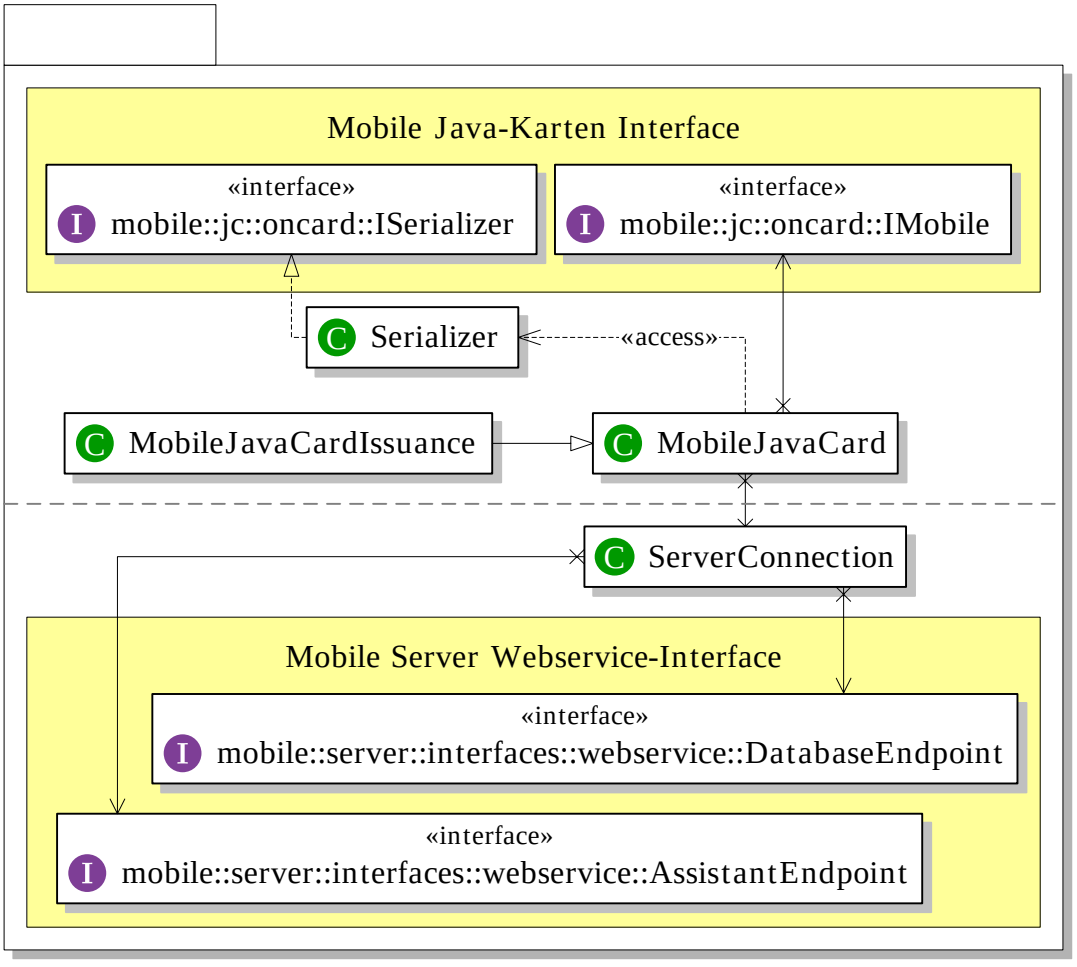

Abbildung 4.6: UML Klassendiagramm der *Mobile* Client Module (*mobile.jc.offcard*)

4.2.4 Der Authentifikationsvorgang

Die Authentifikation des *Mobile* Clients erfolgt über die Eingabe der Karten-PIN im Login-Formular, das die Daten an das Login-Modul weiterleitet. Dieses verwendet das Java-Karten-Modul, um die Karten-PIN an die *Mobile* Java-Karte zu senden. Auf der Karte wird die eingegebene Pin mit der intern gespeicherten PIN verglichen. Stimmt die PIN nicht, wird ein interner Fehlversuchszähler erhöht. Nach insgesamt fünf Fehlversuchen wird die Karte für weitere Anmeldeversuche gesperrt. Bei einer übereinstimmenden PIN wird der Zugriff auf die Karte freigeschaltet und der Fehlversuchszähler auf null zurückgesetzt. Direkt nach einer erfolgreichen Authentifikation gegenüber der *Mobile* Java-Karte nimmt der *Mobile* Client Verbindung zum *Mobile* Server über die Webservice-Schnittstelle auf. Auf jeder *Mobile* Java-Karte befindet sich ein beim Initialisieren der Karte zufällig erzeugtes Passwort, über das sich die Karte gegenüber dem *Mobile* Server ausweisen kann. Der dazugehörige Karten-Benutzername setzt sich aus der Karten-ID und der Benutzer-ID zusammen. Ursprünglich war eine clientseitige Authentifikation über X.509-Zertifikate angedacht, diese ließ sich wegen der fehlenden Unterstützung von Zertifikaten durch die Java-Karte jedoch nicht realisieren. Der gesamte Ablauf der Authentifikation ist dargestellt in Abbildung 4.7.

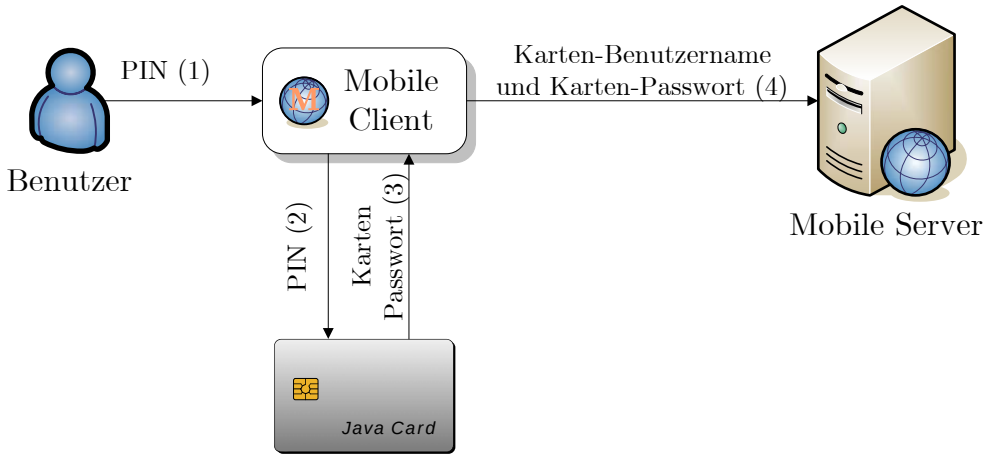

Abbildung 4.7: Anmeldung am *Mobile*-Server

Durch die Authentifikation gegenüber dem *Mobile* Server wird sichergestellt, dass die verwendete Karte korrekt in der Datenbank eingetragen und nicht als gesperrt markiert ist. Unmittelbar nach der Anmeldung am *Mobile* Server überprüft der *Mobile* Client, ob seit der letzten Anmeldung mit der aktuell verwendeten *Mobile* Java-Karte vom Benutzer dem System neue Karten hinzugefügt wurden. Außerdem wird die Liste der verschlüsselten Datenschlüssel auf Einträge überprüft, die nicht für alle Karten verschlüsselt vorliegen. Existieren solche Schlüssel, werden diese automatisch vom *Mobile* Client aktualisiert. Der genaue Ablauf dieser zwei Vorgänge ist identisch

mit der Synchronisation, nachdem dem System eine neue Karte hinzugefügt wurde (siehe Abschnitt 3.3.4.1).

4.2.5 Installation und Start des Mobile Clients

Um den *Mobile* Client auf einem Rechner zu installieren genügt es, alle Dateien aus dem `client`-Verzeichnis[17] zu kopieren. Dieses enthält die benötigten Bibliotheken und Dateien in einer speziellen Verzeichnisstruktur. Das Erzeugen dieser Struktur, der Bibliotheken sowie das Kopieren aller benötigten Dateien erfolgt automatisiert über das Skriptsystem Apache Ant[18]. Das Verzeichnis `conf` enthält die Konfigurationsdatei „client.properties“, in der man den Port festlegt, auf dem der lokale Webserver des *Mobile* Client die webbasierte Benutzungsoberfläche bereitstellt.

Um den Start des *Mobile* Client für den Benutzer so einfach wie möglich zu machen, existiert ein Java-Archiv (`Startup.jar`) im Hauptverzeichnis, das alle notwendigen Startparameter automatisch setzt. Durch einen Doppelklick auf diese Datei oder den Befehl `java -jar Startup.jar` wird der *Mobile* Client gestartet. Dieser öffnet automatisch den Standard-Webbrowser und lässt diesen die webbasierte Benutzungsoberfläche anzeigen.

4.2.6 Benutzungsoberfläche

Nach dem Start des *Mobile* Client kommt man zuerst auf die Login-Seite, wie sie in Abbildung 4.8 dargestellt ist. Diese zeigt alle verfügbaren Chipkartenlesegeräte in einer Liste an. Für jedes Chipkartenlesegerät wird geprüft, ob sich eine Chipkarte darin befindet und ob diese die *Mobile* Karten-Anwendung enthält. Um dem Benutzer die Verwendung zu vereinfachen, wird automatisch eine Vorauswahl getroffen, wenn eine funktionsfähige *Mobile* Java-Karte gefunden wurde. Besitzt der Benutzer eine funktionsfähige *Mobile* Java-Karte, kann er sich über die Eingabe der Karten-PIN im unteren Bereich der Login-Seite gegenüber der Karte authentifizieren. Nach einer erfolgreichen Authentifikation wird automatisch die Hauptseite des *Mobile* Servers über den *Mobile* Client als Proxy aufgerufen. Alle vom *Mobile* Client generierten Webseiten enthalten einen Link, der es erlaubt den *Mobile* Client zu beenden.

Besitzt der Benutzer noch keine funktionsfähige *Mobile* Java-Karte oder will er dem System eine neue Karte hinzufügen, kann er dies auf einer eigenen Seite (siehe Abbildung 4.9) tun, die er über den Link „Karte neu initialisieren“ erreicht. Alle für die Initialisierung einer neuen *Mobile* Java-Karte nötige Angaben werden auf dieser Seite gemacht (siehe auch Abschnitt 3.3.4.1).

[17] siehe CD im Anhang
[18] `http://ant.apache.org`

Abbildung 4.8: Bildschirmfoto des Login-Dialoges

Abbildung 4.9: Bildschirmfoto der Weboberfläche, das den Dialog zum Hinzufügen einer neuen Karte zeigt

Auf die Vorstellung weiterer Seiten der webbasierten Benutzungsoberfläche (z. B. zur Eingabe von Daten) wird in dieser Arbeit verzichtet, da diese nur für die Demonstration der Funktionsfähigkeit des Systems implementiert wurden.

4.3 *Mobile* Server

Der *Mobile* Server ist der zentrale Dienst, der von allen Benutzern der *Mobile*-Plattform genutzt wird. Ein Ausfall dieses Dienstes ist gleichbedeutend mit einem Ausfall der gesamten *Mobile*-Plattform. Deshalb wurde bei der Implementierung des *Mobile* Servers besonders auf Zuverlässigkeit und Stabilität geachtet. Ein weiterer Schwerpunkt bei der Implementierung war die Absicherung des Servers gegen unbefugte Benutzung und unberechtigte Manipulation von Daten. Um diese Anforderungen zu erfüllen, wurde entschieden, den *Mobile* Server als eine J2EE-Anwendung zu implementieren. Dabei handelt es sich um einen Standard für die Entwicklung von Java-basierten Anwendungen im Unternehmens-Umfeld. J2EE-Anwendungen werden bereits seit einigen Jahren mit großem Erfolg verwendet, es handelt sich also um eine Technologie, die seit langem ihre Fähigkeiten tagtäglich unter Beweis stellt. Eine J2EE-Anwendung ist keine eigenständige Anwendung, sie ist ein Programm-Modul, das innerhalb eines J2EE-Anwendungsservers ausgeführt wird. Der Anwendungsserver stellt für die J2EE-Anwendung verschiedene Funktionen zur Verfügung, beispielsweise den Zugriff auf eine Datenbank. Eine J2EE-Anwendung besteht üblicherweise aus zwei Schichten, eine Schicht der Anwendungslogik und eine Präsentationsschicht. Die Anwendungslogik implementiert alle Funktionen, die innerhalb der J2EE-Anwendung benötigt werden. Sie besteht aus ein oder mehreren Modulen, die als Enterprise JavaBean (EJB) bezeichnet werden. Die Präsentationsschicht erzeugt die webbasierte Benutzungsoberfläche und verwendet dabei die bereitgestellten Funktionen der Anwendungslogik. Für die Erzeugung der dynamischen Webseiten existieren zwei Möglichkeiten, zum Einen die Technologie der Java Server Pages (JSP), bei der eine HTML-Seite zusätzlich auch Java-Code enthält. Dieser wird vor der Auslieferung durch den Webserver ausgeführt und ermöglicht so das Einfügen von dynamischen HTML-Code. Zum Anderen die Servlet-Technologie, bei der eine Java-Klasse den gesamten HTML-Code dynamisch erzeugt. Innerhalb einer J2EE-Anwendung können beide Technologien verwendet werden.

Im Folgenden werden zuerst der verwendete J2EE-Anwendungsserver und die dafür entwickelten Module genauer betrachtet. Danach wird die für die Implementierung verwendete Datenbank vorgestellt und das Zusammenspiel der Datenbank-Tabellen erklärt.

4.3.1 J2EE Anwendungs-Server

Nach der Entscheidung, den *Mobile* Server als J2EE-Anwendung zu implementieren, bleibt nur noch die Frage, welcher J2EE-Anwendungsserver zum Einsatz kommen soll. Es existieren eine ganze Reihe von Anwendungsservern, die den J2EE-Standard implementieren[19]. Die Wahl fiel schließlich auf den Anwendungsserver „JBoss“[20], da dieser unter der GNU Lesser General Public License (LGPL)[21], einer anerkannten Open-Source-Lizenz steht und somit kostenlos ist und der vollständige Java-Quellcode zur Verfügung steht. Zusätzlich darf Software, die unter dieser Lizenz steht, in kommerziellen Produkten eingesetzt werden, ohne diese zwangsweise wieder unter der LGPL oder einer anderen Open-Source-Lizenz veröffentlichen zu müssen. Außerdem besitzt JBoss eine sehr aktive Gemeinschaft von Entwicklern, die einige sehr gute Einführungen in die Verwendung von JBoss erstellt haben. Für die Implementierung des *Mobile* Servers wurde die aktuelle Version 4 des JBoss-Servers verwendet, diese beinhaltet den Webserver Apache Tomcat v5.5.

Die Anwendungslogik des *Mobile* Servers ist unterteilt in vier Bereiche. Eine Übersicht, zusammen mit den Modulen der Präsentationsschicht, zeigt Abbildung 4.10. Der erste Bereich der Anwendungslogik enthält das Authentifikations-Modul, das die vom *Mobile* Client übergebenen Anmelde-Daten überprüft. Der zweite Bereich enthält zwei Webservice-Module, über die der *Mobile* Client auf die Datenbank zugreifen oder mit dem Assistenten kommunizieren kann. Der dritte Bereich enthält den Assistenten und der letzte Bereich die Datenbank-Zugriffsschicht. Die Präsentationsschicht enthält zwei einfache Servlets. Das Erste zeigt das im Assistenten vorhandene Ticket an, und das Zweite erlaubt dem Administrator des *Mobile* Servers die Benutzerkonten zu verwalten. Die verschiedenen Module werden in folgenden Abschnitten noch genauer beschrieben.

4.3.1.1 Webservices

Der *Mobile* Server ist sowohl Nutzer, als auch Anbieter von Diensten in Form von Webservices. In diesem Abschnitt geht es nur um die Webservices, welche vom *Mobile* Server angeboten werden. In der vorliegenden Implementierung sind das zwei Dienste. Der erste Webservice ermöglicht die Kommunikation mit dem auf dem Server laufenden Assistenten. Dieser dient beispielsweise dazu, von der *Mobile* Java-Karte ausgestellte Tickets an den Assistenten zu übertragen oder Details einer gestellten Aufgabe abzufragen. Mit Hilfe des zweiten Webservice können Lese-/ oder Schreiboperationen in der Datenbank ausgeführt werden. Alle Datenbank-Zugriffe über den

[19] Übersicht der J2EE Anwendungsserver: `http://java.sun.com/j2ee/compatibility.html`
[20] `http://www.jboss.org`
[21] `http://www.gnu.org/copyleft/lesser.html`

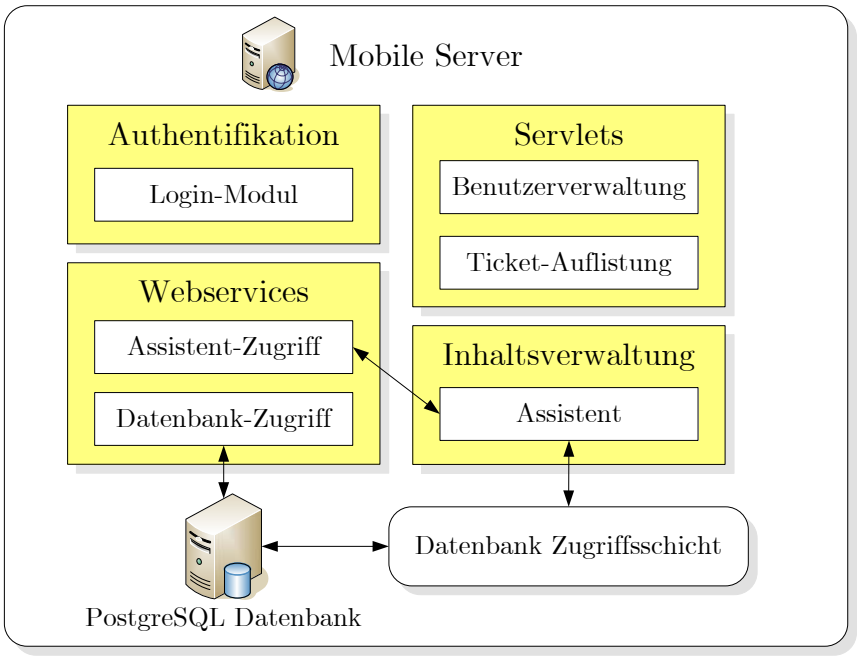

Abbildung 4.10: Module des *Mobile* Servers

Webservice erfolgen ohne den Umweg über die Datenbank-Zugriffsschicht. Ein Zugriff über ein Ticket ist somit auf diesem Weg nicht möglich. Ohne eine passende *Mobile* Java-Karte kann man über den Datenbank-Webservice zwar verschlüsselte Daten auslesen, diese jedoch nicht entschlüsseln.

Bei der Implementierung eines eigenen Webservices für einen J2EE-Server muss man zahlreiche Bedingungen beachten. Neben dem eigentlichen Java-Quelltext der Implementierung muss unter anderem die dazugehörige, sprachenunabhängige Schnittstellenbeschreibung erzeugt werden. Dabei handelt es sich um eine XML-Datei im Format der Web Service Definition Language (WSDL). Auf der Client-Seite wird aus dieser Schnittstellenbeschreibung dann wieder ein Java-Interface erzeugt. Für den Einsatz auf dem JBoss-Server (man spricht dabei vom „deployment“) müssen noch insgesamt vier weitere XML-Dateien mit passendem Inhalt erzeugt werden. Mit diesen konfiguriert man unter anderem den Pfad, unter dem der Webservice auf dem Server erreichbar ist und kann Zugriffe auf den Webservice auf bestimmte authentifizierte Benutzer beschränken. Die Implementierung der Webservices erfolgt über je eine EJB. Die Veröffentlichung der Funktionen dieser EJBs erfolgt automatisch durch das Servlet Webservices for J2EE (WS4EE), ein Modul, das ein Bestandteil des JBoss-Anwendungsservers ist.

4.3.1.2 Assistent

Die Implementierung eines Assistenten ist nicht Ziel dieser Diplomarbeit. Um dennoch die Verwendung von Tickets durch einen Assistenten demonstrieren zu können, enthält die vorliegende Implementierung eine auf die Speicherung eines Tickets be-

schränkte Version. Außerdem fehlt diesem Assistenten die Möglichkeit Aufgaben abzuarbeiten oder Daten aus Webservices externer Anbieter abzufragen. Umgesetzt ist der Assistent als „Java Entity Bean“, eine Technologie, die sich automatisch um die persistente Speicherung des Tickets innerhalb des Assistenten kümmert.

4.3.1.3 Webbasierte Benutzungsoberfläche

Die webbasierte Benutzungsoberfläche enthält einen Bereich, der nur für Benutzer der *Mobile*-Plattform zugänglich ist und einen anderen Bereich, der nur vom Administrator des *Mobile* Servers genutzt werden kann. Der für die Benutzer zugängliche Bereich ermöglicht den Zugriff auf den Assistenten, um neue Aufgaben zu definieren oder die Ergebnisse vorhergehender Aufgaben anzusehen. Da der im Rahmen dieser Arbeit implementierte Assistent nicht die Möglichkeit bietet Aufgaben zu bearbeiten, ist auch die dazu passende webbasierte Benutzungsoberfläche nicht implementiert worden. Es existiert einzig ein Servlet, das Informationen über das im Assistenten gespeicherte Ticket anzeigt. Dies demonstriert, dass der Offline-Zugriff über Tickets wie vorgestellt funktioniert. Zusätzlich wurde noch eine statische HTML-Seite hinzugefügt, die immer nach der Anmeldung eines Benutzers über den *Mobile* Client über das Proxy-Modul vom *Mobile* Server geladen wird. Dadurch lässt sich die Funktionsfähigkeit des Proxy-Moduls demonstrieren.

Der zweite Bereich, der nur für den Administrator des *Mobile* Servers zugänglich ist, erlaubt es neue Benutzerkonten anzulegen, bestehende zu löschen oder das Passwort eines Benutzerkontos neu zu setzen. Die Benutzerkonten des *Mobile* Servers sind in einer Tabelle der Datenbank gespeichert (eine genaue Beschreibung dieser Tabelle folgt in Abschnitt 4.3.2.1). Ohne die Administrations-Weboberfläche bleibt nur der Weg, die Daten dieser Tabelle über SQL-Befehle zu bearbeiten. Das direkte Manipulieren von Datenbanken über die Eingabe von SQL-Befehlen birgt allerdings Risiken. Ein unvollständiger oder fehlerhafter Befehl kann dazu führen, dass Daten in der Datenbank ungewollt verändert oder gelöscht werden. Zudem erfordert das Anlegen eines Benutzerkontos die Eingabe eines Passwortes in Form eines SHA-1 Hashes. Die Administrations-Weboberfläche erleichtert deshalb nicht nur die Verwaltung des *Mobile* Servers, sondern verhindert zusätzlich noch eine mögliche Fehlbedienung durch den Administrator.

Die gesamte Administrations-Weboberfläche ist als eigenständige Webanwendung implementiert, die direkt auf die Datenbank zugreift. Auch die Authentifikation erfolgt unabhängig von den übrigen J2EE-Modulen, die zur *Mobile*-Plattform gehören, sodass kein normaler Benutzer Zugriff auf die Benutzerverwaltung erlangen kann.

4.3.1.4 Datenbank-Zugriffsschicht

Die Implementierung der Datenbank-Zugriffsschicht bedarf besonderer Aufmerksamkeit, da diese gleichzeitig von allen Benutzern (bzw. deren Assistenten) genutzt werden kann um Daten zu entschlüsseln. Eine fehlerhafte Implementierung der Datenbank-Zugriffsschicht kann dazu führen, dass Benutzer Zugriff auf fremde Benutzerdaten erlangen können. Die Datenbank-Zugriffsschicht darf aus diesem Grund nie unverschlüsselte Datenschlüssel oder Benutzerdaten über eine Assistenten-Anfrage hinaus speichern. Sie ist deshalb in Form einer „stateless Session Bean“ implementiert. Sofern man bestimmte Regeln bei der Programmierung einhält (wie z. B. Verzicht auf Klassen-Felder und Instanz-Felder), erfüllt dieser Bean-Typ durch die Eigenschaft der Zustandslosigkeit genau die gestellten Anforderungen, was die Umsetzung der Datenbank-Zugriffsschicht sehr vereinfacht. Nach dem Entschlüsseln des Tickets werden die im Ticket enthaltenen Beschränkungen geprüft, die verschlüsselten Daten aus der Datenbank gelesen, mit den Datenschlüsseln aus dem Ticket entschlüsselt und an den Assistenten zurückgegeben.

4.3.2 Datenbank

Die Aufgabe der persistenten Speicherung aller anfallenden Verwaltungs- und Benutzerdaten auf dem *Mobile* Server übernimmt eine SQL-Datenbank. Datenbanken diesen Typs gibt es von den verschiedensten Herstellern zu den unterschiedlichsten Preisen. Im Bereich der kostenlosen SQL-Datenbanken haben sich zwei Systeme etabliert: MySQL[22] Version 4.0 und PostgreSQL[23] Version 8.0. Als Datenbank für den *Mobile* Server fiel die Entscheidung zu Gunsten von PostgreSQL, da es gegenüber MySQL den Vorteil einer höheren Geschwindigkeit bei vielen gleichzeitigen und komplexen Operationen hat. Außerdem ist PostgreSQL, verglichen mit MySQL, weitgehend konform mit dem Standard ANSI SQL99. Dies minimiert die notwendigen Modifikationen am *Mobile* Server, falls ein Betreiber eine andere ANSI SQL99-konforme Datenbank, statt PostgreSQL, für die Speicherung der Daten verwenden will.

Das verwendete Datenbankschema mit den fünf Tabellen zeigt Abbildung 4.11. In folgenden Abschnitten werden die Tabellen und die darin gespeicherten Daten genauer vorgestellt.

4.3.2.1 Die Tabelle *users*

In der Tabelle *users* befindet sich die Benutzerverwaltung des Systems. Jedem Benutzer wird eine eindeutige Zahl zugeordnet, die Benutzer-ID oder *userID*. Diese

[22] `http://www.mysql.com`
[23] `http://www.postgresql.org`

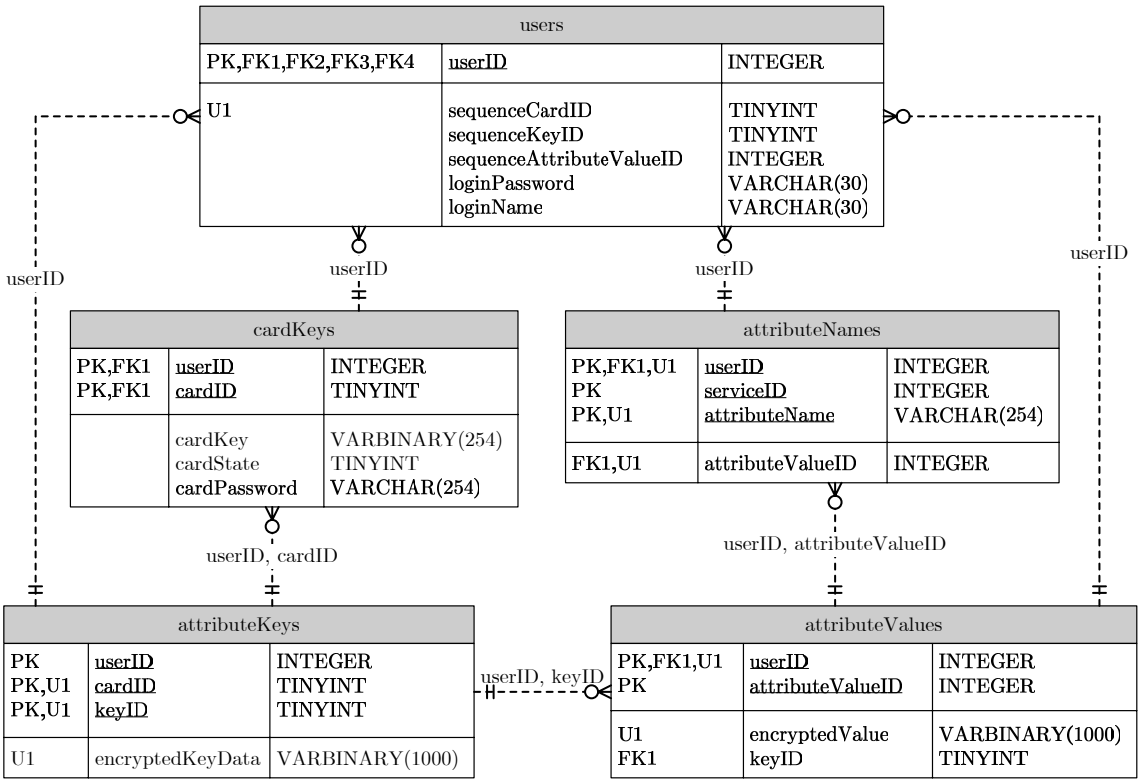

Abbildung 4.11: Tabellen der Datenbank und deren Verknüpfungen

Identifikationsnummer wird in allen Teilen (*Mobile* Java-Karte, Client und Server) des Systems intern verwendet. Der Benutzer selbst kommt nie in Kontakt mit seiner Identifikationsnummer, er kennt nur seinen *loginName*. Dieser wird zusammen mit dem *loginPassword* benötigt, wenn der Benutzer dem System eine neue Java-Karte hinzufügen will. Das Passwort liegt in der Datenbank in Form eines base64 encodierten SHA-1 Hashes vor. Die Spalten *sequenceCardID* und *sequenceKeyID* speichern die größte bereits vergebene Identifikationsnummer für neue Karten bzw. Schlüssel. Genaueres zu diesen beiden Spalten folgt in den Abschnitten 4.3.2.2 und 4.3.2.5

4.3.2.2 Die Tabelle *cardKeys*

Diese Tabelle speichert die symmetrisch signierten, öffentlichen Schlüssel der Mobile Java-Karten aller Benutzer. Das genaue Datenformat der signierten Schlüssel, die in der Spalte *cardKey* abgespeichert sind, ist in Abbildung 4.2 dargestellt. Jede Karte besitzt eine Identifikationsnummer, die *cardID*. Diese ist nur zusammen mit der *userID* eindeutig. Die höchste bisher vergebene *cardID* eines Benutzers ist in der Tabelle *users* in der Spalte *sequenceCardID* gespeichert. Der aktuelle Zustand einer Karte ist in *cardState* vermerkt. Eine Karte ist entweder im Zustand „aktiv“ oder im Zustand „gesperrt/terminiert“.

4.3.2.3 Die Tabelle *attributeNames*

Alle Benutzerdaten werden in der Datenbank in Form von Quadrupeln aus der Benutzer-ID (*userID*), einem Namen (*attributeName*), einer Dienst-ID (*serviceID*) und dem dazugehörigen verschlüsselten Wert gespeichert (*encryptedValue*). Bis auf den verschlüsselten Wert werden diese Daten in der Tabelle *attributeNames* gespeichert. Die Dienst-ID dient dazu, Kollisionen zwischen Werten von verschiedenen Diensten mit identischem Namen zu verhindern. Außerdem besteht dadurch eine einfache Möglichkeit, mehrere Identitäten und Pseudonyme zu verwalten. Jede dieser Identitäten kann beispielsweise die Attribute „Name“, „Vorname“ und „E-Mail Adresse“ umfassen.

Ein Beispiel: Der Benutzer „Hans Mustermann“ mit der Benutzer-ID 3 hat unter der Dienst-ID 0 seine realen Daten hinterlegt. Das Quadrupel (3, „Vorname“, 0, „Hans“) bezieht sich dann auf seinen realen Vornamen. Gleichzeitig hat Hans noch ein oft verwendetes Pseudonym, bei dem er unter dem Namen „Hänschen“ auftritt. Dieses Pseudonym kann er nun in der Datenbank mit der Dienst-ID 1 ablegen, ohne bei den Attributs-Namen mit seiner realen Identität zu kollidieren, denn der Eintrag (3, „Vorname“, 1, „Hänschen“) unterscheidet sich in der Service-ID.

Der verschlüsselte Wert steht in der Tabelle *attributeValues* und ist mit dieser über eine Referenz (*attributeValueID*) verknüpft. Das ermöglicht es, dieselben Benutzerdaten bei verschiedenen Diensten zu nutzen, ohne dabei mehrfach die Daten in der Datenbank abspeichern zu müssen.

4.3.2.4 Die Tabelle *attributeValues*

Die Tabelle *attributeValues* enthält die Benutzerdaten aller Benutzer in der Spalte *encryptedValue*. Die Größe der abgespeicherten Daten ist variabel. Zum Schutz der persönlichen Daten sind alle Daten mit einem zufällig generierten, symmetrischen Schlüssel (Datenschlüssel) verschlüsselt. Das System der Verschlüsselung und die Verteilung der Schlüssel ist bekannt aus Abschnitt 3.3.1. Welcher Datenschlüssel für die Verschlüsselung verwendet wurde, lässt sich über die Spalte *keyID* in der Tabelle *attributeKeys* ermitteln.

4.3.2.5 Die Tabelle *attributeKeys*

Die mit den öffentlichen Kartenschlüsseln verschlüsselten Datenschlüssel der verschiedenen Benutzer sind in der Tabelle *attributeKeys* gespeichert. Jeder Datenschlüssel besitzt eine für einen Benutzer eindeutige *keyID*, über die er identifiziert wird. Jeder Datenschlüssel ist mehrfach in dieser Tabelle vorhanden, jeweils mit einem anderen öffentlichen Kartenschlüssel des Benutzers verschlüsselt.

4.3.2.6 Entschlüsselung von Daten

Um verschlüsselte Daten aus der Datenbank zu entschlüsseln, sind mehrere Schritte notwendig. Abbildung 4.12 zeigt den vereinfachten Ablauf der direkten Entschlüsselung mit Hilfe der *Mobile* Java-Karte (Online-Zugriff). In der Implementierung werden die benötigen Daten durch einen einzigen komplexen SQL-Befehl ausgelesen. Dieser erstellt aus allen beteiligten Tabellen eine virtuelle Tabelle (View). Zum besseren Verständnis, welche Informationen aus welcher Tabelle ausgelesen werden, wird der Ablauf hier in mehreren Schritten dargestellt.

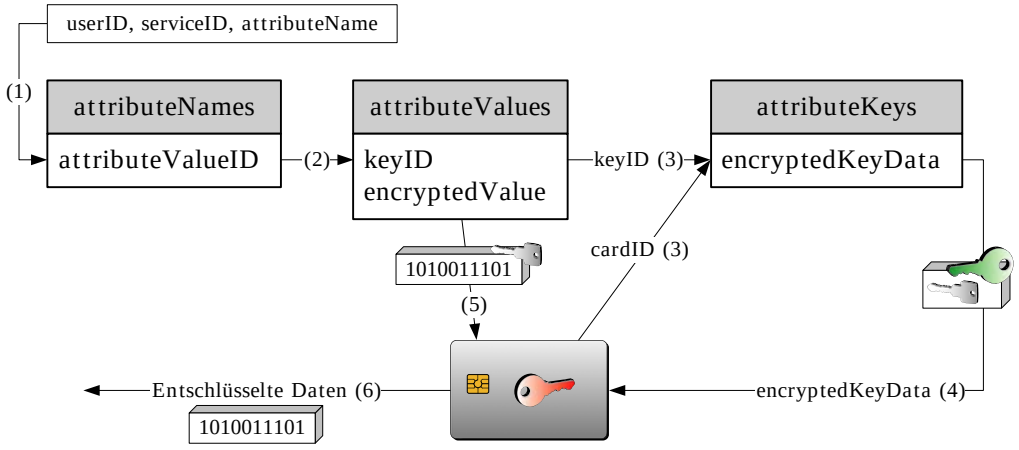

Abbildung 4.12: Entschlüsselung von Daten mit Hilfe der *Mobile* Java-Karte

In Schritt (1) wird über die Eigenschaften *userID*, *serviceID* und den *attributeName* festgelegt, welcher Daten-Block aus der Datenbank ausgelesen werden soll. Über diese Informationen wird in der Tabelle *attributeNames* die *attributeValueID* des Daten-Blocks ermittelt. Über diese *attributeValueID* kann in Schritt (2) in der Tabelle *attributeValues* die Schlüssel-Nummer *keyID* des Datenschlüssels und der mit dem Datenschlüssel verschlüsselte Daten-Block ausgelesen werden. Die *keyID* wird zusammen mit der Karten-Identifikationsmummer *cardID* in Schritt (3) verwendet, um aus der Tabelle *attributeKeys* den mit dem öffentlichen Kartenschlüssel verschlüsselten Datenschlüssel *encryptedKeyData* auszulesen. Zum Entschlüsseln des Datenschlüssels wird dieser auf die *Mobile* Java-Karte transferiert. Liegt der Datenschlüssel auf der Karte entschlüsselt vor, können die verschlüsselten Daten *encryptedValue* zur Karte übertragen werden (5). Diese entschlüsselt die Daten und gibt sie aus (6).

5. Analyse und Bewertung

Es gibt viele Möglichkeiten, ein System, wie das in den vorangegangenen Kapiteln vorgestellte, zu bewerten. Erschwerend kommt hinzu, dass es sich nicht um ein vollständiges Identitätsmanagement-System handelt, sondern nur um einen Teil eines solchen Systems. Die hier angewendeten Bewertungskriteren sind aus der Studie „Identity Management Systems (IMS): Identification and Comparison Study“ [Unab03] übernommen. Auf diese Weise ist es möglich, das vorgestellte System mit den anderen Identitätsmanagement-Systemen aus dieser Studie zu vergleichen.

Die Studie verwendet folgende Bewertungskriterien, die nachfolgend näher erläutert werden:

- Benutzbarkeit
  - Nutzen und Vereinfachungen für den Benutzer
  - Einfachheit der Benutzung
  - Verständlichkeit von kritischen Operationen
- Sicherheit
  - Vertraulichkeit
  - Integrität
  - Verfügbarkeit
- Datenschutz
  - Sensibilisierung des Benutzers
  - Transparenz für den Benutzer
  - Sparsamkeit im Umgang mit Daten
- Durchsetzung der Gesetze und Beweisbarkeit
- Vertrauenswürdigkeit
  - Mehrseitige Sicherheit
  - Zertifikate und Gütesiegel

Innerhalb der Studie wurden für jedes der genannten Bewertungskriterien positive und negative Punkte vergeben. Am Ende der Bewertung wurde für jede Bewertungskategorie die Summe der erreichten Punkte zusammengezählt. Dieses Vorgehen erleichtert die Erkennung der Stärken und Schwächen der einzelnen Systeme. Auf eine solche Punkteverteilung wird im Folgenden jedoch verzichtet, da die Implementierung nicht vollständig ist.

In den folgenden Abschnitten werden die Bewertungskriterien im Detail beschrieben und auf das vorliegende System angewendet. Bewertet wird dabei vorrangig die in Kapitel 4 vorgestellte Implementierung. Ist der zu bewertende Teil nicht oder nur ansatzweise implementiert, wird stattdessen das in Kapitel 3 vorgestellte Design bewertet. In einem solchen Fall wird aber explizit darauf hingewiesen, dass es sich um die Bewertung des Designs handelt.

5.1 Benutzbarkeit

Die Bewertung der Benutzbarkeit umfasst drei Teilbereiche. Der erste, „Nutzen und Vereinfachungen für den Benutzer“, umfasst Effekte, die durch die Benutzung des Identitätsmanagement-Systems im täglichen Gebrauch entstehen. Dabei wird insbesondere auf die Vereinfachung bestehender Abläufe und die dabei erzielbare Zeitersparnis geachtet. Die vorliegende Implementierung ist von einer täglichen Benutzung noch ein gutes Stück entfernt. Deshalb wird stattdessen das im Kapitel Design beschriebene Gesamtsystem bewertet.

Die *Mobile*-Plattform hat das Potential, die Nutzung der über sie verfügbaren Dienste stark zu vereinfachen. Insbesonders regelmäßig wiederkehrende Abläufe lassen sich in Form einer Aufgabe durch den Assistenten immer wieder ausführen. Damit verbunden ist eine enorme Zeitersparnis. Die Benutzerakzeptanz des Systems dürfte aber vor allem von Anzahl und Umfang der zur Verfügung stehenden Dienstleistungen abhängig sein und davon, welche Aufgaben sich damit durchführen lassen. Da die *Mobile*-Plattform keine eigenen Dienste anbietet, sondern diese von externen Webservices bezieht, hängt dieser Punkt eher von der Anzahl und Qualität der möglichen Aufgaben als direkt vom System ab.

Die „Einfachheit der Benutzung“ ist der zweite Teilbereich der Bewertung der Benutzbarkeit. Dabei wird bewertet, wie einfach ein System verwendet werden kann. Man spricht beispielsweise von einer „intuitiven Benutzbarkeit“, wenn die Gestaltung der Benutzungsoberfläche ähnlich funktioniert, wie der Benutzer es von anderen Programmen gewohnt ist. Auch die Gestaltung und Notwendigkeit einer Hilfe zählt zu diesem Bewertungs-Teilbereich.

Die Voraussetzungen für die Nutzung der *Mobile*-Plattform sind, verglichen mit anderen IDM-Systemen aus der Studie, auf Benutzerseite relativ hoch. Der Grund dafür ist die notwendige Hardware, die auf jedem Endgerät verfügbar sein muss. Da Chipkartenlesegeräte bisher selten zur Grundausstattung eines Computers gehören, muss ein solches nachgerüstet werden. Der Anschluss der Hardware und die Installation der Treiber im Betriebssystem ist nicht sehr schwierig. Mit Hilfe einer guten Anleitung kann dies auch von weniger erfahrenen Benutzern durchgeführt werden. Zusätzlich zur Hardware muss auch noch der *Mobile* Client installiert werden. Über ein geeignetes Installationsprogramm kann dies fast ohne Benutzereingaben erfolgen.

Der letzte Punkt bei der Bewertung der Benutzbarkeit ist die „Verständlichkeit von kritischen Operationen“. Dazu gehört die Benutzerführung, wenn ein system- oder benutzerbedingter Fehler auftritt. Außerdem wird bewertet, welche kritischen Fehler vom Benutzer gemacht werden können und wie schwerwiegend die Auswirkungen dieser Fehler sind.

Innerhalb der *Mobile*-Plattform gibt es mehrere kritische Aktionen, die ein Benutzer vermeiden sollte. Der schwerste Fehler, der von einem Benutzer begangen werden kann, ist der Verlust seiner letzten aktiven *Mobile* Java-Karte oder das Vergessen der PIN dieser Karte. Tritt dieser Fall ein, sind alle verschlüsselten Daten in der Datenbank verloren, da sie sich nicht mehr entschlüsseln lassen. Neue Karten können aber nach wie vor dem System hinzugefügt werden. Diese können dann jedoch nur auf Daten zugreifen, die nach dem Hinzufügen der neuen Karte gespeichert wurden. Aus dieser Problematik ergibt sich die Konsequenz, dass jeder Benutzer immer über mindestens zwei aktive *Mobile* Java-Karten verfügen sollte. Ist dies nicht der Fall, muss der Benutzer durch nicht zu übersehende Hinweise darauf aufmerksam gemacht werden, dass die Gefahr eines Datenverlustes mit nur einer aktiven Karte sehr hoch ist. Die bisher existierenden Fragmente einer Benutzungsoberfläche enthalten noch keinen solchen Warnhinweis.

Neben solchen Problemen des zu Grunde liegenden Designs kann es auch vorkommen, dass sich technische Probleme auf die Nutzung auswirken. Sollte beispielsweise die Netzwerkverbindung zwischen Client und Server während einer Aktualisierung der Datenschlüssel unterbrochen werden, kann es vorkommen, dass einige Datenschlüssel nicht für alle Karten eines Benutzers verschlüsselt vorliegen und somit die Liste der verschlüsselten Datenschlüssel unvollständig ist. Beim nächsten Login des Benutzers wird zwar versucht diese Liste zu vervollständigen, es können jedoch Fälle auftreten, bei denen für die Vervollständigung eine der anderen *Mobile* Java-Karten des Benutzers erforderlich ist. Der Benutzer wird in einem solchen Fall mit einem Hinweis gebeten, den Login-Vorgang mit einer anderen Karte zu wiederholen, solange wie die Schlüsselliste nicht vollständig ist.

Systemfehler können wie bei jeder Software auch bei der *Mobile*-Plattform auftreten. Dabei können die Ursachen sowohl auf eine Hardwarekomponente (Festplattenausfall, defekte Netzwerkverbindung) oder auf eine Softwarekomponente (abgestürztes Programm, Programmierfehler) zurückzuführen sein. Im derzeitigen Stand der *Mobile*-Plattform ist die Anzeige solcher Fehler eher für den Programmierer optimiert als für einen Benutzer.

Zu diesem Zeitpunkt lassen sich zum Punkt der Benutzbarkeit noch keine endgültigen Aussagen machen. Trotzdem ist erkennbar, dass schon die vorliegende Implementierung die Voraussetzungen für ein einfach zu bedienendes System erfüllt. Insbesondere die hohe Flexibilität des zu Grunde liegenden Designs und die damit verbundene Möglichkeit dieses für die verschiedensten Plattformen, vom Desktop-Computer bis zum Mobiltelefon, umzusetzen, ermöglichen die Nutzung der *Mobile*-Plattform in (fast) jeder Situation. Zudem vereinfacht die persistente Speicherung der Benutzerdaten an zentraler Stelle die Nutzung mit mehreren unterschiedlichen Endgeräten.

Steht erst einmal ein vollständig funktionsfähiger Assistent zur Verfügung, können Nutzer der *Mobile*-Plattform auf eine umfangreiche Unterstützung bei der Ausführung von wiederkehrenden Aufgaben zurückgreifen. Wenn dann noch bei der Gestaltung der endgültigen Oberfläche auf die genannten Problemstellen Rücksicht genommen wird, kann die *Mobile*-Plattform zu einem sicheren und doch einfach benutzbaren System ausgebaut werden, das seinen Nutzern einen hohen Mehrwert bietet.

5.2 Sicherheit

Der Abschnitt über die Bewertung der Sicherheit ist gegliedert in die Betrachtung der Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit.

Das Schutzziel der Vertraulichkeit beinhaltet alle Maßnahmen gegen die „unautorisierte Informationsgewinnung“ [Ecke02, S.8]. Darunter fällt das Auslesen von sensiblen Benutzerdaten, wie beispielsweise die Kreditkartennummer, durch einen externen Angreifer, einen anderen Benutzer oder den Administrator des Systems. Der Schutz von Daten vor unberechtigten Manipulationen und das Erkennen von stattgefundenen Manipulationen umfasst das Schutzziel der Integrität. Das dritte Schutzziel, die Verfügbarkeit, beinhaltet Maßnahmen gegen Überlastung und Ausfall der verwendeten Hardware und Software. Bedroht wird die Verfügbarkeit eines Systems einerseits durch externe Angreifer, die absichtlich eine Flut von Anfragen erzeugen, unter deren Last das System zusammenbricht (Denial of Service). Anderseits können Design- und Programmierfehler dafür sorgen, dass die vorhandenen Ressourcen, wie Speicher und Rechenkapazität ungleichmäßig unter den Benutzern verteilt werden.

Ein möglicher Angreifer kann versuchen, einen oder mehrere Bereiche der *Mobile*-Plattform (Java-Karte, Client oder Server) anzugreifen. Außerdem können auch die Kommunikationskanäle zwischen den Bereichen Möglichkeiten für Angriffe bieten. Für die genaue Analyse werden daher die drei Teile separat untersucht. Danach wird die Sicherheit der Kommunikationskanäle zwischen den Teilen auf mögliche Angriffsmöglichkeiten analysiert.

5.2.1 *Mobile* Server

Die Daten eines Identitätsmanagement-Systems wie der *Mobile*-Plattform bestehen zum großen Teil aus den Daten der verschiedenen Benutzer. Der Zugang zum *Mobile* Server ist durch ein passwortbasiertes Zugangskontrollsystem geschützt. Nur einem korrekt über Benutzername und zugehörigen Passwort authentifizierten Benutzer ist es gestattet, auf die webbasierte Benutzungsoberfläche und die Webservices zuzugreifen. Die Passwörter werden vom System in einer Datenbank verwaltet, in der sie in Form ihres SHA-1 Hashwertes abgespeichert sind. Nur diese Hashwerte können vom Administrator oder einem Angreifer, der sich Zugriff auf die Datenbank verschafft hat, ausgelesen werden. Für einen Angreifer sind Passwörter deshalb von Interesse, da viele Benutzer dasselbe Passwort für mehrere Dienste verwenden. Die direkte Rekonstruktion des Passwortes aus dem Hashwert ist nicht möglich, da es sich bei der Hashfunktion SHA-1 um eine Einwegfunktion handelt. Es besteht allerdings die Möglichkeit, durch sukzessives Durchprobieren (brute force) das Passwort zu erraten. Je einfacher und kürzer das Passwort ist, desto schneller kann es erraten werden. Um einen solchen Angriff zu erschweren, kann der Benutzer bei der Wahl des Passwortes unterstützt werden, beispielsweise durch die Bewertung der Komplexität des eingegebenen Passwortes.

Nach der korrekten Authentifikation eines Benutzers erlaubt der *Mobile* Server diesem den Zugriff auf seine Benutzerdaten. Ein Zugriff auf die Daten anderer Benutzer ist nicht gestattet. Bei allen Datenbankzugriffen, die in Form von SQL-Abfragen erfolgen, wurde darauf geachtet, dass diese über *PreparedStatements* erfolgen. Das verhindert zuverlässig Angriffe durch „SQL-Injection“, einer Technik bei der ein Angreifer beliebige SQL-Befehle ausführen kann [Bach04]. Selbst wenn ein Angreifer es schaffen sollte, sich Zugriff auf die Datenbank zu verschaffen, ist der Zugriff auf die Benutzerdaten noch nicht möglich, da diese verschlüsselt in der Datenbank abgespeichert sind. Die Entschlüsselung der Daten ist nur mit Hilfe einer der *Mobile* Java-Karten des Benutzers oder über ein passendes Ticket möglich.

Die Tickets werden benötigt, um dem Assistenten die Ausführung der Aufgaben zu ermöglichen. Die Tickets werden zusammen mit den Informationen der entsprechenden Aufgabe im Assistenten gespeichert, bis diese ausgeführt wird. Gelingt es

einem Angreifer, sich Zugriff auf die Tickets im Assistenten zu verschaffen und hat er Zugriff auf die Datenbank-Zugriffsschicht, kann er die Benutzerdaten auslesen, deren Datenschlüssel sich in den Tickets befinden. Hat der Angreifer es zusätzlich geschafft, sich Zugriff auf den privaten Schlüssel der Datenbank-Zugriffsschicht zu verschaffen, kann er direkt die Tickets und die darin enthaltenen Datenschlüssel entschlüsseln. Die Einschränkungen der Tickets auf die Ver- oder Entschlüsselung sowie das Gültigkeitsdatum lassen sich so umgehen. Ist der Angreifer außerdem in der Lage, die dazugehörigen verschlüsselten Daten aus der Datenbank auszulesen, kann er diese mit Hilfe der Datenschlüssel aus dem Ticket selbst entschlüsseln. Ein solcher Angriff lässt sich nicht mit den vorhandenen Schnittstellen zwischen den verschiedenen Teilen des *Mobile* Servers durchführen. Weitreichende Kontrolle über einige der Teile oder den gesamten *Mobile* Server sind dafür notwendig, sodass im Normalfall einzig ein Administrator des Systems einen solchen Angriff durchführen kann. Der gesamte Ablauf eines solchen Angriffs basiert darauf, Zugriff auf die Tickets im Assistenten zu erhalten. Deshalb darf der Assistent immer nur die Tickets enthalten, die notwendig sind („need-to-know-Prinzip“). Nicht mehr benötigte Tickets sind vom Assistenten sofort zu löschen. Gleiches gilt für die entschlüsselten Daten, die von der Datenbank-Zugriffsschicht über die Tickets ausgelesen wurden. Außerdem darf der Assistent keinen Zugriff auf die enthaltenen Tickets ermöglichen. Es bietet sich deshalb an, den Assistenten von den anderen Teilen des *Mobile* Servers zu trennen und auf einem eigenen, besonders geschützten System auszuführen.

Der Webserver-Teil des *Mobile* Servers, der die Webservice- und HTML-Schnittstelle für den *Mobile* Client bereitstellt, muss als einziger Teil aus dem Internet erreichbar sein. Solche von außen erreichbaren Dienste sind das erste Ziel externer Angreifer. Sowohl die Datenbank, als auch die Datenbank-Zugriffsschicht bieten keine Dienste an, die aus dem Internet erreichbar sein müssen. Deshalb sollte man diese, wie den Assistenten, auf einem eigenen, nicht aus dem Internet erreichbaren System ausführen. Dadurch ist der private Schlüssel der Datenbank-Zugriffsschicht besser vor Angriffen geschützt. Schwieriger zu schützen ist der private Schlüssel, der für Authentifikation des Servers beim Aufbau der Verbindung über TLS verwendet wird. Dieses Problem besteht jedoch bei allen Webservern, die TLS mit einem Serverzertifikat nutzen.

Bei Systemen, die dauerhaft mit dem Internet verbunden sind, ist es sehr schwierig, diese gegen externe Angriffe zu schützen, deren Ziel es ist, die Verfügbarkeit zu beeinträchtigen. Durch den Einsatz tausender von Zombie-Computern (Computer unter fremder Kontrolle) ist es prinzipiell möglich jeden Server zu überlasten und damit einen Ausfall zu provozieren (Distributed Denail of Service - DDOS). Der Einsatz von mehreren Servern, auf die ankommende Anfragen gleichmäßig verteilt

werden, zusammen mit einer redundanten und möglichst schnellen Internetanbindung können die Auswirkung solcher Angriffe minimieren. Die verschiedenen Dienste des *Mobile* Servers ließen sich beispielsweise auf verschiedene Maschinen verteilen, je ein Webserver für die Weboberfläche und die Webservices, ein dritter für die Datenbank. Der eingesetzte Java-Anwendungsserver JBoss ermöglicht es zusätzlich, mehrere physikalische Server zu einem logischen Server (Cluster) zu verbinden. Dadurch lassen sich beispielsweise die ankommenden Webservice-Anfragen auf mehrere Server verteilen.

Im Gegensatz zu den externen Angriffen sind im Moment keine internen Angriffe gegen die Verfügbarkeit bekannt. Theoretisch könnte man jedoch über die häufige Ausführung von sehr rechenintensiven Funktionen Teile des *Mobile* Servers überlasten. Die aufwändigste Funktion des *Mobile* Servers ist das Auslesen von Daten über ein Ticket. Dies liegt an der rechenintensiven RSA-Entschlüsselung und der nachfolgenden 3DES-Entschlüsselung. Durch das wiederholte Einlösen mehrerer Tickets könnte es möglich sein, dass ein Benutzer sehr viel Rechenzeit verbraucht und dadurch andere Benutzer warten müssen. Eine solche Situation könnte eintreten, wenn z. B. eine Aufgabe regelmäßig in sehr kurzen Abständen (wenige Sekunden) hintereinander ausgeführt wird. Durch Vorgabe einer minimalen Wiederholungszeit von Aufgaben lässt sich dieses Problem verhindern.

Zusammenfassend kann man sagen, dass alle notwendigen Schritte unternommen wurden, um die Sicherheit des *Mobile* Servers und der Benutzerdaten zu gewährleisten. Die Analyse zeigt jedoch auch, dass die erreichte Sicherheit nicht perfekt ist. Insbesondere die Tickets (bzw. die Möglichkeit Benutzerdaten ohne Kommunikation mit dem Client und dem Benutzer entschlüsseln zu können) bieten einige mögliche Angriffspunkte um unberechtigt Benutzerdaten auszulesen. Durch die vorgegebene Arbeitsweise des *Mobile* Servers kann die vorliegende Implementierung nur die Ausnutzung dieser „Problemstelle“ so schwierig wie möglich machen. Nur ein Verzicht auf den serverseitigen Assistenten könnte diese Angriffe dauerhaft verhindern, allerdings wäre dann das gesamte Konzept der *Mobile*-Plattform nicht mehr realisierbar. Die Implementierung ist deshalb ein Kompromiss zwischen der Sicherheit der Benutzerdaten auf der einen Seite und der Funktionalität des *Mobile* Servers auf der anderen Seite.

5.2.2 *Mobile* Java-Karte

Die Authentifikation des Benutzers gegenüber der *Mobile* Java-Karte erfolgt über eine maximal sechsstellige, alphanumerische PIN. Diese wird vom Benutzer in der durch den *Mobile* Client generierten Weboberfläche eingetragen und über den Client zur Java-Karte übertragen. Nach fünf Authentifikationsversuchen mit der falschen

PIN deaktiviert sich die Karte und ist danach weder nutzbar noch entsperrbar. Jede erfolgreiche Authentifikation setzt den Fehlversuchszähler wieder zurück. Wird die Java-Karte nach einer erfolgreichen Authentifikation aus dem Chipkartenlesegerät entfernt, ist nach dem Wiedereinstecken eine erneute Authentifikation notwendig. Durch die Überprüfung der PIN innerhalb der Karten-Anwendung über die *OwnerPIN*-Klasse ist es möglich den Zugriff sehr flexibel zu gestalten und sogar Funktionen anzubieten, die ohne vorherige Authentifikation funktionieren. Beispielsweise könnte man erlauben einen Kartennamen auszulesen um dem Benutzer das Erinnern an die passende PIN zu erleichtern, falls dieser für jede Karte eine eigen PIN vergeben hat. Die Flexibilität bringt aber auch einen schwerwiegenden Nachteil mit sich. Einige Chipkartenlesegeräte verfügen über ein eigenes Tastenfeld für die PIN-Eingabe. Eine darüber eingegeben PIN kann nicht durch Schadsoftware auf dem Computer belauscht werden. Diese PIN-Eingabe lässt sich jedoch nicht zusammen mit der *OwnerPIN*-Klasse nutzen. Ein eventuell vorhandenes Tastenfeld muss so ungenutzt bleiben und kann nicht zur sicheren PIN-Eingabe genutzt werden.

Die Vertraulichkeit der auf der *Mobile* Java-Karte gespeicherten Daten übernimmt, wie in Abschnitt 2.2.8 beschrieben, die JVM auf der Karte und die Anwendungs-Firewall. Als Entwickler muss man nur noch darauf achten keine Funktion zu implementieren, die es erlaubt die geheimen Daten auszulesen oder zu manipulieren. Auf der *Mobile* Java-Karte existieren einige Daten, die niemals die Karte verlassen dürfen, z. B. der private Schlüssel der Karte. Die Generierung des privaten und öffentlichen Schlüssels erfolgt in der *Mobile* Java-Karte. Zum Schutz dieses Schlüssels existiert keine Funktion, um ihn auszulesen. Ebenso wird dieser Schlüssel niemals in einen Speicherbereich kopiert, der später die Karte verlässt. Dadurch ist es auch nicht möglich, durch einen eventuellen Fehler im Programm, Teile des Schlüssels auszulesen. Die einzige bekannte Angriffsmöglichkeit auf den privaten Schlüssel ist ein Seitenkanal-Angriff (Sidechannel attack). Dabei analysiert man beispielsweise beim Entschlüsseln den Stromverbrauch in Abhängigkeit von der Zeit und versucht darüber Rückschlüsse auf den verwendeten privaten Schlüssel zu ziehen. Durch Modifikationen der Kryptoalgorithmen kann man eine solche Analyse verhindern. Ob und wie resistent die gewählte Java-Karte „JCOP 41 v2.2“ bzw. der darin enthaltenen Chip „SmartMX“ von Philips gegen Seitenkanal-Angriffe ist, konnte nicht genau in Erfahrung gebracht werden, da sowohl IBM, als auch Philips noch keine technischen Details zu diesem Kartentyp herausgeben haben. Einzig eine sehr allgemeine Werbeaussage in einem Dokument über die Chip-Familie SmartMX auf der Webseite von Philips bewirbt die „Umfangreichen Sicherheitsfunktionen gegen Seitenkanal-Angriffe“ [Phil04, S.4], weshalb man wohl davon ausgehen kann, dass Angriffe dieser Art nicht die Sicherheit der *Mobile* Java-Karte gefährden.

Ein weiterer zu schützender Bereich auf der *Mobile* Java-Karte ist der Passwortsatz des Benutzers. Dieser wird bei der Initialisierung der *Mobile* Java-Karte vom Benutzer eingegeben und auf der Karte gespeichert. Da die Vertraulichkeit dieses Passwortsatzes die Grundlage der Sicherheit des gesamten Verschlüsselungsystems bildet, wurde absichtlich keine Möglichkeit implementiert, um diesen später aus der Karte auszulesen. Ansonsten hätte ein Angreifer mit Zugriff auf eine *Mobile* Java-Karte diesen auslesen können und wäre dadurch in der Lage, beliebig viele weitere, voll funktionsfähige Karten zu erstellen. Mit diesen Karten hätte der Angreifer die Identität des Angegriffenen übernehmen können (Identitätsdiebstahl).

Der Passwortsatz wird auf der Karte verwendet, um die öffentlichen Kartenschlüssel gegen Manipulation zu sichern. Dafür wird ein Keyed-Hash Message Authentication Code (HMAC) über den gesamten öffentlichen Kartenschlüssel inklusive aller weitern Daten erstellt (siehe Abschnitt 4.1.4.1). Der Passwortsatz dient dabei als Schlüssel. Da sich der HMAC nur unter Verwendung des Schlüssels erstellen oder prüfen lässt, kann er gleichzeitig die Authentizität des Schlüssel sicherstellen. Die Prüfung des HMAC erfolgt beim Import eines neuen Kartenschlüssels. Nur nach einer erfolgreichen Überprüfung wird der Teil mit dem öffentlichen RSA-Schlüssel in den dauerhaften Speicher der *Mobile* Java-Karte übernommen. Danach gibt es keine Möglichkeit mehr den Schlüssel zu modifizieren oder gegen einen anderen auszutauschen. Es existiert jedoch die Möglichkeit, einen Schlüssel aus dieser Liste zu löschen. Dies ist notwendig, wenn der Benutzer eine Karte verloren hat oder diese durch einen technischen Defekt nicht mehr weiterverwendet werden kann.

Gleichzeitig mit dem Löschen eines öffentlichen Kartenschlüssels wird die Karten-ID des Schlüssels in die Liste der gesperrten Kartenschlüssel übernommen. Vor der Installation eines neuen öffentlichen Kartenschlüssels wird mit Hilfe dieser Liste überprüft, ob der zu installierende Kartenschlüssel zu einer als gesperrt markierten Karte gehört. Ist die Karte gesperrt, wird die Installation des Kartenschlüssels verweigert. Der Fall, dass versucht wird, den Kartenschlüssel einer gesperrten Karte zu installieren, kann nur durch einen Defekt in der Datenbank auf dem *Mobile* Server oder durch eine Manipulation eines Angreifers eintreten. Da sich solche Manipulationen nur mit Hilfe der auf jeder Karte gespeicherten Liste der gesperrten Karten erkennen lassen, muss diese Liste gegen jegliche Manipulation geschützt werden. Deshalb wurde keine Funktion implementiert, die es erlaubt, eine gesperrte Karte zu reaktivieren (entsperren), da diese Funktionalität die Möglichkeit einer Manipulation der Liste ermöglicht hätte. Jedoch könnte ein Angreifer durch Manipulation der Datenbank die Karten aller Benutzer sperren und damit die *Mobile*-Plattform lahmlegen (Angriff gegen die Verfügbarkeit).

Fasst man die Analyse der *Mobile* Java-Karte zusammen, kommt man zu dem Ergebnis, dass die *Mobile* Java-Karte selbst keine Möglichkeiten für einen Angriff bietet. Verantwortlich dafür sind zum Einen die in Abschnitt 2.2.8 beschriebenen Sicherheitsfunktionen der Java-Karte und zum Anderen die sorgsame Implementierung der *Mobile* Karten-Anwendung. Einzig die Eingabe der Karten-PIN ist wegen der genannten Gründe noch anfällig für ein Ausspähen durch Schadsoftware. Eine mögliche Lösung für dieses Problem könnte mit der Schnittstelle PC/SC Version 2 schon bald verfügbar sein. Diese wird im nächsten Kapitel in Abschnitt 6.2.2 vorgestellt.

5.2.3 *Mobile* Client

Der *Mobile* Client ist ein Programm, dass auf dem Endgerät des Benutzers installiert wird. Die Sicherheit eines solchen Programmes ist immer abhängig von der Sicherheit der Umgebung, in der es ausgeführt wird. Deshalb erfolgt die Analyse und Bewertung der Sicherheit des *Mobile* Clients in zwei unterschiedlichen Umgebungen. Die erste Umgebung ist frei von Schadsoftware und steht vollständig unter der Kontrolle des Benutzers. Angriffe können hier nur von außen erfolgen. Es wird untersucht, welchen Bedrohungen der *Mobile* Client ausgesetzt ist und welche Schutzmaßnahmen dagegen getroffen wurden. Die zweite Umgebung ist durch Schadsoftware kompromittiert und der *Mobile* Client dadurch stark gefährdet. Es wird analysiert, welche Angriffe die Schadsoftware gegen den *Mobile* Client durchführen kann und welche Gegenmaßnahmen zur Verfügung stehen, sowie deren Wirksamkeit.

Auf einem System, das vollständig unter Kontrolle des Benutzers steht, ist ein Angriff auf den *Mobile* Client für einen Außenstehenden nahezu unmöglich. Der lokale Webserver, der die webbasierte Benutzungsoberfläche zur Verfügung stellt, lässt sich nur auf dem Rechner selbst nutzen. Die Sitzungs-Verwaltung (Session-Management) des eingesetzten Webservers Jetty verwendet ein zufällig erzeugtes, nicht persistentes Cookie, das keine sicherheitsrelevanten Daten enthält. Ein Ausspähen dieses Cookies, z. B. über eine Sicherheitslücke im verwendeten Webbrowser, bringt einem Angreifer keinen Nutzen, solange dieser nicht Möglichkeit hat, sich mit dem im *Mobile* Client enthaltenen Webserver zu verbinden. Ohne eine Schadsoftware oder einen von außen nutzbaren „Proxy“ (z. B. die Secure Shell – SSH – mit Port-Forwarding) auf dem gleichen Rechner wie der *Mobile* Client ist eine solche Verbindung nicht möglich. Nach einer erfolgreichen Authentifikation speichert der *Mobile* Client einige sensible Daten in seinem Speicher, beispielsweise das Passwort für den Server-Zugriff und die authentifizierte Verbindung zur Java-Karte. Alle diese Daten sind an eine Sitzung (HTTP-Session) gebunden, die durch ein Authentifikations-Cookie identifiziert wird. Nach 10 Minuten ohne einen Zugriff durch den Benutzer wird die Sitzung automatisch beendet und alle dazugehörigen Daten im Speicher löscht.

Ist das System des Benutzers durch eine Schadsoftware kompromittiert, gibt es keine Möglichkeit, die Sicherheit der in der *Mobile*-Plattform gespeicherten Daten zu garantieren. Durch die gewählte Realisierung des *Mobile* Client als normale Software, die im Kontext des Benutzers ausgeführt wird, ist es unmöglich eine Manipulation durch Schadsoftware zu verhindern. Auch eine digitale Signierung des Java Bytecodes, die zum Schutz vor Manipulation verwendet wird, kann von einer Schadsoftware ohne Probleme entfernt werden. Da die JVM eine solche Signatur nicht vorschreibt und eine im Programm integrierte Signaturprüfung durch Manipulation unwirksam gemacht werden kann, gibt es keine Möglichkeit, das Entfernen einer Signatur zu verhindern oder zu erkennen. Durch Manipulation des *Mobile* Clients kann die Schadsoftware beispielsweise alle Benutzereingaben über die webbasierte Benutzungsoberfläche (Benutzerdaten, Passwörter oder die Karten-PIN) protokollieren und an einen Angreifer weiterleiten. Außerdem kann der *Mobile* Client so manipuliert werden, dass für den Benutzer unbemerkt die Verbindung statt zum *Mobile* Server zu einem anderen, vom Angreifer kontrollierten Server umgeleitet wird. Dadurch sind Man-in-the-Middle Angriffe möglich.

Das Ergebnis der Analyse des *Mobile* Clients ist nicht überraschend. Einer entsprechend programmierten Schadsoftware kann der *Mobile* Client nichts entgegensetzen. Immerhin muss diese Schadsoftware speziell für den *Mobile* Client entwickelt oder angepasst sein, um die vorhandenen Sicherheitsfunktionen zu umgehen. Auf einem Endgerät hingegen, das frei von Schadsoftware ist und nur von einem Benutzer verwendet wird, gibt es keine Möglichkeit für einen erfolgreichen Angriff auf den *Mobile* Client. Anders sieht es auf Mehrbenutzer-Systemen aus, wie sie unter Unix/Linux üblich sind. Durch die Möglichkeit des Port-Forwarding über SSH können andere Benutzer des Mehrbenutzer-Systems relativ einfach Zugriff auf den ungeschützten Bereich der webbasierten Benutzungsoberfläche erhalten. Ohne Kenntnis der Karten-PIN oder den Besitz eines aktuell gültigen Authentifikations-Cookies kann dieser Zugriff jedoch nicht für weitergehende Angriffe genutzt werden.

5.2.4 Kommunikationskanäle

Die Funktionalität der *Mobile*-Plattform ergibt sich aus der Zusammenarbeit von vier Teilen: Dem *Mobile* Server mit der Datenbank, dem *Mobile* Client, der *Mobile* Java-Karte und dem Webbrowser, der für die Darstellung der Benutzungsoberfläche verantwortlich ist. Für die Zusammenarbeit ist der Austausch von Daten zwischen allen Teilen notwendig. Von zentraler Bedeutung ist dabei der *Mobile* Client, der die Verbindungen zur Java-Karte und zum Server aufbaut, sowie die webbasierte Benutzungsschnittstelle anbietet. Über jede dieser Verbindungen werden Daten übertragen, deren Vertraulichkeit und Integrität gewährleistet werden muss.

Besonders gefährdet sind alle Verbindungen zwischen dem *Mobile* Client und dem *Mobile* Server, da diese das Internet durchlaufen und ohne entsprechende Sicherheitsmaßnahmen sehr leicht abgehört oder manipuliert werden können. Aus diesem Grund werden alle Verbindungen zwischen Client und Server mit Hilfe des Protokolls TLS abgesichert. Dieses sichert die Vertraulichkeit und Integrität der übertragenen Daten. Zusätzlich authentifiziert sich der *Mobile* Server gegenüber dem *Mobile* Client über ein X.509-Zertifikat, das automatisch vom *Mobile* Client überprüft wird. Man-in-the-Middle-Angriffe sind deshalb ohne eine Manipulation des *Mobile* Clients nicht möglich. Ideal wäre auch eine Authentifikation des Clients über ein X.509-Zertifikat. Wegen der fehlenden Unterstützung solcher Zertifikate durch die Java-Karten-Plattform, ist dies jedoch nicht möglich.

Ganz anders sieht der Schutzbedarf bei der Verbindung zwischen *Mobile* Client und dem Webbrowser aus. Die Verwendung des TLS-Protokolls bei dieser Verbindung ist zwar möglich, aber aus zwei Gründen nicht sinnvoll.

Erstens handelt es sich dabei um eine lokale Verbindung (über den sogenannten „LoopBack-Adapter“), die den Computer nie verlässt. Um eine solche Verbindung abzuhören oder zu manipulieren benötigt ein Angreifer Kontrolle über den Computer des Benutzers, beispielsweise über eine installierte Schadsoftware. Wie im vorhergehenden Abschnitt (5.2.3) gezeigt, kann auf einem so kompromittierten System der *Mobile* Client auf vielfältige Weise manipuliert werden, sodass jede zwischen *Mobile* Client und Webbrowser eingesetzte Verschlüsselung unwirksam gemacht werden kann.

Zweitens ist das TLS-Protokoll für lokale Verbindungen nicht geeignet. Das Problem dabei sind die X.509-Zertifikate, mit denen der „Server“ (in diesem Fall also der *Mobile* Client) sich authentifiziert. X.509-Zertifikate werden von einem Trustcenter (z. B. VeriSign[1] oder Thawte[2]) erstellt und enthalten Informationen über den Besitzer des Zertifikates und den Fully Qualified Domain Name (FQDN) des Servers. Der lokale Webserver besitzt jedoch keinen vorher bekannten FQDN, was die Erstellung eines Zertifikates durch ein Trustcenter ausschließt. Alternativ könnte ein selbstzertifiziertes Zertifikat verwendet werden. Oder es wird der „Anonyme Modus“ des TLS-Protokolls eingesetzt, bei dem der Server sich nicht über ein Zertifikat authentifiziert. Beide Möglichkeiten haben jedoch den gravierenden Nachteil, dass sie im Webbrowser des Benutzers eine Sicherheitswarnung über ein nicht vertrauenswürdiges bzw. nicht vorhandenes Zertifikat erzeugen. Diese muss der Benutzer vor dem Zugriff auf die webbasierte Benutzungsoberfläche bestätigen. Der Benutzer lernt dabei, dass Sicherheitswarnungen über nicht vertrauenswürdige Zertifikate

[1] `http://www.verisign.com`
[2] `http://www.thawte.com`

übergangen werden können bzw. in diesem Fall sogar übergangen werden müssen, ein solcher negativer Lerneffekt ist unter keine Umständen tolerierbar. Außerdem ist zu erwarten, dass zukünftige Webbrowser (z. B. der Internet Explorer 7) in den Standardeinstellungen dem Benutzer gar nicht mehr erlauben, ein ungültiges Zertifikat zu akzeptieren. Deshalb wurde darauf verzichtet, die lokale Verbindung zwischen Webbrowser und *Mobile* Client über das TLS-Protokoll abzusichern.

Damit verbleibt nur die Verbindung zwischen *Mobile* Java-Karte und dem *Mobile* Client, die noch nicht näher analysiert wurde. Die Kommunikation zwischen einer Chipkarte und der Anwendung auf dem Computer ist üblicherweise verschlüsselt, beispielsweise mit Hilfe der Funktionen aus dem GlobalPlatform-Standard. Dies ist notwendig, da die Chipkarte normalerweise Daten enthält, die der Nutzer der Karte zwar verwenden aber nicht ändern darf (z. B. der Betrag auf der Geldkarte). Das hat zur Folge, dass aus Sicht der Chipkarte (bzw. des Besitzers der Chipkarte – bei der Geldkarte die ausgebende Bank) auch der Nutzer der Karte ein möglicher Angreifer ist, und deshalb die Kommunikation auch vor ihm geschützt ablaufen muss. Ein wenig anders ist das im *Mobile*-Projekt. Da die *Mobile* Java-Karte dem Benutzer gehört und nur seine eigenen Daten bzw. Schlüssel enthält, macht ein Angriff durch den Benutzer wenig Sinn. Außerdem sind die übertragenen Daten zwischen *Mobile* Java-Karte und *Mobile* Client bereits so geschützt, dass auch das Abhören der Verbindung durch einen Angreifer diesem keinen Nutzen bringt. Einzige Ausnahme ist die Initialisierung der *Mobile* Java-Karte, bei der sensible Daten wie der Passwortsatz übertragen werden. Diese ließe sich aber auch durch eine verschlüsselte Verbindung nicht schützen, da sich eine neue Karte gegenüber dem *Mobile* Client nicht authentifizieren kann, was wiederum einen Man-in-the-Middle-Angriff zwischen Client und Java-Karte möglich macht.

Die Analyse der drei Kommunikationskanäle innerhalb der *Mobile*-Plattform zeigt, wie differenziert man die umgesetzten Maßnahmen hinsichtlich ihrer Sicherheit bewerten muss. Insbesondere die Tatsache, dass der Verzicht auf eine Verschlüsselung (bei der Verbindung zwischen *Mobile* Client und dem Webbrowser des Benutzers) sich positiv auf die Sicherheit auswirkt, ist nicht auf den ersten Blick ersichtlich. Die zweite unverschlüsselte Verbindung (zwischen der *Mobile* Java-Karte und dem *Mobile* Client) hingegen könnte man verschlüsseln. Wegen der Komplexität des GlobalPlatform-Frameworks und dem nicht vorhandenen Nutzen für die Sicherheit, wurde jedoch darauf verzichtet. Einzig die Verbindung zwischen *Mobile* Client und *Mobile* Server über das öffentliche Internet muss zwingend verschlüsselt und gegen Manipulationen geschützt erfolgen, was in der Implementierung mit Hilfe des TLS-Protokolls geschieht.

5.3 Datenschutz

Die Bewertung des Datenschutzes unterteilt sich in drei Bereiche: Erstens die „Sensibilisierung des Benutzers“, bei der bewertet wird, wie das Identitätsmanagement-System dem Benutzer hilft, mehr über das Thema Datenschutz zu erfahren. Dies kann z. B. durch einen im System vorhandenen Hilfe-Bereich geschehen oder über Hinweise und Verweise, die in die Benutzungsoberfläche integriert sind. Die aktuelle Implementierung der *Mobile*-Plattform enthält keine solche Funktion.

Der zweite Bereich bewertet die „Transparenz für den Benutzer“, die sich damit beschäftigt, ob ein Benutzer nachvollziehen kann, was innerhalb des Identitätsmanagement-Systems mit seinen Daten passiert. Bei der Entwicklung der *Mobile*-Plattform wurde besonders auf dieses Kriterium geachtet. Das Ziel der Entwicklung geht sogar noch einen Schritt weiter: Der Benutzer kann nicht nur verstehen, was mit seinen Daten gemacht wird, sondern er kann es zudem auch noch kontrollieren. Durch den Einsatz der Java-Karte erhält der Benutzer einen physischen Gegenstand, ohne den der Zugriff auf die gespeicherten Benutzerdaten unmöglich ist (Authentifikation durch Besitz). Nur der Benutzer kann mit Hilfe der Java-Karte Daten lesen oder modifizieren. Über die Java-Karte kann der Benutzer außerdem Tickets für den Assistenten erstellen und damit den Zugriff auf seine Daten delegieren. Vor der Erstellung kann dem Benutzer genau gezeigt werden, welche Daten über ein Ticket zugreifbar werden. Ein Benutzer der *Mobile*-Plattform ist deshalb immer informiert, welche seiner Daten gerade für die verschiedenen Aufgaben verwendet werden.

Der letzte Bereich des Datenschutzes bewertet die „Sparsamkeit im Umgang mit Daten“. Dazu gehört einerseits, welche Daten ein Benutzer angeben muss, um das Identitätsmanagement-System nutzen zu dürfen und andererseits, wie das IDM-System dem Benutzer ermöglicht, anonym oder unter einem Pseudonym aufzutreten. Damit ein Benutzer die *Mobile*-Plattform nutzen kann, benötigt dieser nur einen Benutzernamen und ein Passwort. Von technischer Seite ist es deshalb möglich die *Mobile*-Plattform anonym zu nutzen. Innerhalb der Plattform ist dem Benutzer überlassen, welche persönlichen Daten dieser in das System einpflegt oder welche Pseudonyme er für die verschiedenen Dienste anlegt. Außerdem unterscheidet die *Mobile*-Plattform nicht zwischen einem Profil mit persönlichen Daten und einem Profil, das Daten eines Pseudonyms enthält. Nur der Benutzer kann diese unterscheiden. Auch die Tickets ermöglichen es, mit der Freigabe von Benutzerdaten so sparsam wie möglich umzugehen, da ein Ticket immer nur den Zugriff auf die für eine Aufgabe notwendigen Benutzerdaten ermöglicht.

Das Ergebnis der Bewertung zeigt, dass bei der Entwicklung der *Mobile*-Plattform immer auf den Datenschutz geachtet wurde. Sowohl die Sparsamkeit der Plattform im Umgang mit Daten, als auch die dabei erreichte Transparenz für den Benut-

zer sind ganz im Sinne des Datenschutzes umgesetzt. Im Bereich der Transparenz übertrifft die Plattform sogar die für eine positive Bewertung notwendigen Anforderungen. Nur bei der Sensibilisierung des Benutzers kann die *Mobile*-Plattform nicht viel bieten, was jedoch hauptsächlich an der noch nicht vollständigen Umsetzung der webbasierten Benutzungsoberfläche liegt, deren vollständige Implementierung nicht Ziel dieser Arbeit war.

5.4 Durchsetzung der Gesetze und Beweisbarkeit

Zur Vereinfachung der Strafverfolgung ist es möglich, dass der Gesetzgeber die Implementierung einer staatlichen Hintertür vorschreibt. In Deutschland ist im Jahr 2005 beispielsweise mit der Telekommunikations-Überwachungsverordnung (TKÜV) eine Verordnung in Kraft getreten, die E-Mail Provider zwingt, eine spezielle Schnittstelle zu implementieren, über die beliebige Daten ausgelesen werden können. Bei dem vorgestellten System der *Mobile*-Plattform war eines der Hauptziele zu verhindern, dass andere Personen als der Benutzer Zugriff auf die gespeicherten Daten erlangen können. Eine „Abhörschnittstelle“, wie sie in der TKÜV beschrieben wird, ist nicht kompatibel mit dem Design der *Mobile*-Plattform.

Im Bereich der Beweisbarkeit geht es zum einen um die Frage: „Welcher Benutzer hat wann welche Aktion durchgeführt?“. Das Umfasst die Protokollierung aller stattgefundenen Aktionen innerhalb der Plattform und bewertet die Aussagekraft eines solchen Protokolls vor Gericht. In der *Mobile*-Plattform existiert keine Protokoll- oder Historienfunktion. Einzig die Zugriffs-Protokolle des Webservers ermöglichen es Rückschlüsse auf die verwendeten Funktionen zu ziehen. Da es sich dabei nur um einfache Textdateien handelt, haben diese jedoch vor Gericht keinerlei Beweiskraft.

Bleibt noch die Frage offen, ob und in wie weit sich eine fortgeschrittene oder qualifizierte Signatur mit der *Mobile*-Plattform nutzen lässt. Die eingesetzten Techniken der symmetrischen Signatur sind inkompatibel mit den X.509-Zertifikaten der fortgeschrittenen und qualifizierten Signatur. Eine Nutzung dieser Signaturen innerhalb des *Mobile*-Plattform ist deshalb nicht möglich.

5.5 Vertrauenswürdigkeit

Die Bewertung der Vertrauenswürdigkeit eines Identitätsmanagement-Systems erfolgt an Hand von zwei Kriterien: Der erreichten „multilateralen Sicherheit“ und den erhaltenen Zertifikaten und Gütesiegeln. Die multilaterale Sicherheit bewertet, ob die Sicherheitsbedürfnisse aller am System Beteiligten (Benutzer, Anbieter usw.) erfüllt werden. Außerdem wird analysiert, welche Vertrauensverhältnisse zwischen den Beteiligten bestehen, z. B. ob der Quellcode der verwendeten Software öffentlich

zugänglich ist (Open Source). Dadurch sind die Benutzer in der Lage zu überprüfen, ob die Software so funktioniert, wie vom Anbieter beschrieben. Ein Benutzer kann so beispielsweise überprüfen, ob die Datenbank-Zugriffsschicht die in den Tickets enthaltenen Einschränkungen durchsetzt und die Verwendung von ungültigen oder abgelaufenen Tickets verweigert.

Für eine Bewertung der *Mobile*-Plattform im Hinblick auf die erreichte multilaterale Sicherheit müssen alle Beteiligten identifiziert werden. Es bilden sich dabei drei Gruppen: Die erste umfasst die Benutzer der *Mobile*-Plattform, die zweite den Betreiber des *Mobile* Servers und in der dritten befinden sich die Anbieter, deren Dienste für die Erfüllung der durch die Benutzer gestellten Aufgaben verwendet werden. Sofern der Betreiber des *Mobile* Servers nicht gleichzeitig auch Anbieter von Diensten ist, ergibt sich eine klare Trennung in drei Vertrauensbereiche. Der Betreiber des *Mobile* Servers kann in diesem Fall als vertrauenswürdige Instanz zwischen den Nutzern und den Anbietern der Dienste vermitteln.

Bei der Entwicklung der *Mobile*-Plattform wurde darauf geachtet, keine Software-Komponenten zu verwenden, die eine spätere Veröffentlichung als Open-Source Projekt verhindern. So stehen z. B. die beiden verwendeten Webserver, auf der Clientseite „Jetty“ und auf der Serverseite „JBoss“, unter einer anerkannten Open-Source-Lizenz.

Das zweite Kriterium für die Bewertung der Vertrauenswürdigkeit sind die vom System erhaltenen Zertifikate und Gütesiegel. In Deutschland gibt es beispielsweise das Datenschutz-Gütesiegel des Unabhängigen Landeszentrum für Datenschutz Schleswig-Holstein[3], das nach einer Prüfung durch einen Gutachter vergeben wird. Da die vorliegende Implementierung der *Mobile*-Plattform noch nicht vollständig ist, hat sie im Moment weder Zertifikate noch Gütesiegel vorzuweisen.

Auch wenn die *Mobile*-Plattform im Moment noch keine offiziellen Bestätigungen über ihre Vertrauenswürdigkeit vorweisen kann, sind die Voraussetzungen, nach einer Veröffentlichung eine solche zu erhalten, sehr gut. Eine eventuelle spätere Freigabe der Quelltexte unter einer Open-Source-Lizenz dürfte sich zusätzlich positiv auf die Vertrauenswürdigkeit der *Mobile*-Plattform auswirken. Wenn dann noch der Betreiber des *Mobile*-Servers als neutraler Vermittler zwischen den Nutzern und den Anbietern auftritt, erhöht das noch einmal die Vertrauenswürdigkeit der *Mobile*-Plattform.

[3] `http://www.datenschutzzentrum.de`

5.6 Fazit

In vielen Bereichen ist die vorliegende Implementierung der *Mobile*-Plattform noch nicht so weit fortgeschritten, dass man noch eine endgültige Bewertung abgeben könnte. Die bereits implementierten Teile zeigen jedoch, dass die Stärken der *Mobile*-Plattform im Bereich des Datenschutzes, der Sicherheit sowie der Benutzbarkeit liegen.

Im Bereich des Datenschutzes ist insbesondere die Kontrolle der Benutzer über die Verwendung ihrer Daten hervorzuheben. Dies sorgt gleichzeitig für eine hohe Transparenz seitens der Nutzer, was mit ihren Daten passiert. Die gezielte Ausstellung von Tickets für den Offline-Zugriff auf bestimmte Daten sorgt außerdem für einen sehr sparsamen Umgang mit den Benutzerdaten.

Nicht ganz so positiv sieht die erreichte Sicherheit der *Mobile*-Plattform und ihrer Teile aus, wobei man die durchgeführte Analyse relativieren muss: Unter der Annahme, dass ein Angreifer einen bestimmten Computer der Plattform ganz oder teilweise unter seine Kontrolle gebracht hat, finden sich bei jedem Software-System Möglichkeiten für Angriffe. So ist beispielsweise der Grund für die verschiedenen Angriffsmöglichkeiten auf den *Mobile* Client sehr oft in der Unsicherheit der verwendeten Umgebung und damit des verwendeten Betriebssystems zu suchen.

Unter diesem Gesichtspunkt muss man zwischen zwei Alternativen abwägen. Wegen des Ziels der hohen Benutzerkontrolle konzentriert die *Mobile*-Plattform so viele Funktionen wie möglich auf der Clientseite. Auf Grund der noch sehr geringen Fähigkeiten der Java-Karte bedeutet dies, dass diese Funktionen vom *Mobile* Client übernommen werden müssen. Der *Mobile* Client ist jedoch, wie in Abschnitt 5.2.3 gezeigt, abhängig von der Sicherheit der Umgebung und damit anfällig für Schadsoftware.

Die Alternative wäre, mehr Funktionen auf den Server zu verlagern, um die Auswirkungen im Falle einer Kompromittierung des Clients so gering wie möglich zu halten. Dies hat jedoch für den Benutzer zu Folge, dass er mehr und mehr die Kontrolle über seine Daten verliert. Dies widerspricht jedoch dem Ziel, dem Benutzer mehr Kontrolle über seine Daten zu ermöglichen, weshalb auf diese Alternative nicht weiter eingegangen wird.

Das eigentliche Problem der ersten Möglichkeit und damit der *Mobile*-Plattform ist das Fehlen einer sicheren und flexiblen Umgebung für den *Mobile* Client. Lösen könnte man dieses Problem beispielsweise durch ein garantiert sicheres Betriebssystem, das nicht durch Schadsoftware infiziert werden kann. Da ein solches sicheres Betriebssystem für Endanwender nicht existiert, bleibt nur die Alternative, die schon vorhandene sichere Umgebung (die Java-Karte) als Basis zu verwenden. Diese könn-

te man so erweitern, sodass sie alle Funktionen des *Mobile* Clients übernimmt und diese damit unangreifbar für Schadsoftware macht. Im Bereich der Chipkarten gibt es tatsächlich mehrere angekündigte Produkte, die auf dieser Idee basieren, eine davon wird im nächsten Kapitel in Abschnitt 6.2.3 genauer vorgestellt.

Aber auch auf der Serverseite gibt es ein paar mögliche Angriffe, die insbesondere von Administratoren des *Mobile* Servers zum unberechtigten Auslesen von Benutzerdaten genutzt werden könnten. Verantwortlich dafür sind die Tickets, auf die jedoch in dieser Arbeit nicht verzichtet werden kann. Als Konsequenz sollte die *Mobile*-Plattform nicht zur Speicherung von Daten genutzt werden, deren Schutzbedarf nach dem IT-Grundschutzhandbuch[Bund04] mit „hoch“ oder „sehr hoch“ bewertet wird. Da Benutzerdaten normalerweise nicht einen so hohen Schutzbedarf besitzen, stellt dies keine Einschränkung der *Mobile*-Plattform dar.

Bei der Entwicklung und Implementierung der *Mobile*-Plattform wurde besonders auf einfache Benutzbarkeit geachtet, was sich in der positiven Bewertung des gesamten *Mobile*-Projektes niederschlägt. Dabei wurden jedoch drei problematische Punkte nicht erwähnt, da sie nur im weitesten Sinne zur Benutzbarkeit gezählt werden können. Diese könnten im Moment eine weite Verbreitung der *Mobile*-Plattform stark behindern.

Es bleibt die Schwierigkeit bestehen, Chipkarten mit mobilen Geräten, insbesondere in Mobiltelefonen und Handheld-Computern zu verwenden. Hier besteht immerhin die Hoffnung, das solche Geräte in Zukunft vermehrt auf dem Markt zur Verfügung stehen.

Stehen diese Geräte einmal zur Verfügung, bleibt das Problem, wie die *Mobile* Karten-Anwendung auf der SIM-Karte installiert wird. Dazu ist die Erlaubnis des Besitzers der SIM-Karte, also des Netzbetreibers, notwendig. Solange nicht der Netzbetreibers gleichzeitig der Betreiber des *Mobile* Servers ist, besteht kaum Hoffnung eine solche Erlaubnis zu erhalten.

Das dritte Problem sind die Benutzer selbst. Das Beispiel des Homebanking Computer Interface (HBCI), einem sicheren Standard für das Abwickeln von Online-Banktransaktionen, zeigt, dass vielen Kunden Bequemlichkeit wichtiger ist als Sicherheit. Bedenkt man den notwendigen Aufwand für die Nutzung der *Mobile*-Plattform, bestehend aus dem Besitz eines Chipkartenlesegerätes, ein oder mehrerer Java-Karten und der Installation einer Software auf dem Computer, kommt man zu dem Schluss, dass ein solches System kaum Chancen auf dem Markt hat. Einzig der Verzicht auf die Java-Karte könnte die *Mobile*-Plattform für mehr Benutzer attraktiv machen. Ein solcher Verzicht stand in dieser Arbeit aber nie zur Debat-

te, schliesslich ist die Java-Karte schon durch den Titel „Identitätsmanagement mit Javacards“ ein fester Bestandteil der Arbeit.

Trotz der genannten Probleme in den verschiedenen Bereichen fällt das Gesamtergebnis der *Mobile*-Plattform grundsätzlich positiv aus. Insbesondere die Integration der Java-Karte führt zu einer signifikanten Verbesserung der Sicherheit der Benutzerdaten auf dem Server, sowie der Kontrolle der Benutzer über ihre Daten. Damit ist die *Mobile*-Plattform bestens geeignet, die persönlichen Daten ihrer Benutzer vor unberechtigtem Zugriff zu schützen, dennoch ist die Plattform auch von technisch weniger versierten Menschen einfach zu benutzen. Damit erfüllt die *Mobile*-Plattform alle in dieser Arbeit gestellten Anforderungen und Ziele, inklusive der drei grundlegenden Eigenschaften eines Identitätsmanagement-Systems: „Schutz der Privatsphäre, Sicherheit und Benutzbarkeit“ [Unab03, S.v].

Zusätzlich ist es gelungen, das alte MOBILE-Projekt (siehe Abschnitt 2.3) durch den Einsatz aktueller und bewährter Techniken dem Stand der Technik anzupassen. Dies ist für die Akzeptanz der *Mobile*-Plattform seitens der möglichen Betreiber wichtig, denn diese verlassen sich lieber auf bereits bewährte Techniken. Aus diesem Grund ist der Einsatz von „Standard-Software“, wie dem J2EE-Server JBoss, positiv zu bewerten. Gut geschulte Mitarbeiter für die Installation, den Betrieb und die Wartung einer weit verbreiteten Software sind wesentlich leichter zu finden, als für Software mit einer geringen Verbreitung.

6. Ausblick

Bei der Implementierung des *Mobile*-Projektes in dieser Arbeit lag der Schwerpunkt auf der Umsetzung eines sicheren Identitätsmanagement-Systems. Auf einige im Design erwähnte oder angedachte Komponenten wurde deshalb bei der Realisierung verzichtet. Es existieren demnach noch einige Möglichkeiten für Erweiterungen des *Mobile*-Projektes. Neben dem Einsatz von bestehenden Technologien für diese Erweiterungen zeichnen sich bereits einige neue Technologien ab, die innerhalb der nächsten ein bis zwei Jahre auf den Markt kommen und dann das *Mobile*-Projekt in vielen Punkten bereichern könnten.

6.1 Erweiterungen mit bestehenden Technologien

Im Verlauf der Entwicklung des *Mobile*-Projektes und dieser Arbeit haben sich Ideen für Erweiterungen angesammelt, die sich auf Grund des begrenzen Umfangs der Arbeit nicht umsetzen ließen. In den folgenden Abschnitten werden die Erweiterungen vorgestellt, deren Umsetzung sich mit den aktuell verfügbaren Technologien durchführen lässt.

6.1.1 Java Web Start

Die Implementierung, die in dieser Arbeit vorgestellt wird, verlangt vom Benutzer neben dem Anschließen und Einrichten des notwendigen Chipkartenlesegrätes auch die Installation einer Software (*Mobile* Client) auf dem Endgerät. Zusätzlich muss auch noch eine passende Java-Laufzeitumgebung installiert sein, was beispielsweise bei allen Windows-Betriebssystemen standardmäßig nicht der Fall ist. Sowohl beim Anschließen der Hardware als auch bei der Installation der Java-Laufzeitumgebung sind keine Vereinfachungen für den Benutzer möglich, wohl aber bei der Installation

des *Mobile* Clients. Mit einigen Anpassungen kann dieser sehr einfach und komfortabel über die Technologie „Java Web Start“ installiert werden. Der Benutzer muss für die Installation nur eine bestimmte Adresse in seinem Webbrowser eingeben oder einen Link auf einer Webseite anklicken. Die Installation erfolgt danach automatisch. Dieses Verfahren hat zudem den Vorteil, dass Web-Start-Anwendungen bei jedem Ausführen automatisch überprüfen, ob eine neuere Version existiert. Ein eventuell notwendiges Update des *Mobile* Clients kann so ohne Benutzerinteraktion erfolgen.

6.1.2 Geräteprofile

Zu den angedachten, aber nicht umgesetzten Funktion der *Mobile*-Plattform gehört die Anpassung der webbasierten Benutzungsoberfläche an die verschiedenen Gerätetypen. Problematisch ist dabei die Erkennung der verschiedenen Geräte und ihrer Fähigkeiten. Die einfachste Möglichkeit ist die Auswertung der „Browser ID“ (auch als „User-Agent“ bekannt), die Namen und Version des verwendeten Webbrowsers enthält. Diese wird bei jedem Aufruf im HTTP-Header an den Server übermittelt. Für bekannte Webbrowser lassen sich so Rückschlüsse auf die verwendete Geräte-Plattform ziehen, die wiederum im gewissen Umfang etwas über die Gerätefähigkeiten aussagt. Diese Informationen sind jedoch nicht sehr genau und benötigen zudem eine regelmäßig um neue Geräte ergänzte Datenbank, um vom Webbrowser auf das mobile Gerät schließen zu können.

Wesentlich genauere Informationen lassen sich beispielsweise über das „Composite Capability/Preference Profiles“ (CC/PP)[1] übertragen. Dieser vom World Wide Web Consortium (W3C) im Jahr 2004 verabschiedete Standard erlaubt die maschinenlesbare Spezifikation von Gerätefähigkeiten wie die Art der Tastatur, die Bildschirmauflösung, die maximal darstellbaren Farben und viele weitere Eigenschaften. Für sehr viele Mobiltelefone bieten die Hersteller auf ihren Webseiten bereits vollständige Profile an[2]. Diese lassen sich dann ähnlich der „Browser ID“ einfach im HTTP-Header referenzieren. Mit Hilfe dieser Informationen über die Gerätefähigkeiten könnte dann der *Mobile* Server unterschiedliche, an die Endgeräte angepasste, Versionen der webbasierten Benutzungsoberfläche ausliefern.

6.1.3 Historien-Funktion

Eine weitere mögliche Erweiterung des *Mobile*-Projektes bezieht sich auf das darin enthaltene Identitätsmanagement-System. Mit relativ geringem Aufwand lässt sich die vorgestellte Implementierung um eine Historen-Funktion erweitern, die sowohl das Einlösen der Tickets, als auch den direkten Zugriff über die *Mobile* Java-Karte

[1] `http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115`
[2] UAProf profile repository: `http://w3development.de/rdf/uaprof_repository`

protokollieren könnte. Durch die noch nicht genau festgelegte Arbeitsweise des Assistenten macht die Implementierung beim aktuellen Stand des *Mobile*-Projektes jedoch noch keinen Sinn.

6.1.4 Liberty Alliance Project

Das *Mobile*-Projekt, wie es in dieser Arbeit beschrieben wird, bietet durch die sichere Speicherung der Benutzerdaten und die Bearbeitung von Aufgaben schon einen hohen Nutzwert für die Benutzer. Neben diesen Einsatzmöglichkeiten bietet es sich an, die *Mobile*-Plattform um Funktionen aus dem Liberty Alliance Project zu erweitern. Beispielsweise ist es ohne große Probleme möglich, den *Mobile* Server zu einem Authentifikationsdienst (identity provider) zu erweitern. Dadurch lässt sich die *Mobile*-Plattform vom Benutzer zusätzlich als Single Sign-On-System für normale Webseite nutzen. Durch die serverseitige Speicherung der Benutzerdaten bietet sich der *Mobile* Server auch an, um den Dienst des „attribute provider“ bereitzustellen. Ob sich jedoch dieser Dienst mit dem Zugriff über Tickets vereinbaren lässt, müsste noch genauer geprüft werden.

6.2 Neue Technologien

Die Entwicklung im Bereich der Informations-Technologie erfolgt mit einer sehr hohen Geschwindigkeit. Täglich werden neue oder weiterentwickelte Technologien und Produkte veröffentlicht oder angekündigt. Diese ersetzen ihre Vorgänger oder verdrängen konkurrierende Systeme. Entwicklungen, wie das *Mobile*-Projekt, unterliegen deshalb einem ständigen Wandel, um sich an diese Veränderungen anzupassen. Während der Planung und Umsetzung dieser Arbeit wurden mehrfach neue Produktversionen vorgestellt, die bereits in der finalen Version der Implementierung integriert sind, sofern der Stand der Arbeit es erlaubte. Daneben gab es eine Reihe sehr interessanter Ankündigungen, die in Zukunft die Weiterentwicklung des *Mobile*-Projektes beeinflussen könnten. Die wichtigsten Ankündigungen aus dem Bereich der Chipkarten und der Identitätsmanagement-Systeme werden in den folgenden Abschnitten genauer vorgestellt.

6.2.1 Java Smart Card I/O API

Die Erfahrungen dieser Arbeit zeigen, dass eine der größten Hürden beim Einsatz von Java-Karten auf den verschiedenen Java-Plattformen und Betriebssystemen die fehlende standardisierte Schnittstelle für die Kommunikation mit der Java-Karte ist. Die Schnittstelle SATSA auf der J2ME-Plattform ist ein Anfang, sie krankt jedoch im Moment noch an der mangelnden Unterstützung seitens der Mobiltelefon- und PDA-Hersteller. Bisher existiert nur eine einzige Ankündigung für ein Mobiltelefon,

das diese Schnittstelle unterstützt (Nokia N91). Eine solche einheitliche Chipkarten-Schnittstelle wäre auch wünschenswert für die Plattform J2SE. Dies hat auch Sun Microsystems erkannt und für die kommende Java 2 Standard Edition Version 6.0 (Codename „Mustang“) mit der „Smart Card I/O API“[3] eine solche Schnittstelle angekündigt. Die Veröffentlichung dieser Version ist geplant für Mitte 2006[4]. Die Entwicklung von Java-Anwendungen, die Chipkarten nutzen, wird durch diese betriebssystemunabhängige Schnittstelle zukünftig wesentlich vereinfacht.

6.2.2 PC/SC Version 2.0

Aber auch die „Smart Card I/O API“ greift nicht direkt auf das Chipkartenlesegerätes zu, sondern nutzt dafür die PC/SC Schnittstelle des Betriebssystems. Es handelt sich demnach nur um eine Abstraktionsschicht (Layer). Leider fehlen der aktuell verwendeten PC/SC-Schnittstelle in Version 1.x einige wichtige Funktionen. Sie bietet beispielsweise keine Möglichkeit, eine eventuell vorhandene Anzeige oder ein Tastenfeld des Chipkartenlesegerätes zu nutzen. Deshalb hat die PC/SCWorkgroup[5] seit 1999 an der Nachfolgeversion 2.0 gearbeitet. Diese wurde Mitte 2005 veröffentlicht[6] und bietet unter anderem endlich die Möglichkeit das Tastenfeld und die Anzeige des Chipkartenlesegerätes zu nutzen[7]. Bis jedoch PC/SC 2.0 von den beteiligten Betriebssystemen und den Treibern der Chipkartenlesegeräte unterstützt wird, können noch mehrere Jahre vergehen.

6.2.3 Internet Smart Card

Neben neuen Technologien im Softwarebereich gibt es auch einige sehr interessante Weiterentwicklungen im Bereich der Chipkarten, die sehr gut in das *Mobile*-Projekt passen. Beispielsweise haben verschiedene Chipkartenhersteller spezielle „Internet-Smartcards“ angekündigt. Es handelt sich dabei um Chipkarten, die über einen integrierten Webserver verfügen. Ein Beispiel für diese Chipkarten ist die „Internet Smart Card“ von Giesecke & Devrient. Ihr integrierter Webserver kann direkt als Netzwerkgerät über das TCP/IP-Protokoll angesprochen werden. Außerdem kann die Internet Smart Card über eine TLS-verschlüsselte Verbindung Daten mit beliebigen Servern im Internet austauschen. Mit Hilfe einer solchen Chipkarte wäre es möglich, den *Mobile* Client teilweise in die Chipkarte zu verlagern. Dadurch können die sicherheitskritischen Aktionen komplett innerhalb der Chipkarte ausgeführt werden, was eine Manipulation durch eine Schadsoftware unmöglich macht.

[3] JSR268: `http://jcp.org/en/jsr/detail?id=268`
[4] `http://de.sun.com/company/events/2005/isv/pdf/java_roadmap.pdf`
[5] `http://www.pcscworkgroup.com`
[6] `http://www.pcscworkgroup.com/specifications/specdownload.php`
[7] `http://www.pcscworkgroup.com/specifications/files/pcsc10_v2.01.5.pdf`

6.2.4 Microsoft InfoCard

Mit der Entwicklung eines neuen und gegenüber dem Passport Dienst verbesserten Identitätsmanagement-Systems beschäftigt sich im Moment die Firma Microsoft. Deren Identitätsmanagement-System namens „InfoCard“ wird voraussichtlich 2006 zusammen mit Windows Vista veröffentlicht werden. InfoCard ist ein vollständig neues System, das im Gegensatz zu Passport mehrere Identitäten verwalten kann. Jede Identität entspricht dabei einer sogenannten „InfoCard“. Eine InfoCard ist ein Authentifikations-Token, das der Benutzer entweder von einem Authentifikations-Dienst („identity provider“) ausgestellt bekommt oder sich selbst erstellen kann. Greift ein Benutzer beispielsweise auf einen Dienst zu, der eine Authentifikation erfordert, meldet der Dienst dies dem InfoCard-System auf die Computer des Benutzers. Der Benutzer kann daraufhin aus einer Liste seiner passenden InfoCards eine auswählen, mit deren Identität er den Dienst nutzen will. Nach der Auswahl der Identität fordert das InfoCard-System des Benutzers vom Authentifikationsdienst ein Sicherheitstoken („security token“) an. Mit Hilfe dieses Sicherheitstokens kann der Benutzer nun den Dienst nutzen [Micr05a, S.6f].

Alle Spezifikationen von InfoCard sollen ohne eine Nutzungsgebühr veröffentlicht werden. Dies ermöglicht die InfoCard Implementierung auch für nicht von Microsoft unterstützte Betriebssysteme. InfoCard soll als „Identitätsmanagement-Metasystem“ arbeiten und dabei sogar fähig sein, Systeme aus dem Liberty Alliance Projekt zu integrieren. Das grundlegende Konzept von InfoCard ist jedoch wesentlich einfacher aufgebaut als beim LAP. InfoCard sieht z. B. keine vom Benutzer anlegbaren Verknüpfungen zwischen Identitäten von verschiedenen Anbietern. Nur Dienstanbieter können sich durch ihre Sicherheitsrichtlinien einseitig oder gegenseitig vertrauen.

InfoCard zeigt, dass Microsoft aus dem Scheitern des Passport-Dienstes gelernt hat. Der Benutzer von InfoCard hat weitreichende Kontrolle über seine Informationen und darüber, wem er sie preisgibt. Durch die geringere Komplexität gegenüber dem Liberty Alliance Project ist InfoCard zudem für die Benutzer wesentlich einfacher zu Verstehen, was sich sicherlich positiv auf die Benutzerakzeptanz auswirken wird. Mit der Offenlegung der zu Grunde liegenden Spezifikationen könnte InfoCard auch außerhalb der Microsoft-Welt Anhänger finden, allerdings steht die genaue Lizenz für eine Veröffentlichung noch nicht fest. Es bleibt also abzuwarten, wie offen InfoCard letztendlich sein wird und ob die Benutzer das System akzeptieren werden.

Literaturverzeichnis

[7816-4] ISO7816-4: Identification cards - Integrated circuit cards Part 4: Organization, security and commands for interchange, 2004.

[Bach04] Daniel Bachfeld. Giftspritze. `http://www.heise.de/security/artikel/43175/1`, Jan 2004. [Online, zuletzt zugegriffen 2006-01-10].

[BDSG] Bundesdatenschutzgesetz 2003. `http://www.bfd.bund.de/information/BDSG.pdf`.

[Blei05] Holger Bleich. US-Bank vermisst Backup-Bänder mit 1,2 Millionen Kreditkarten-Datensätzen. `http://www.heise.de/newsticker/meldung/56836`, Feb 2005. [Online, zuletzt zugegriffen 2006-01-10].

[Bund04] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutzhandbuch*, 2004. `http://www.bsi.de/gshb/deutsch/download/GSHB2004.pdf`.

[Cap02] Prof. Dr. Clemens H. Cap. Smart Cards Overview and Introduction. `http://wwwiuk.informatik.uni-rostock.de/sites/lehre/lehrveranstaltungen/javacard-zrh/v01_smartx_intro_1_2_v1.pdf`, Apr 2002. [Online, zuletzt zugegriffen 2006-01-10].

[Chan04] Marc Chanliau. Selbstbehauptung - Web Services-Sicherheit und die SAML. *XML & Web Services Magazin* (1), 2004.

[DGTT98] Deutsche Telekom AG - Telesec, GMD - Forschungszentrum Informationstechnik GmbH, TÜV Informationstechnik GmbH und TELETRUST Deutschland e.V. CT-API Version 1.1. `https://www.secure.trusted-site.de/Download/CTAPI/CTAPI11.pdf`, Okt 1998.

[Dhar04] Sumit Dhar. Introduction to Smart Cards. `http://sumitdhar.blogspot.com/2004/11/introduction-to-smart-cards.html`, Nov 2004. [Online, zuletzt zugegriffen 2006-01-10].

[Ecke02] Prof. Dr. Claudia Eckert. *IT-Sicherheit*. Oldenbourg. 2002.

[GHHM+04] Piklu Gupta, Mario Hoffmann, Bernhard Holtkamp, Wiebke Möhr, Jan Peters, Matthias Ritscher und Agnès Voisard. Mobile kontextabhängige Multimediadienste. *Informatik Spektrum* 27(1), 2004, S. 35–43.

[HaKR04] Marit Hansen, Henry Krasemann und Martin Rost. Persönlichkeitsspaltung. *c't Magazin für Computertechnik* (9), 2004, S. 164–167.

[HaNS02] Uwe Hansmann, Martin S. Nicklous und Thomas Schäck. *Smart Card Application Development Using Java*. Springer, Heidelberg. 2002.

[Hoff04] Dipl.-Inform. Mario Hoffmann (Hrsg.). M-Services - Sichere Dienste für mobile Bürger (MOBILE). Schlussbericht, Fraunhofer Institute SIT, IPSI und IGD, Sept 2004.

[Kuri04] Jürgen Kuri. AOL-Mitarbeiter wegen Verkaufs von Kundendaten verhaftet. `http://www.heise.de/newsticker/meldung/48542`, Jun 2004. [Online, zuletzt zugegriffen 2006-01-10].

[Lero01] Xavier Leroy. On-Card Bytecode Verification for Java Card. *Lecture Notes in Computer Science* Band 2140, 2001, S. 150+.

[Micr05a] Microsoft Corporation. A Guide to Integrating with InfoCard v1.0. `http://download.microsoft.com/download/6/c/3/6c3c2ba2-e5f0-4fe3-be7f-c5dcb86af6de/infocard-guide-beta2-published.pdf`, Aug 2005. [Online, zuletzt zugegriffen 2006-01-10].

[Micr05b] Microsoft Corporation. Microsoft's Vision for an Identity Metasystem. `http://www.identityblog.com/stories/2005/07/05/IdentityMetasystem.htm`, Mai 2005. [Online, zuletzt zugegriffen 2006-01-10].

[More76] Roland Moreno. Systems for storing and transferring data. `http://www.freepatentsonline.com/4092524.html`, Mai 1976. [Online, zuletzt zugegriffen 2006-01-10].

[Ort01] Ed Ort. Writing a Java Card Applet. `http://developers.sun.com/techtopics/mobility/javacard/articles/intro/`, Jan 2001. [Online, zuletzt zugegriffen 2006-01-10].

[Phil04] Philips. *SmartMX platform features*, Mär 2004. `http://www.semiconductors.philips.com/acrobat_download/other/identification/095710.pdf`.

[RaEf02] Wolfgang Rankl und Wolfgang Effing. *Handbuch der Chipkarten*. Hanser Fachbuchverlag. 2002.

[Sun 01] Sun Microsystems. *Java Card™ Platform Security*, Okt 2001. `http://java.sun.com/products/javacard/JavaCardSecurityWhitePaper.pdf`.

[Sun 03] Sun Microsystems. *Java Card™ Platform – Virtual Machine Specification, Version 2.2.1*, Okt 2003.

[Sun 05] Sun Microsystems. Introduction to the Liberty Alliance Project. `http://docs.sun.com/source/817-7648/intro.html`, 2005. [Online, zuletzt zugegriffen 2006-01-10].

[Unab] Unabhängige Landeszentrum für Datenschutz Schleswig-Holstein. Was ist Identitätsmanagement? `http://www.datenschutzzentrum.de/projekte/idmanage/was.htm`. [Online, zuletzt zugegriffen 2006-01-10].

[Unab03] Unabhängige Landeszentrum für Datenschutz Schleswig-Holstein. *Identity Management Systems (IMS): Identification and Comparison Study*, Sept 2003. `http://www.datenschutzzentrum.de/idmanage/study/ICPP_SNG_IMS-Study.pdf`.

[Welt03] Die Welt. Neue Sicherheitspanne bei Microsoft. `http://www.welt.de/data/2003/05/12/91548.html`, Mai 2003. [Online, zugegriffen 2005-08-21].

[Wiki05a] Wikipedia. Chipkarte. `http://de.wikipedia.org/wiki/Chipkarte`, 2005. [Online, zugegriffen 2005-07-13].

[Wiki05b] Wikipedia. Liberty Alliance Project. `http://de.wikipedia.org/wiki/Liberty_Alliance_Project`, 2005. [Online, zugegriffen 2005-07-10].

[Wiki05c] Wikipedia. Microsoft Passport-Netzwerk. `http://de.wikipedia.org/wiki/Microsoft_Passport-Netzwerk`, 2005. [Online, zugegriffen 2005-07-09].

[Wiki05d] Wikipedia. Smartcard. `http://en.wikipedia.org/wiki/Smartcard`, 2005. [Online, zugegriffen 2005-07-13].

[Zieg05] Peter-Michael Ziegler. Erneut millionenfacher Diebstahl von Kreditkarten-Daten. `http://www.heise.de/newsticker/meldung/58733`, Apr 2005. [Online, zuletzt zugegriffen 2006-01-10].

Glossar

Triple DES (3DES)

Auch bekannt unter dem Namen „DESede“. 3DES ist eine Erweiterung des DES-Verschlüsselungsalgorithmus, der Daten über die dreimalige Verwendung des DES-Algorithmus mit jeweils unterschiedlichen Schlüsseln verschlüsselt. Dadurch kommt man auf eine effektive Schlüssellänge von 112 Bit.

Advanced Encryption Standard (AES)

Symmetrischer Verschlüsselungsalgorithmus, auch als Rijndael-Algorithmus bekannt. Bei gleicher Schlüssellänge ist dieser Algorithmus sicherer als der verwandte Algorithmus 3DES.

Application Protocol Data Unit (APDU)

Ein 255 Byte großer Datenblock, der zwischen der Host-Anwendung und der Karten-Anwendung ausgetauscht wird.

Application Programming Interface (API)

Beschreibung einer Schnittstelle, die Informationen über Funktionen und deren Aufrufparameter enthält, die von einem Anwendungsprogramm verwendet werden können.

Applet

Ein in eine HTML-Seite eingebettetes Java-Programm mit eigener Benutzungsoberfläche. Applets laufen in einer sicherheitstechnisch beschränkten Laufzeitumgebung (Sandbox), die den Zugriff auf lokale Ressourcen verhindert.

Compact Hypertext Markup Language (CHTML)

Eine für mobile Endgeräte vereinfachte Variante der Beschreibungssprache HTML. Nicht enthalten ist unter anderem die Unterstützung von JPEG-Bildern, Tabellen, verschiedene Schriftarten und die Möglichkeit Frames und Stylesheets zu nutzen.

Card Operating System (COS)

Abkürzung für das Betriebssystem einer Chipkarte.

Cascading Style Sheets (CSS)

Eine Formatierungssprache, mit der sich das Aussehen und die Formatierung von HTML- oder XML-Dokumenten festlegen lassen.

Card Terminal Application Programming Interface (CT-API)

Sehr hardwarenahe Schnittstelle um Speicher- oder Prozessor-Chipkarten sowie Chipkartenlesegeräte anzusprechen.

Domain Name System (DNS)

Ein verteilter Netzwerk-Dienst, der die Zuordnung von Internet-Adressen in Form eines FQDN zu der entsprechenden IP-Adresse verwaltet.

Enterprise JavaBean (EJB)

Eine Komponenten-Architektur für die J2EE-Plattform. EJBs vereinfachen die Entwicklung von modularen Web-Anwendungen.

Fully Qualified Domain Name (FQDN)

Der vollständige DNS-Name eines Computers. Beispiel: www.informatik.tu-darmstadt.de

Keyed-Hash Message Authentication Code (HMAC)

Eine spezielle parametrisierte Hashfunktion, die in RFC 2104 beschrieben wird. Die intern verwendete Hashfunktion ist beliebig austauschbar (z. B. HMAC-SHA1 oder HMAC-MD5).

Host-Anwendung

Ein Programm, das außerhalb der Chipkarte ausgeführt wird und dabei auf Funktionen der Chipkarte zurückgreift.

Hypertext Transfer Protocol (HTTP)

Ein vom W3C standardisiertes Protokoll der Anwendungsebene. Es bildet die Grundlage für den Zugriff auf das World Wide Web (WWW).

Internet Information Server (IIS)

Ein von Microsoft entwickelter Webserver, der bei allen Windows-Betriebssystemen der Server-Reihe enthalten ist.

Java 2 Enterprise Edition (J2EE)

Schnittstellendefinition für die Entwicklung von Web-basierte Anwendungen im Unternehmens-Umfeld. J2EE basiert auf J2SE.

Java 2 Micro Edition (J2ME)

Spezifikation für Java-Umgebungen auf Endgeräten wie Mobiltelefonen, PDAs und Set-Top-Boxen. J2ME umfasst mehrere Profile mit unterschiedlichem

Funktionsumfang und optionale Erweiterungen um sich besser an die verschiedenen Geräte anpassen zu können.

Java 2 Standard Edition (J2SE)

Schnittstellendefinition für die normale Desktop-Edition von Java. Es existieren mehrere Versionen der Schnittstellendefinition, aktuell ist die Version 5.0, aber auch die Vorgängerversion 1.4 wird sehr häufig noch eingesetzt. Sun bietet die J2SE für verschiedene Betriebssysteme als Download an. Das Paket umfasst die Java Laufzeitumgebung und die Implementierung der Schnittstellendefinition, die Klassenbibliothek.

JBoss

Ein J2EE-konformer Anwendungs-Server, der unter der GNU Lesser General Public License (LGPL) steht.

Java Card Remote Method Invocation (JC-RMI)

Erlaubt es, aus der Host-Anwendung transparent Methoden auf der Java-Karte aufzurufen. Das Verfahren ist eine Variante von RMI, jedoch inkompatibel dazu.

Jetty

Ein besonders kleiner J2EE-konformer Webserver mit Unterstützung für Java Server pages (JSP) und Servlets. Er steht unter der Apache 2.0 License.

Java Server Pages (JSP)

Eine Technologie für die Erzeugung von dynamischen Webseiten mit Hilfe von Java. JSP-Seiten enthalten sowohl gewöhnlichen HTML-Code, als auch Java-Code. Vor dem ausliefern einer Webseite führt der Webserver den enthaltenen Java-Code aus, der die HTML-Seite um dynamisch erzeugten HTML-Code ergänzt.

Java Virtual Machine (JVM)

Die Laufzeitumgebung, in der jeweils ein Java-Programm ausgeführt wird.

GNU Lesser General Public License (LGPL)

Open Source-Lizenz der Free Software Foundation. Programme und Bibliotheken, die unter der LGPL stehen müssen immer mit ihrem Quellcode vertrieben werden, sie dürfen jedoch auch in kommerziellen Projekten verwendet werden, deren Quellcode nicht veröffentlicht wird.

Multi Application Card Operating Systems (MACOS)

Abkürzung für das Betriebssystem einer Chipkarte, das es erlaubt, mehrere Anwendungen auf einer Karte zu installieren.

Network Address Translation (NAT)

Ein Verfahren, bei dem die IP-Adressen in IP-Paketen ersetzt werden. NAT wird häufig eingesetzt in Netzwerk-Routern um ein privates Netzwerk an ein

öffentliches Netzwerk (z. B. Internet) anzubinden. Aus dem öffentlichen Netz ist nur der Router zugreifbar, weshalb NAT-Router oft auch als Firewall bezeichnet werden.

Personal Identification Number (PIN)

Bezeichnet eine mehrstellige Zahl, mit deren Hilfe sich ein Benutzer als rechtmäßiger Eigentümer ausweisen kann, beispielsweise gegenüber einer Chipkarte. Mittlerweile darf eine PIN auch oftmals Buchstaben enthalten und ist damit zu einem Synonym für ein Passwort geworden.

Public Key Infrastruktur (PKI)

Infrastruktur, mit der kryptographische Schlüssel erstellt, verwaltet und verteilt werden können.

Remote Method Invocation (RMI)

Eine Technik, um transparent Methoden von Java-Objekten auf entfernten Rechner aufzurufen.

RSA

Asymmetrischer Verschlüsselungsalgorithmus, benannt nach den drei Erfindern Rivest, Shamir und Adleman.

Secure Assertion Markup Language (SAML)

„SAML ist eine auf XML basierende Beschreibungssprache, mit der Web Services auf einfache Art und Weise authentifizierungs- und autorisierungsbezogene Informationen austauschen können.“ [Chan04]

Security and Trust Services API for J2ME (SATSA)

Auch bekannt unter der Abkürzung JSR177. Eine optionale Erweiterung für J2ME. Enthält Funktionen aus dem Bereich der Kryptographie für die Verschlüsselung und Erzeugung von digitalen Signaturen. Außerdem ermöglicht es den Zugriff auf Chipkartenlesegeräte über APDU oder JC-RMI.

Secure Mobile Agent (SeMoA)

Eine Plattform für sichere mobile Agenten, entwickelt am Fraunhofer IGD.

Servlet

Eine Technologie für die Erzeugung von dynamischen Webseiten mit Hilfe von Java. Servlets sind Java-Klassen, die, wenn sie ausgeführt werden, üblicherweise HTML-Code erzeugen. Für die Implementierung von Webservices gibt es auch Servlets, die XML-Code erzeugen.

Simple Object Access Protocol (SOAP)

Ein vom W3C standardisiertes Protokoll für die plattformübergreifende Kommunikation mehrerer Programme. SOAP-Nachrichten sind XML-Dateien mit einem bestimmten XML-Schema. Mit SOAP lässt sich ein entfernter Methodenaufruf inklusive aller Parameter in Form von typisierten Daten kodieren.

Für die Übertragung von SOAP-Nachrichten kann jedes beliebige Protokoll verwendet werden. Bei Webservices erfolgt z. B. der Transport über das Hyper Text Transfer Protocol (HTTP).

Single Sign-On (SSO)

Ein Authentifikations-System bezeichnet man als Single Sign-On System, wenn der Benutzer sich nur einmal authentifizieren muss und danach verschiedene Dienste nutzen kann.

Transport Layer Security (TLS)

Nachfolger des von Netscape entwickelten Protokolls „Secure Sockets Layer"(SSL) zum sicheren Übertragen von Daten. Verbindungen über TLS werden verschlüsselt und die Integrität der übertragenen Daten wird sichergestellt. Zudem ermöglicht TLS eine einseitige oder beidseitige Authentifikation der Kommunikationspartner über X.509-Zertifikate.

World Wide Web Consortium (W3C)

Internationales Konsortium, das sich um die Neu- und Weiterentwicklung der verschiedenen Internet-Protokolle und Datenformate kümmert.

Webservice

Ein Webservice bietet Funktionen oder Dienste an, die von entfernt laufenden Anwendungen über ein Netzwerk genutzt werden können. Die Kommunikation zwischen Anwendung und Webservice erfolgt über das Simple Object Access Protocol (SOAP).

Wireless Markup Language (WML)

Eine XML-basierte Webseiten-Beschreibungssprache für Mobiltelefone, die ausschliesslich in Verbindung mit dem „Wireless Application Protokoll"(WAP) genutzt wird.

Webservices for J2EE (WS4EE)

Java-Standard für die Implementierung von Webservices auf J2EE-Servern.

Web Service Definition Language (WSDL)

Eine auf XML basierende Beschreibungssprache, die von Webservices genutzt wird um die angebotenen Funktionen, sowie die zum Aufruf nötigen Parametertypen zu spezifizieren.

Extensible Hypertext Markup Language (XHTML)

Eine XML-basierte Webseiten-Beschreibungssprache, die nur die logische Struktur einer Webseite definiert. Die Gestaltung erfolgt über Cascading Style Sheets (CSS).

Extensible Markup Language (XML)

Hierarchisch aufgebaute Textdatei, die sowohl computerlesbar ist, als auch von Menschen interpretiert werden kann.

CD mit den Quelltexten der Implementierung

Die Dokumentation der CD-ROM befindet sich in der Datei `Liesmich.htm` im Wurzelverzeichnis der CD.