Implementierung eines qualifiziert signierenden Timestamping-Servers

Dimplomarbeit von Thomas Abbé

1. März 2007

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Theoretische Informatik

Betreuer: Marcus Lippert

Prüfer: Prof. Dr. Johannes Buchmann

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 1. März 2007 Thomas Abbé

Danksagung

An dieser Stelle möchte ich allen Personen danken, die zum Gelingen dieser Diplomarbeit beigetragen haben.

Besonderen Dank gilt

- meinem Betreuer Marcus Lippert für die fachliche Unterstützung dieses interessanten und herausfordernden Themas
- Professor Buchmann, der mir die Welt der Public Key Infrastrukturen näher gebracht hat
- den Mitarbeitern von FlexSecure für die gute Zusammenarbeit
- Frank Thilo Müller für die vielen Tipps und Anregungen
- Stefan Uhrig, meinem treuen Wegbegleiter durch das Studium, der jederzeit für Fragen ansprechbar war
- meiner Familie, insbesondere meinen Eltern für ihre Unterstützung, ohne die diese Diplomarbeit nicht zustande gekommen wäre
- meiner Verlobten, Ingrid Labrosse, für ihr Verständnis und ihren moralischen Beistand

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AOP	Aspektorientierte Programmierung
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certififaction Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
EJB	Enterprise JavaBean
ETSI	European Telecommunications Standards Institute
FIFO	First In First Out
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoC	Inversion of Control
J2EE	Java 2 Platform Enterprise Edition
JAAS	Java Authentication and Authorization Service
Java EE	Java Platform, Enterprise Edition
JCA	J2EE Connector Architecture
JDBC	Java Database Connectivity
JDK	Java Development Kit
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JTA	Java Transaction API
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
OCSP	Online Certificate Status Protocol
PGP	Pretty Good Privacy
PIN	Personal Identification Number
PKCS	Public Key Cryptography Standard
PKI	Public-Key-Infrastruktur
POJI	Plain Old Java Interface
POJO	Plain Old Java Object
RA	Registration Authority
RFC	Request for Comments
RMI	Remote Method Invocation

SHA ..... Secure Hash Algorithm
SigG ..... Signaturgesetz
SigV ..... Signaturverordnung
SQL ..... Structured Query Language
SSEE ..... Sichere Signaturerstellungseinheit
TLS ..... Transport Layer Security
TSA ..... Time Stamping Authority
TSP ..... Time Stamp Protocol
UML ..... Unified Modeling Language
URL ..... Uniform Resource Locator
XML ..... Extensible Markup Language
ZDA ..... Zertifizierungsdiensteanbieter

Kapitel 1

Einleitung

Durch die Gleichstellung der elektronischen Signatur mit der eigenhändigen Unterschrift eröffnen sich neue Möglichkeiten für den elektronischen Schriftverkehr.
Seitdem ist die Absicherung elektronischer Geschäftsprozesse durch den Einsatz von rechtssicheren Signaturen mit kryptographischen Methoden möglich geworden. Durch die Automatisierung von solchen Geschäftsprozessen - darunter fallen beispielsweise die elektronische Steuererklärung oder Rechnungsstellung - ergeben sich Einsparpotentiale an Zeit und Kosten.
Da der eigenhändigen Unterschrift eine hohe Bedeutung als individuelle Willenserklärung im Rechtsverkehr zukommt, werden an die elektronische Signatur hohe Anforderungen gestellt. So muss einerseits der Urheber einer elektronischen Signatur eindeutig identifiziert werden können (Authentizität), andererseits müssen die Integrität und Nichtabstreitbarkeit der Signatur sichergestellt sein. Ebenso gelten für elektronische Dokumente die gleichen Anforderungen wie für papiergebundene Dokumente, insbesondere bezüglich ihrer Aufbewahrungsdauer.
In Deutschland wird die elektronische Signatur durch das Signaturgesetz [6] geregelt. Vor allem an Zertifizierungsdienste werden demnach hohe Anforderungen gestellt, damit die Beweiskraft der so genannten *qualifizierten elektronischen Signatur* gewährleistet wird.
In diesem Zusammenhang spielen auch Zeitstempeldienste eine Rolle. Ein Zeitstempel ist ein Beweis dafür, dass ein Dokument zu einem bestimmten Zeitpunkt existiert hat. Wurde ein Dokument mit einem Zeitstempel versehen, so ist dadurch ausgeschlossen, dass nachträglich das Datum geändert werden kann.
Neben dem Beweis zur Einhaltung von Fristen werden Zeitstempel auch für die Langzeitarchivierung von elektronisch signierten Dokumenten verwendet.
Durch den Fortschritt von Wissenschaft und Technik können die kryptographischen Algorithmen unsicher werden, wodurch die Beweiskraft der Signaturen gefährdet ist. Daher ergibt sich der Bedarf, die elektronische Signatur von Zeit zu Zeit mit einem Zeitstempel zu erneuern.

1.1 Ziele

Das Ziel dieser Diplomarbeit ist die Entwicklung eines Timestamping-Servers zur Ausstellung von rechtskräftigen Zeitstempeln. In dieser Arbeit sollen durchgeführte Recherchen sowie alle Phasen der Entwicklung, angefangen von Spezifikation, Konzeption, Implementierung und Tests beschrieben werden.
Um Zeitstempeldienste besser einordnen zu können, werden in Kapitel 2 verschiedene grundlegende Konzepte vorgestellt. Zuerst werden kryptographische Methoden angesprochen, die als

Grundlage für Public-Key-Infrastrukturen dienen. Da der entwickelte Timestamping-Server eine Java-Enterprise-Anwendung ist, wird anschließend ein Überblick über die Java Enterprise Edition und verwandte Technologien gegeben.
In Kapitel 3 werden gesetzliche Anforderungen diskutiert, die mit elektronischen Signaturen und Zeitstempeldiensten verbunden sind. Insbesondere wird auf die Problematik zur langfristigen Archivierung elektronisch signierter Dokumente eingegangen, bevor einige technische Standards und Lösungen für den Einsatz von Zeitstempeln beschrieben werden.
Der im Laufe dieser Diplomarbeit entstandene Timestamping-Server wird anschließend in Kapitel 4 ausführlich behandelt. Dabei werden alle Phasen der Entwicklung abgedeckt.
In Kapitel 5 werden diese Arbeit und die daraus gewonnenen Erkenntnisse noch einmal zusammengefasst.
Im den Anhängen A und B befinden sich schließlich Informationen zur Erstellung und Konfiguration des Timestamping-Servers.

1.2 Konventionen

In dieser Diplomarbeit werden wichtige Begriffe *kursiv* hervorgehoben. Klassennamen, Attribute oder ähnliche Typbezeichner sowie URLs werden jeweils in `Schreibmaschinenschrift` gesetzt. Literaturverweise werden in eckigen Klammern [...] angegeben.

Kapitel 2

Grundlagen

2.1 Kryptographie

Die Kryptographie wurde ursprünglich als die Lehre der Datenverschlüsselung bezeichnet. Heutzutage umfasst die Kryptographie über die Verschlüsselung hinaus noch weitere Verfahren, um eine Reihe von *Sicherheitszielen* zu erreichen.
Einige wichtige Sicherheitsziele sind:

Vertraulichkeit Das Ziel von *Vertraulichkeit* ist der Schutz von Informationen vor unberechtigtem Zugriff.

Authentizität Unter der *Authentizität* von elektronischen Daten versteht man den Nachweis, dass sie zweifelsfrei einem Urheber oder Absender zugeordnet werden können.

Integrität Die *Integrität* ist ein Nachweis, dass Daten vollständig und unverändert sind.

Nichtabstreitbarkeit Unter *Nichtabstreitbarkeit* versteht man den Nachweis, dass man z. B. die Urheberschaft einer Signatur im Nachhinein nicht abstreiten kann.

In diesem Abschnitt werden einige kryptographische Grundlagen behandelt, die insbesondere die Sicherheitsziele Vertraulichkeit, Integrität und Authentizität abdecken. Weitere Informationen hierzu und den vorgestellten Verfahren finden sich in [12] und [19].

2.1.1 Hashfunktionen

Eine *Hashfunktion* ist eine Abbildung, die Daten beliebiger Länge auf den so genannten Hashwert mit einer festen Länge abbildet:

$$h : \{0,1\}^* \to \{0,1\}^n, \quad n \in \mathbb{N}.$$

Die Anwendungen von Hashfunktionen sind vielfältig. Sie werden z. B. zur schnellen Suche von Daten in großen Datenbeständen mittels Hashtabellen verwendet. In dieser Diplomarbeit stehen aber zwei andere Anwendungsfelder im Vordergrund:

- Elektronische Signaturen (s. auch Abschnitt 2.1.2) und
- Prüfsummen. Dabei kann anhand des Hashwerts die Integrität von Daten überprüft werden.

Um Hashfunktionen in diesem Umfeld nutzen zu können, müssen sie *unumkehrbar* und *kollisionsresistent* sein.
Eine Hashfunktion ist eine *Einwegfunktion*, wenn sie nicht umkehrbar ist, das heißt, es ist praktisch unmöglich, für einen festen Hashwert w ein $x \in \{0,1\}^*$ zu finden mit $h(x) = w$.
Ebenso ist eine Hashfunktion kollisionsresistent, wenn es praktisch unmöglich ist, eine Kollision von h zu finden. Unter einer Kollision versteht man in diesem Zusammenhang ein Paar $(x, x') \in (\{0,1\}^*)^2$, für die $x \neq x'$ aber $h(x) = h(x')$ gilt.
Hashfunktionen müssen des Weiteren effizient berechnet werden können, da zum Teil große Datenmengen verarbeitet werden.
Einige bekannte Hashfunktionen sind

SHA-1 Der sichere Hashalgorithmus (Secure Hash Algorithm) ist ein weit verbreitetes Hashverfahren, das Hashwerte von 160 Bit Länge erzeugt.

Allerdings verliert dieser Algorithmus langsam seine Sicherheitseignung, da erfolgreich Angriffe gegen SHA-1 gestartet wurden. Weitere Information dazu finden sich in [53].

SHA-2-Familie Zur SHA-2-Familie gehören die Hashverfahren SHA-224, SHA-256, SHA-384 und SHA-512. Sie bauen alle auf SHA-1 auf, erzeugen jedoch längere Hashwerte.

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) bildet Hashwerte auf 160 Bit ab.

MD5 (Message Digest 5) Erzeugt Hashwerte von 128 Bit Länge. Dieses Verfahren gilt aber bereits heutzutage als unsicher, da sich Kollisionen finden lassen.

2.1.2 Signatur

An die *elektronische Signatur* werden die gleichen Ansprüche wie an die eigenhändige Unterschrift gestellt.
Angenommen, Alice unterschreibt ein Dokument, so muss auch im Nachhinein überprüfbar sein, dass die Unterschrift von Alice und nur von ihr getätigt wurde. Ebenso kann Alice ihre Unterschrift nicht mehr abstreiten, wodurch ein Beweiswert z. B. vor Gericht besteht.
Bei der Erzeugung von elektronischen bzw. digitalen Signaturen werden in der Regel *asymmetrische Verfahren* verwendet. Ausschlaggebend ist hierfür die Verwendung eines Schlüsselpaares. Signaturen werden mit einem *privaten* Signaturschlüssel erzeugt und können mit einem *öffentlichen* Schlüssel verifiziert werden.
Will Alice ein Dokument x signieren, so geht sie folgendermaßen vor:

1. Sie berechnet den Hashwert $w = h(x)$ einer gängigen kollisionsresistenten Hashfunktion.

2. Anschließend wird das Signaturverfahren σ auf w angewendet. Die Signatur s erhält man durch die Berechnung von $s = \sigma(w, d)$, wobei d der private Schlüssel von Alice ist.

Damit Bob die Signatur von Alice überprüfen kann, muss er in Besitz ihres öffentlichen Schlüssels und des signierten Dokuments x sein.
Bob verifiziert dazu die Signatur mit dem öffentlichen Schlüssel von Alice und vergleicht diesen Wert mit dem Hashwert $h(x)$, den er ebenfalls berechnet, da er die verwendete Hashfunktion kennt.
Die geleistete Signatur ist jedoch nur so lange gültig, wie auch die verwendeten kryptographischen Verfahren sicher sind. Aus diesem Grund muss die Signatur von Zeit zu Zeit erneuert

werden, sollten die Algorithmen ihre Sicherheitseignung verlieren. Auf diese Problematik wird im folgenden Kapitel näher eingegangen.
Gebräuchliche Signaturverfahren sind beispielsweise:

RSA Der weit verbreitete Algorithmus findet neben der elektronischen Signatur auch in der Verschlüsselung von Daten Verwendung. Die Sicherheit von RSA basiert auf der Faktorisierung von Primzahlen.

DSA Der Digital Signature Algorithm (DSA) ist ein US-amerikanischer Standard für digitale Signaturen und eine effizientere Variante des ElGamal-Verfahrens, das von der Lösung diskreter Logarithmen abhängt.

2.2 Public-Key-Infrastrukturen

In vorigem Abschnitt wurden bereits Signaturen erwähnt. In diesem Zusammenhang wurde auch das Verfahren erklärt, wie eine Signatur erzeugt und verifiziert werden kann. Nun stellt sich jedoch die Frage, wie man Alice einer von ihr getätigten Signatur auch verlässlich zuordnen kann. Ebenso muss ihr geheimer Schlüssel auch vor Fälschung und Missbrauch geschützt sein. Aus diesem Grund sind *Public-Key-Infrastrukturen (PKI)* von zentralem Interesse. Eine auf dem X.509-Standard [38] basierende PKI ist eine Dienstleistung eines Zertifizierungsdiensteanbieters (s. auch Abschnitt 3.2.2) und setzt sich in der Regel aus mehreren Komponenten zusammen. Wesentliche Bestandteile von PKIs sind

- Zertifikate,
- Sperrinformationen,
- Anwenderkomponenten,
- Registrierungsinstanz,
- Zertifizierungsinstanz,
- Verzeichnisdienst,
- Zeitstempeldienst.

In dieser Diplomarbeit wird nur auf einige dieser Komponenten näher eingegangen. Weitere Informationen zu Public-Key-Infrastrukturen finden sich in [62] und [16].

2.2.1 Zertifikate

Um einer Person einen öffentlichen Schlüssel zuzuordnen, werden *Zertifikate* verwendet, die im X.509 Standard [38] spezifiziert werden. Ein Zertifikat besteht dabei aus strukturierten Daten, die von einer vertrauenswürdigen *Zertifizierungsinstanz* signiert werden, wodurch die Richtigkeit der im Zertifikat enthaltenen Daten garantiert wird. Durch die Signatur wird zusätzlich sichergestellt, dass der Inhalt des Zertifikats nachträglich nicht mehr geändert werden kann, ansonsten wäre die Signatur nicht mehr gültig.
Im Folgenden werden die wichtigsten Daten genannt, die in einem Zertifikat gespeichert werden:

- Name des Eigentümers (Schlüsselinhaber) des Zertifikats

- Öffentlicher Schlüssel des Eigentümers
- Gültigkeitsdauer des Zertifikats
- Seriennummer des Zertifikats
- Aussteller (*Issuer*) des Zertifikats
- Angaben über Verwendungszweck des öffentlichen Schlüssels

Auch die Zertifizierungsinstanz besitzt ein Zertifikat, das in allen ausgestellten Zertifikaten referenziert wird. Auf diese Weise kann ein *Zertifizierungspfad* konstruiert werden. Bei der Verifikation eines Zertifikats kann man, ausgehend von der *Wurzelinstanz*, der man vertraut, alle weiteren Zertifikate des Zertifizierungspfades überprüfen.
Ein Zertifikat ist dabei als gültig anzusehen, wenn

- die Signatur gültig ist,
- die Gültigkeit des Zertifikates nicht abgelaufen ist,
- das Zertifikat nicht gesperrt wurde (z. B. bei eventuellem Missbrauch des privaten Schlüssels),
- die geleistete Signatur bzw. Verschlüsselung dem Verwendungszweck des Zertifikats entspricht.

2.2.2 Sperrinformationen

Manchmal muss ein Zertifikat vor Ablauf seiner Gültigkeit gesperrt werden, z. B. um Missbräuche des Signaturschlüssels zu verhindern. Mit Hilfe von *Sperrinformationen* kann ein Nutzer also feststellen, ob ein Zertifikat gesperrt wurde. Es sei an dieser Stelle angemerkt, dass das Fehlen von Sperrinformationen keine Garantie für die Gültigkeit eines Zertifikates bietet (s. Abschnitt 2.2.1).
Konkret können die in RFC 3280 [38] spezifizierten Sperrlisten (*Certificate Revocation List, CRL*) oder auch Abfragen über das *Online Certificate Status Protocol (OCSP)* [48] als Sperrinformationen herangezogen werden.

2.2.3 Anwenderkomponenten

Die Anwenderkomponente kommt beim Endbenutzer zum Einsatz, wenn er z. B. Daten signieren will. Sie besteht im Wesentlichen aus drei Komponenten:

- dem Anwendungssystem. Damit ist die konkrete Anwendung gemeint, die die Signaturerzeugung und -verifikation anstößt.
- der *Signaturanwendungskomponente* als Schicht zwischen dem Anwendungssystem und der Signaturerstellungseinheit. Meistens werden sie als Programmierschnittstellen realisiert.
- der *Signaturerstellungseinheit*, wo der private Signaturschlüssel gespeichert und die Signatur erzeugt wird. Es kann sich dabei sowohl um Hardtoken (der Signaturschlüssel ist z. B. auf einer Chipkarte gespeichert) als auch um Softtoken (der Signaturschlüssel ist z. B. in einer verschlüsselten Datei abgelegt) handeln.

2.2.4 Registrierungsinstanz

In der *Registrierungsinstanz* (Registration Authority, RA) werden gewöhnlich die Zertifikate beantragt oder auch bei Bedarf gesperrt. Ebenso überprüft die Registrierungsinstanz die Identität der Antragssteller.

2.2.5 Zertifizierungsinstanz

Die *Zertifizierungsinstanz* (Certififaction Authority, CA) ist hauptsächlich für das Ausstellen von Zertifikaten verantwortlich. In manchen Fällen werden hier auch die privaten Schlüssel generiert, sofern sie nicht vom Endbenutzer selbst erzeugt wurden.

Die Zertifizierungsinstanz ist der sicherheitskritischste Teil einer PKI. Sollte ihr privater Schlüssel kompromittiert werden, so kann kein Teilnehmer mehr ihren Diensten vertrauen.

2.2.6 Verzeichnisdienst

In einem *Verzeichnisdienst* werden in der Regel Zertifikate nachprüfbar und möglicherweise auch abrufbar gehalten. Die Nachprüfbarkeit von Zertifikaten reduziert sich dabei auf die Bereitstellung von Sperrinformationen (s. 2.2.2), so dass keine generelle Aussage über die Gültigkeit eines Zertifikats getroffen werden kann.

In der Praxis werden normalerweise hierarchische Datenbanken eingesetzt, die kompatibel zum X.500-Standard sind. Meistens wird dabei auf das Lightweight Directory Access Protocol (LDAP) zurückgegriffen.

2.2.7 Zeitstempeldienst

Ein Zeitstempeldienst, auch *Time Stamping Authority (TSA)* genannt, bestätigt mit Hilfe von Zeitstempeln, dass Daten zu einem bestimmten Zeitpunkt existiert haben.

In Kapitel 3 werden Zeitstempeldienste speziell in Hinblick auf rechtliche Rahmenbedingungen und technische Standards betrachtet.

2.2.8 Gültigkeitsmodelle

Bei der Überprüfung einer Signatur gibt es mehrere Modelle, wann die Signatur als gültig zu erachten ist.

Bei dem *Schalenmodell* zählt allein der Zeitpunkt der Verifikation, das heißt, dass alle Zertifikate der Zertifikatskette zum Prüfzeitpunkt gültig sein müssen. Insbesondere lässt bereits ein abgelaufenes oder gesperrtes Zertifikat die Überprüfung fehlschlagen.

Das *Hybridmodell* baut auf dem einfachen Schalenmodell auf, jedoch wird zur Signaturprüfung der Signaturzeitpunkt herangezogen. Alle Zertifikate vom Benutzerzertifikat bis hin zu den CA-Zertifikaten müssen zum Zeitpunkt der Signatur gültig gewesen sein.

Nach dem *Kettenmodell* müssen alle Zertifikate zum Verwendungszeitpunkt des Signaturschlüssels gültig sein. Genauer gesagt heißt das, dass das Benutzerzertifikat zum Zeitpunkt der Signatur gültig sein muss und dass das jeweilige CA-Zertifikat zum Zeitpunkt der Ausstellung ebenfalls gültig war.

2.3 Java Platform, Enterprise Edition

Nachdem in vorigen Abschnitten einige Grundlagen zu elektronischen Signaturen und PKIs angesprochen wurden, sollen an dieser Stelle einige Java-Technologien vorgestellt werden. In diesem Zusammenhang wird besonders auf die Java Enterprise Edition eingegangen, da sie ein zentraler Aspekt des im Laufe dieser Diplomarbeit entstandenen Timestamping-Servers ist.

2.3.1 Übersicht

Die *Java Platform, Enterprise Edition*, abgekürzt Java EE oder in älteren Versionen auch J2EE, ist eine Spezifikation eines Komponentenmodells für mehrschichtige Server-Anwendungen in Java. Sinn und Zweck von Java EE ist es, dass sich die Entwickler auf die so genannte Business-Logik konzentrieren können und sich nicht um ständig benötigte Funktionalitäten wie Sicherheit, Transaktionen oder Datenbankzugriffe zu kümmern brauchen. Diese Aufgaben werden von *Application Servern* übernommen.

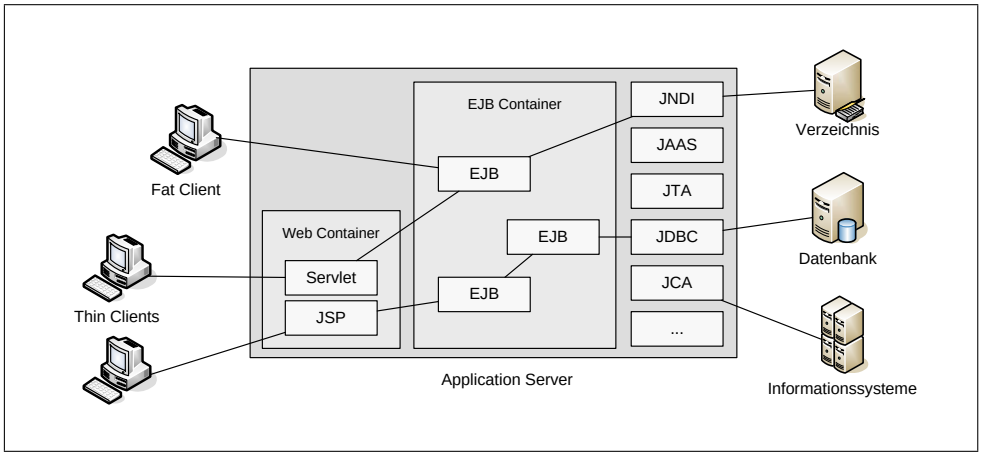

Abbildung 2.1: Überblick Java EE (vgl. [20])

Abbildung 2.1 zeigt einen Überblick über verschiedene Java EE Komponenten. Einige wichtige Konzepte von Java EE sollen an dieser Stelle besprochen werden, für weitere Informationen sei auf [59] und [54] verwiesen.

Insgesamt werden drei verschiedenen Komponenten unterschieden:

Client-Komponenten Zu Client-Komponenten zählen sowohl Client-Anwendungen als auch Java-Applets.

Web-Komponenten Unter Web-Komponenten versteht man entweder *Servlets* oder z. B. durch *Java Server Pages (JSP)* generierte Seiten. JSPs kommen bei der Erzeugung von dynamischen Webseiten zum Einsatz und erlauben es, Java-Code in statischen Inhalt einzubetten. Intern lassen sich JSPs auf Servlets zurückführen. Servlets kann man als Erweiterungen des Servers verstehen, die in der Regel Anfragen über das Hypertext Transfer Protocol (HTTP) [27] beantworten.

Web-Komponenten werden von einem *Web-Container* verwaltet, der sich u. a. um die Ausführung der Servlets kümmert.

Enterprise JavaBeans *Enterprise JavaBeans (EJB)* sind die wichtigsten Komponenten in Java EE und beschreiben die Business-Logik. EJBs werden von einem *EJB-Container* verwaltet, der Schnittstellen zu weiteren *Middleware*-Diensten wie z. B. Persistenz und Sicherheit anbietet sowie sich um den Lebenszyklus der Komponenten kümmert.

Es gibt mehrere Arten von EJBs:

Session-Beans *Session-Beans* kapseln in die Geschäftslogik und werden in der Regel von Clients angesprochen. Man unterscheidet zwischen *stateless* (zustandslosen) und *stateful* (zustandsbehafteten) Session-Beans.

Der Unterschied zwischen stateless und stateful liegt darin, dass ein Stateful Session-Bean genau einem Client zugeordnet wird und so Daten (der Zustand) zwischen aufeinander folgenden Methodenaufrufen (serverseitig) gespeichert werden können.

Da ein Stateless Session-Bean mehreren Clients zugeordnet werden kann, muss der Zustand clientseitig verwaltet werden und gegebenenfalls bei Methodenaufrufen übergeben werden.

Message-Driven-Beans Mit *Message-Driven-Beans* kann man asynchron über das Verschicken von Nachrichten kommunizieren.

Entities *Entity-Beans* bilden die (persistenten) Datenobjekte ab.

Wie bereits angesprochen, bietet ein Application Server mehrere Dienste und Schnittstellen an. Einige wichtige Dienste werden im Folgenden aufgelistet:

JNDI Das *Java Naming and Directory Interface (JNDI)* ist eine Schnittstelle, über die auf Namens- und Verzeichnisdienste zugegriffen werden kann. Insbesondere wird JNDI für den Zugriff auf Java EE Komponenten verwendet.

JAAS Der *Java Authentication and Authorization Service (JAAS)* ist für die Authentifizierung und Authorisierung von Benutzern zuständig. In Abschnitt 2.3.3 werden einige Konzepte von JAAS näher vorgestellt.

JTA Die *Java Transaction API (JTA)* steuert und verwaltet Transaktionen.

JDBC Die *Java Database Connectivity (JDBC)* erlaubt die Anbindung von relationalen Datenbanken.

JCA Die *J2EE Connector Architecture (JCA)* ist eine Schnittstelle, um heterogene Informationssysteme zu integrieren.

2.3.2 EJB 3.0

Nachdem in vorigem Abschnitt ein kurzer Überblick über Java EE gegeben wurde, steht an dieser Stelle die aktuelle EJB 3.0-Spezifikation im Mittelpunkt.
Die Entwicklung von Enterprise JavaBeans wurde zuletzt (bis EJB Spezifikation 2.1) sehr komplex. Ein EJB 2.x besteht neben der eigentlichen Quellcodedatei aus bis zu sechs weiteren Dateien, die jeweils erstellt und gepflegt werden müssen. Dazu gehören verschiedene Interfaces für den lokalen und entfernten Zugriff auf die Komponenten sowie eine Reihe von *Deployment-Deskriptoren*. Die Wartung dieser Anzahl von Dateien ist fehleranfällig und zeitaufwändig, da etwaige Konfigurationsfehler erst während des *Deployens* auftreten.
Aus diesem Grund wurde mit der aktuellen EJB Spezifikation 3.0 eine neue Richtung eingeschlagen, die das Entwickeln von Java EE-Anwendungen grundlegend vereinfacht. Im Folgenden werden einige Neuerungen erwähnt.

- Deployment-Deskriptoren werden nicht mehr benötigt. Stattdessen werden *Annotationen* verwendet, die mit dem JDK 1.5 in Java eingeführt wurden. Mit Annotationen kann eine Klasse mit Metadaten versehen werden. Eine Einführung zu Annotationen findet sich in [56].
- Die EJBs sind nun *POJOs* (Plain Old Java Objects). EJBs müssen also nicht mehr frameworkspezifische Interfaces implementieren. Dadurch können die neuen EJBs schneller entwickelt werden, besser mit anderen EJBs kombiniert werden sowie auch außerhalb des EJB-Containers genutzt und getestet werden.
- Als Schnittstellen für Clients können *POJIs (Plain Old Java Interfaces)* verwendet werden, die die EJBs implementieren.
- Die neue Spezifikation sieht *Dependency Injection* vor. Abhängigkeiten zu anderen Objekten können über Annotationen aufgelöst werden.
- Vereinfachung von Persistenzmechanismen, insbesondere die direkte Unterstützung von OR-Mappern[1].
- Eingebautes Transaktions- und Sicherheitsmanagement wird mit Annotationen gesteuert.
- Unterstützung von *Aspektorientierter Programmierung (AOP)* durch den Einsatz von *Interceptors*.

In Listing 2.1 findet sich ein vereinfachtes Beispiel eines EJBs, das im Rahmen dieser Diplomarbeit entwickelt wurde.
Ein EJB implementiert in der Regel ein *Business-Interface*[2] als Schnittstelle für Clients. Ein Business-Interface muss zwar nicht unbedingt vorhanden sein, ist aber aus Gründen der Wiederverwendbarkeit und Fehlererkennung zur Kompilierzeit empfehlenswert. In diesem Fall wird das Business-Interface `TspProcessor` vom EJB `TspProcessorBean` implementiert.
Wie bereits erwähnt, werden in der EJB Spezifikation 3.0 Annotationen intensiv genutzt, um alle Informationen zu beschreiben, für die früher ein Deployment-Deskriptor vonnöten war. Die Annotationen werden anschließend vom EJB-Container ausgelesen und verarbeitet.
Stateless Session-Beans werden durch `@Stateless` annotiert. Analog dazu existieren andere Annotationen für die anderen EJB-Typen.
Durch `@Local` wird die Komponente (genauer gesagt das Business-Interface) Clients zur Verfügung gestellt, die in der gleichen *Java Virtual Machine (JVM)* wie der EJB-Container ausgeführt werden. Mit der Annotation `@Remote` kann der Zugriff von außerhalb aktiviert werden. Dazu wird das Protokoll *RMI-IIOP* [3] verwendet.
Mit der Annotation `@EJB` werden schließlich andere EJBs durch einen JNDI-*Lookup* in die Komponente injiziert.

Listing 2.1: Die EJB-Komponente TspProcessor

```
/*
Das Business-Interface TspProcessor
*/
```

[1] *Object-Relational Mapping* bezeichnet das Abbilden von Objekten auf relationale Datenbankmodelle.
[2] Ein Business-Interface deklariert die Business-Methoden, die ein EJB für den Zugriff von außen anbietet.
[3] *Remote Method Invocation (RMI)* ist der Standardweg in Java, um mit verteilten Objekten zu kommunizieren. In Java EE wird *RMI-IIOP*, eine Erweiterung von RMI, für die Integration von *CORBA* verwendet.

```
public interface TspProcessor
{
  public TimeStampResp processTimeStampRequest(TimeStampReq
     request);
}

/*
Das EJB TspProcessorBean
*/
@Local({ TspProcessor.class })
@Remote({ TspProcessor.class })
@Stateless
public class TspProcessorBean implements TspProcessor
{
        @EJB
        private CryptoProcessor cryptoProc;


  public TimeStampResp processTimeStampRequest(TimeStampReq
     request)
  {
    TimeStampResp response = new TimeStampResp();

    ...

    return resp;
  }

}
```

2.3.3 Java Authentication and Authorization Service

Die Java EE-Spezifikation beschreibt zwar Schnittstellen zur Sicherheit von Application Servern, jedoch nicht, wie genau die Authentifizierung und Authorisierung von Benutzern innerhalb der Server auszusehen hat. Oftmals wird das JAAS-Framework verwendet.
Da JAAS im Rahmen dieser Diplomarbeit ebenfalls benutzt wurde, soll an dieser Stelle eine kurze Einführung bezüglich der Authentifizierung von Benutzern gegeben werden. Für eine ausführliche Betrachtung wird auf [22] verwiesen.
Ein zentraler Bestandteil von JAAS ist das *Subject*, das einem Benutzer zugeordnet wird und mehrere Identitäten verwaltet. Eine Identität, auch *Principal* genannt, entspricht dem Anmeldenamen. Ein Benutzername, eine E-Mailadresse oder eine Personalnummer sind z. B. jeweils eine Identität des Benutzers.
Wichtig in diesem Zusammenhang sind auch die Anmeldedaten, die so genannten *Credentials*. Credentials können jeweils ein Benutzername, ein Passwort, ein Zertifikat oder auch biometrische Daten sein, die zur Identifikation des Benutzers geführt haben.
Die Authentifizierung der Benutzer wird schließlich durch die *Login-Module* durchgeführt, wobei mehrere Login-Module hintereinander geschaltet werden können. Dies hat den Vorteil, dass

sie leicht austauschbar und konfigurierbar sind, und dass auch mehrere Anmeldemöglichkeiten gleichzeitig möglich sind.
Der Anmeldevorgang der Login-Module läuft in der Regel in zwei Schritten ab:

1. Während des *Login*-Vorgangs werden die Anmeldedaten auf ihre Korrektheit überprüft.

2. Nur nach erfolgreichem Login des gesamten Anmeldevorgangs werden die Anmeldedaten während des *Commit*-Vorgangs dem Subject hinzugefügt.

Ob der gesamte Anmeldevorgang erfolgreich ist, richtet sich nach den Kontroll-Flags der einzelnen Login-Module (s. Tabelle 2.1).

FLAG	ERFOLG ERFORDERLICH	AUSFÜHRUNG WEITERER MODULE
required	ja	immer
requisite	ja	bei Erfolg
sufficient	nein	bei Misserfolg
optional	nein	immer

Tabelle 2.1: Kontroll-Flags der Login-Module (vgl. [47])

Die Bedeutung der Kontrollflags soll anhand eines Beispiels, das auch in dieser Diplomarbeit zum Einsatz kommt, erklärt werden.
Angenommen, man würde drei Login-Module mit den Kontroll-Flags `sufficient`, `requisite` und `required` hintereinander schalten. Wie in Tabelle 2.2 dargestellt, können mehrere Fälle unterschieden werden.
War der Login-Vorgang beim ersten Login-Modul erfolgreich, so wird der Gesamtvorgang sofort erfolgreich beendet. Schlägt der Login-Vorgang bei Modul eins fehl, so muss der Login bei den Modulen zwei und drei erfolgreich sein, falls der gesamte Anmeldevorgang noch erfolgreich sein soll. In allen anderen Fällen schlägt er fehl.

LOGIN-MODUL	FALL 1	FALL 2	FALL 3	FALL 4
1 (sufficient)	erfolgreich	fehlgeschlagen	fehlgeschlagen	fehlgeschlagen
2 (requisite)		erfolgreich	erfolgreich	fehlgeschlagen
3 (required)		erfolgreich	fehlgeschlagen	
Gesamtvorgang	erfolgreich	erfolgreich	fehlgeschlagen	fehlgeschlagen

Tabelle 2.2: Hintereinanderschaltung von Login-Modulen

2.3.4 Java Management Extensions

EJBs sind von einem Container verwaltete Business-Objekte, die nicht auf die Verwaltung von Ressourcen ausgelegt sind (vgl. [57]). Mit EJBs ist es also nicht möglich, Geräte (z. B. Kartenleser) direkt anzusprechen, den Zustand des Application Servers zu erfragen oder gar Dienste zur Laufzeit zu verwalten.
An dieser Stelle kommen die *Java Management Extensions (JMX)* zum Einsatz, die eine einheitliche Verwaltung und Überwachung von Java-Anwendungen und Diensten erlauben. Im Java EE-Umfeld wird JMX hauptsächlich zur Administration von Application Servern eingesetzt.

Der Zugriff auf Ressourcen wird von den so genannten *MBeans (Managed Beans)* übernommen, die in einem *MBean Server* registriert werden. Dadurch kann auf die MBeans von anderen Anwendungen aus zugegriffen werden.
Innerhalb der Server-JVM existiert ein MBean als *Singleton*[4]. Anders als bei EJBs kommunizieren Anwendungen also immer mit dem gleichen Objekt, was für die Verwaltung von Ressourcen essenziell ist.
Für eine tiefere Betrachtung von JMX sei auf [55] und [41] verwiesen.

2.3.5 Spring

Aufgrund der Komplexität von J2EE-Anwendungen konnten sich ebenfalls andere Frameworks in der Entwicklung von Multitier-Anwendungen etablieren. Ein Vertreter davon ist das Spring-Framework[5], das in dieser Diplomarbeit Verwendung findet.
Spring ist ein leichtgewichtiges[6] Open Source Applikationsframework, das speziell die J2EE-Entwicklung vereinfachen soll und inzwischen weit verbreitet ist - nicht zuletzt, weil es sich mit *Dependency Injection* einen Namen gemacht hat.
Bei Dependency Injection geht es darum, Abhängigkeiten zwischen Objekten nicht selbst im Quellcode aufzulösen, sondern die abhängigen Objekte von außen durch ein Framework zu „injizieren“. Dieses Verfahren, wobei der Programmfluss durch das Framework kontrolliert wird, nennt man auch *Inversion of Control (IoC)* (s. dazu auch den Artikel von M. Fowler [29] zu diesem Thema).
Im Wesentlichen werden zwei Formen der Dependency Injection benutzt:

Constructor-Injection Die betreffenden Klassen haben einen oder mehrere Konstruktoren, bei denen die Abhängigkeiten als Parameter übergeben werden. Dadurch wird die Instanziierung von konsistenten Objekten garantiert.

Setter-Injection Die abhängigen Objekte werden durch Setter-Methoden gesetzt.

In der Praxis ist es abzuwägen, welche der beiden Formen man einsetzt. Spring empfiehlt jedoch den Einsatz von Setter-Injection, die anhand eines vereinfachten Beispiels aus dieser Arbeit veranschaulicht werden soll:

Listing 2.2: Setter-Injection

```
public class Client
{

  protected CertificateVerifier certVerifier;

        public CertificateVerifier getCertificateVerifier()
        {
                return certVerifier;
        }
```

[4] Gemeint ist das Singleton Design Pattern (s. [21]).
[5] http://www.springframework.org
[6] Unter „leichtgewichtig“ versteht man die Eigenschaft, Komponenten in mehreren Umgebungen verwalten zu können. Gleichzeitig sind die Komponenten nicht an ein Framework gekoppelt und lassen sich durch Konfiguration austauschen.

```
    public void setCertificateVerifier(CertificateVerifier
        value)
    {
            certVerifier = value;
    }

}
```

Während bei EJB 3.0 Abhängigkeiten über JNDI-Lookups aufgelöst werden, verfolgt Spring einen anderen Ansatz:
Spring benutzt als Business-Objekte ebenfalls bereits erwähnte POJOs. Anhand von Metadaten werden diese Objekte im so genannten IoC-Container beliebig erstellt und miteinander verknüpft. Wie Abbildung 2.2) zeigt, stellt der Container fertig konfigurierte und sofort benutzbare Objekte zur Verfügung.

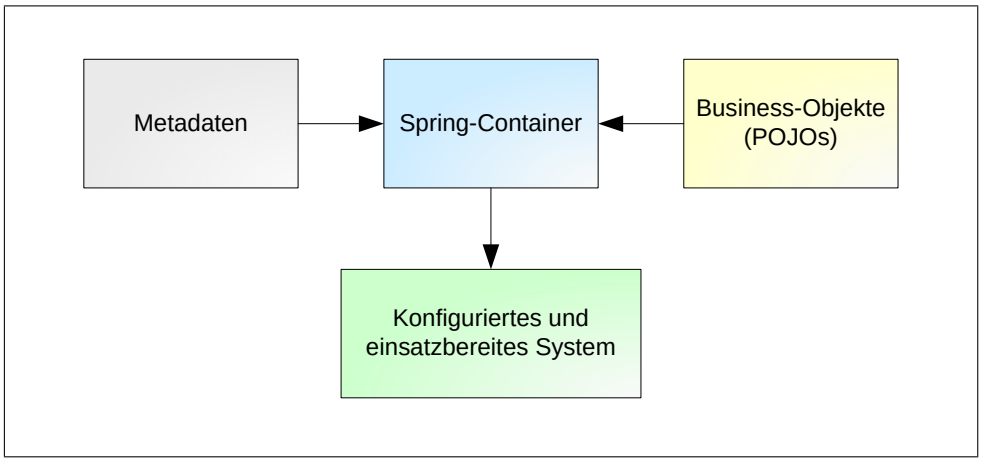

Abbildung 2.2: Der Spring IoC-Container (vgl. [42])

Die Metadaten, in der die Abhängigkeiten definiert werden, liegen in einer XML-Datei, also *außerhalb* des IoC-Containers. Dadurch können die Business-Objekte besonders einfach konfiguriert werden.
Folgendes Listing zeigt die Fortsetzung des Beispiels aus Listing 2.2. Dabei wird je nach Gültigkeitsmodell ein `CertificateVerifier` von einer *Factory*[7] erstellt und anschließend dem `Client` zugewiesen.

Listing 2.3: Konfiguration der Abhängigkeiten

```
<beans>

  <bean id="1" class="jmx.clients.Client">
    <property name="certificateVerifier" ref="CHAIN" />
  </bean>

  <bean id="CHAIN"
      class="jmx.clients.CertificateVerifierFactory"
      factory-method="createCertifiaceVerifier">
```

[7] Gemeint ist das Factory Design Pattern (s. [21]).

```
    <constructor-arg value="CHAIN" />
  </bean>

</beans>
```

Spring umfasst jedoch weit mehr Funktionalität als nur ein IoC-Container:

- Spring vereinfacht die Anbindung an Datenbanken.
- Es werden Schnittstellen für AOP angeboten.
- Spring stellt Frameworks für verteilte Anwendungen zur Verfügung.
- Spring vereinfacht die Entwicklung von Web-Applikationen.

Das Spring-Framework ist also ein Framework, das auf den Einsatz im J2EE-Umfeld ausgerichtet ist. Dabei ist es keine Konkurrenz zu anderen Frameworks, sondern kann Hand in Hand mit EJBs, OR-Mappern wie Hibernate oder anderen Diensten zusammenarbeiten.
Spring als Alternative zu EJB wird in dieser Arbeit nicht weiter verfolgt. Ebenso wird auf die Integration von Spring bezüglich des EJB-Containers nicht näher eingegangen. In diesem Zusammenhang sei auf [46] und [43] verwiesen.
Weitere Informationen zu Spring finden sich in der Referenz [42].

Kapitel 3

Zeitstempeldienst

> „Die Zeit kommt aus der Zukunft, die nicht existiert, in die Gegenwart, die keine Dauer hat, und geht in die Vergangenheit, die aufgehört hat zu bestehen."
>
> Aurelius Augustinus

3.1 Überblick

Von einem Zeitstempeldienst ausgegebene Zeitstempel gelten als Nachweis, dass Daten zu einem bestimmten Zeitpunkt existiert haben und werden in Zusammenhang mit der Signaturgesetzgebung im Wesentlichen in zwei Anwendungen eingesetzt:

1. Um sicher zu stellen, dass Signaturen zu einem bestimmten Zeitpunkt existiert haben und um so z. B. ausschließen zu können, dass sie nach Ablauf oder Sperrung eines Zertifikats erzeugt wurden,

2. Um die langfristige Beweiskraft von Signaturen zu gewährleisten, müssen sie von Zeit zu Zeit (bedingt durch die Sicherheitseignung der kryptographischen Algorithmen) mit einem Zeitstempel erneuert werden.

Der grobe Ablauf bei der Anfrage eines Zeitstempels ist in Abbildung 3.1 dargestellt. Der Client sendet die Anfrage mit dem Hashwert der Daten, die mit einem Zeitstempel versehen werden sollen, an den Zeitstempeldienst. Dabei soll der Begriff „Client" durchaus breit gefasst sein. Es kann sich dabei sowohl um Endanwender als auch um serverseitige Signatursysteme handeln. Der Zeitstempeldienst ergänzt schließlich den empfangenen Hashwert u. a. mit der gesetzlich gültigen Zeit und schickt diese Daten signiert wieder an den Client zurück.

3.2 Gesetzliche Rahmenbedingungen

Rechtliche Grundlage der elektronischen Signatur in Deutschland bietet das Signaturgesetz (SigG) [6], das durch die Signaturverordnung (SigV) [7] präzisiert wird. Die deutsche Signaturgesetzgebung wurde 2001 der EU-Richtlinie [4] angepasst, wodurch das alte Signaturgesetz von 1997 [2] abgelöst wurde. 2005 wurden SigG und SigV durch das erste Gesetz zur Änderung des Signaturgesetzes [8] wiederum abgeändert.

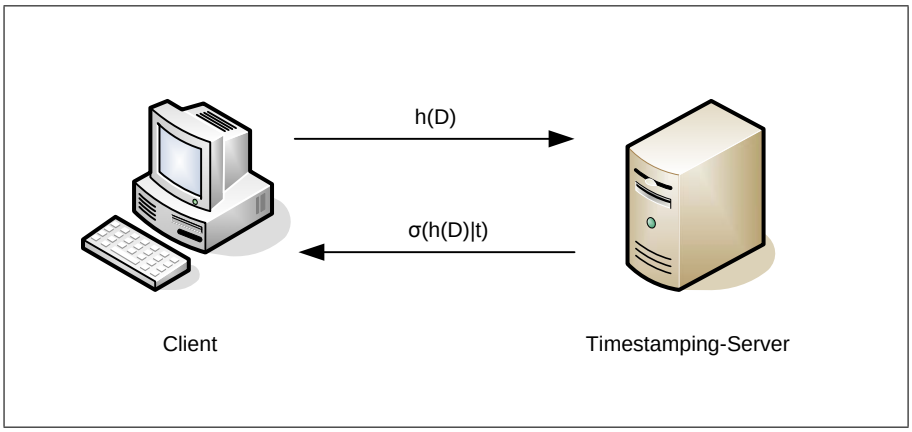

Abbildung 3.1: Überblick Zeitstempeldienst (vgl. [16])

3.2.1 Begriffsbestimmungen

Das Signaturgesetz [6] unterscheidet zwischen mehreren elektronischen Signaturen (s. Abbildung 3.2).
Gemäß § 2 SigG sind

1. *„elektronische Signaturen“* Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen,

2. *„fortgeschrittene elektronische Signaturen“* elektronische Signaturen nach Nummer 1, die

   (a) ausschließlich dem Signaturschlüssel-Inhaber zugeordnet sind,

   (b) die Identifizierung des Signaturschlüssel-Inhabers ermöglichen,

   (c) mit Mitteln erzeugt werden, die der Signaturschlüssel- Inhaber unter seiner alleinigen Kontrolle halten kann, und

   (d) mit den Daten, auf die sie sich beziehen, so verknüpft sind, dass eine nachträgliche Veränderung der Daten erkannt werden kann,

3. *„qualifizierte elektronische Signaturen“* elektronische Signaturen nach Nummer 2, die

   (a) auf einem zum Zeitpunkt ihrer Erzeugung gültigen qualifizierten Zertifikat beruhen und

   (b) mit einer sicheren Signaturerstellungseinheit erzeugt werden.

Die erwähnten *„Signaturschlüssel-Inhaber“* müssen nach § 2 Nr. 9 eine natürliche Person sein. *„Zertifizierungsdiensteanbieter (ZDA)“* (§ 2 Nr. 8) sind dabei natürliche oder juristische Personen, die qualifizierte Zertifikate oder qualifizierte Zeitstempel ausstellen. Ein ZDA hat außerdem die Wahl, sich akkreditieren zu lassen (§ 2 Nr. 15), was mit weiteren Rechten und Pflichten verbunden ist (vgl. § 15f SigG).
Ein *„qualifiziertes Zertifikat“* (§ 2 Nr. 7) ist eine elektronische Bescheinigung eines ZDA, der bestimmte gesetzlichen Anforderungen (s. Abschnitt 3.2.2) erfüllt, und ordnet einer natürlichen Person einen Signaturprüfschlüssel zu. In § 7 SigG werden die detaillierten Inhalte eines qualifizierten Zertifikats näher beschrieben.

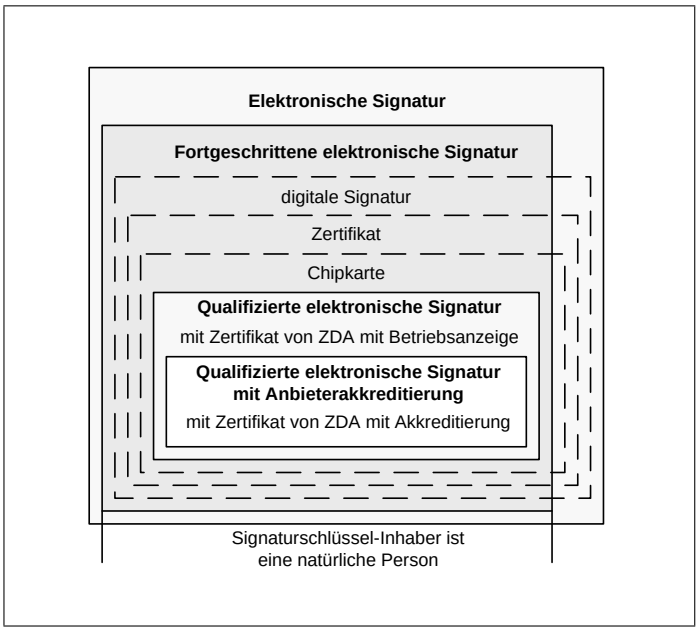

Abbildung 3.2: Abstufungen der elektronischen Signatur (vgl. [16, S. 8])

Eine *„Sichere Signaturerstellungseinheit“ (SSEE)* (§ 2 Nr. 10) ist eine Software- oder Hardwareeinheit zur Speicherung und Anwendung des jeweiligen Signaturschlüssels, die verschiedene Eigenschaften erfüllen muss (vgl. § 17 Abs. 1 SigG und § 15 Abs. 1 SigV), die gemäß § 18 SigG durch eine Prüfung nachgewiesen werden müssen. Softwareprodukte sind also per Gesetz nicht ausgeschlossen, jedoch handelt es sich bei allen heutigen SSEEs um Chipkarten (s. [18]).
„Signaturanwendungskomponenten“ (§ 2 Nr. 11) sind „Software- und Hardwareprodukte, die dazu bestimmt sind,

1. Daten dem Prozess der Erzeugung oder Prüfung qualifizierter elektronischer Signaturen zuzuführen oder

2. qualifizierte elektronische Signaturen zu prüfen oder qualifizierte Zertifikate nachzuprüfen und die Ergebnisse anzuzeigen.“

Ein Beispiel für eine Signaturanwendungskomponente ist ein E-Mailprogramm, das E-Mails mit einer qualifizierten Signatur versehen kann und diese auch verifizieren kann.
Des Weiteren sind *„Produkte für qualifizierte elektronische Signaturen“* ein Oberbegriff für SSEEs, Signaturanwendungskomponenten und technische Komponenten für Zertifizierungsdienste (§ 2 Nr. 13).
„Qualifizierte Zeitstempel“ (§ 2 Nr. 14) sind elektronische Bescheinigungen eines Zertifizierungsdiensteanbieter, dass ihm bestimmte Daten zu einem bestimmten Zeitpunkt vorgelegen haben. Hierbei muss der ZDA bestimmte Anforderungen erfüllen, auf die in Abschnitt 3.2.2 näher eingegangen wird.
Die (einfache) elektronische Signatur hat den geringsten Beweiswert, da an sie keinerlei Sicherheitsanforderungen gestellt werden, die die Signatur mit einer Person verknüpft. So handelt sich es bei einer eingescannten eigenhändigen Unterschrift bereits um eine elektronische Signatur.
Wurde eine fortgeschritten elektronische Signatur mit einem assymetrischen Verschlüsselungsverfahren erzeugt, so kann die Authentizität des öffentlichen Schlüssels mittels X.509-Zertifikaten

oder eines „Web of Trust“[1], wie es beispielsweise bei PGP[2] zum Einsatz kommt, sichergestellt werden. Für die Signatur können sowohl hardware- als auch softwarebasierende Verfahren zum Einsatz kommen.
Für qualifizierte elektronische Signaturen gilt, dass sie von sicheren Signaturerstellungseinheiten und qualifizierten Zertifikaten erzeugt werden müssen. Dies hat durch die erhöhten technischen Sicherheitsmaßnahmen der SSEE zur Folge, dass die Sicherheit des privaten Schlüssels gewährleistet ist und kaum noch vom Nutzerverhalten abhängt. Die für die Signaturerzeugung eingesetzten Smartcards lassen das Auslesen des privaten Schlüssels erst gar nicht zu.
Zusätzlich gibt es qualifizierte elektronische Signaturen mit Anbieterakkreditierung. Dabei werden die gleichen Anforderungen wie an eine qualifizierte Signatur gestellt, jedoch werden die Prozesse und die Infrastruktur des Zertifizierungsdiensteanbieters geprüft und von der Bundesnetzagentur[3] (als zuständiger Behörde) bestätigt (vgl. dazu Abschnitt 3.2.2).
Auf die Tatsache, dass die qualifizierte elektronische Signaturen der eigenhändigen Unterschrift rechtlich gleichgestellt ist, wird in [16, S. 14-19] näher eingegangen.

3.2.2 Zertifizierungsdiensteanbieter

Zertifizierungsdiensteanbieter, die qualifizierte Zertifikate oder Zeitstempel ausstellen, müssen gemäß SigG und SigV eine Reihe von Anforderungen erfüllen, um den letztendlich erzeugten elektronischen Signaturen eine hohe Beweiskraft zukommen zu lassen. Darunter fallen insbesondere (vlg. hierzu auch [16, S. 10f]):

- die Ausarbeitung eines Sicherheitskonzeptes (§ 4 Abs. 2 SigG und § 2 SigV)
- die erforderliche Zuverlässigkeit und Fachkunde des Personals (§ 4 Abs. 2 SigG und § 5 Abs. 3 SigV)
- die Antragssteller von qualifizierten Zertifikaten müssen zuverlässig identifiziert werden (§ 5 Abs. 1 Satz 1 SigG und § 3 Abs. 1 SigV)
- der Betrieb eines hochverfügbaren Verzeichnis- (§ 5 Abs. 1 Satz 2 SigG) und Sperrdienstes (§ 8 SigG und § 7 SigV)
- die Behandlung von Attributen (§ 5 Abs. 2 SigG, § 8 Abs. 2 SigG und § 3 Abs. 2 SigV) und Pseudonymen (§ 5 Abs. 3 SigG und § 14 Abs. 2 SigG)
- die Unterrichtungspflicht gegenüber Antragstellern (§ 6 SigG)
- die Dokumentation (§ 10 SigG und § 8 SigV)
- die Haftung (§ 11 SigG) und Deckungsvorsorge (§ 12 SigG und § 9 SigV)
- Datenschutzbestimmungen (§ 14 SigG)
- der Einsatz von geprüften Produkten für elektronische Signaturen mit Bestätigung nach § 18 SigG oder Herstellererklärung (§ 17 SigG, § 15 SigV und Anlage 1 SigV).

[1] Im Gegensatz zu herkömmlichen Public-Key-Infrastrukturen wird beim Web of Trust die Authentizität der öffentlichen Schlüssel nicht durch eine zentrale Zertifizierungsinstanz sondern durch gegenseitiges Vertrauen der Benutzer beglaubigt.

[2] Pretty Good Privacy (PGP) ist ein von Phil Zimmermann entwickeltes Programm, mit dem man Daten verschlüsseln und digital signieren kann.

[3] http://www.bundesnetzagentur.de

Spätestens bei Aufnahme des Betriebs des Zertifizierungsdienstes muss er bei der zuständigen Behörde (nach § 3 SigG ist dies die Bundesnetzagentur) angezeigt werden (§ 4 Abs. 3 SigG und § 1 SigV). Darüber hinaus kann sich er sich auf Antrag freiwillig akkreditieren lassen (§ 15 SigG).
Die freiwillige Akkreditierung ist ein Gütesiegel der Bundesnetzagentur für Zertifizierungsdiensteanbieter und ist mit weiteren Rechten und Pflichten verbunden. So ist beispielsweise

- das Sicherheitskonzept mindestens alle drei Jahre zu prüfen und zu bestätigen (§ 15 Abs. 2 SigG)
- nur der Einsatz von geprüften und bestätigten Produkten nach § 18 SigG vorgesehen
- ein qualifiziertes Zertifikat nur für Personen in Besitz einer geprüften und bestätigten sicheren Signaturerstellungseinheit auszustellen (§ 15 Abs. 7 Satz 2 SigG)
- der Signaturschlüssel-Inhaber über geprüfte und bestätigte Signaturanwendungskomponenten zu informieren (§ 15 Abs. 7 Satz 2 SigG)
- eine Aufbewahrungsdauer von 30 Jahren vorgesehen (s. dazu Abschnitt 3.2.5.1).

Neben der Bestätigung der nachgewiesenen Sicherheit durch die Bundesnetzagentur (§ 15 Abs. 1 SigG) hat die Akkreditierung noch weitere Vorteile. Die Bundesnetzagentur stellt die benötigten qualifizierten Zertifikate dem akkreditierten Zertifizierungsdiensteanbieter aus (§ 16 Abs. 1 SigG) und garantiert außerdem die langfristige Überprüfbarkeit der Zertifikate (§ 15 Abs. 6 SigG). Dies ist besonders für die Langzeitarchivierung elektronisch signierter Dokumente von Bedeutung (s. Abschnitt 3.2.5). Signaturen können also auch z. B. bei Insolvenz des Zertifizierungsdiensteanbieters 30 Jahre überprüft werden.

3.2.3 Signaturanwendungen

Zum Signieren von Daten braucht man in der Regel sowohl eine sichere Signaturerstellungseinheit als auch eine Signaturanwendungskomponente.
In § 17 SigG werden einige technische Anforderungen für genannte Produkte näher beschrieben. So müssen die eingesetzten SSEEs vor Verfälschungen der zu signierenden Daten oder der Signatur sowie vor der unberechtigten Nutzung geschützt werden (Abs. 1). § 15 Abs. 1 sieht hierfür eine Identifikation der Anwender vor, die in der Regel mit einer *Personal Identification Number (PIN)* realisiert wird.
In diesem Zusammenhang werden auch Anforderungen an Signaturanwendungskomponenten genannt (Abs 2.):

> „Für die Darstellung zu signierender Daten sind Signaturanwendungskomponenten erforderlich, die die Erzeugung einer qualifizierten elektronischen Signatur vorher eindeutig anzeigen und feststellen lassen, auf welche Daten sich die Signatur bezieht.
>
> (...)
>
> Signaturanwendungskomponenten müssen nach Bedarf auch den Inhalt der zu signierenden oder signierten Daten hinreichend erkennen lassen. Die Signaturschlüssel-Inhaber sollen solche Signaturanwendungskomponenten einsetzen oder andere geeignete Maßnahmen zur Sicherheit qualifizierter elektronischer Signaturen treffen.“

Bietet ein Zertifizierungsdiensteanbieter einen Zeitstempeldienst an, so muss nach Abs 3. Nr. 3 sichergestellt sein, dass „bei [der] Erzeugung qualifzierter Zeitstempel Fälschungen und Verfälschungen auszuschließen [sind].“
In § 15 Abs. 2-4. SigV werden schließlich diese Anforderungen weiter spezifiziert:

> „(2) Signaturanwendungskomponenten nach § 17 Abs. 2 des Signaturgesetzes müssen gewährleisten, dass
>
> 1. bei der Erzeugung einer qualifizierten elektronischen Signatur
>   (a) die Identifikationsdaten nicht preisgegeben und diese nur auf der jeweiligen sicheren Signaturerstellungseinheit gespeichert werden,
>   (b) eine Signatur nur durch die berechtigt signierende Person erfolgt,
>   (c) die Erzeugung einer Signatur vorher eindeutig angezeigt wird und
> 2. bei der Prüfung einer qualifizierten elektronischen Signatur
>   (a) die Korrektheit der Signatur zuverlässig geprüft und zutreffend angezeigt wird und
>   (b) eindeutig erkennbar wird, ob die nachgeprüften qualifizierten Zertifikate im jeweiligen Zertifikat-Verzeichnis zum angegebenen Zeitpunkt vorhanden und nicht gesperrt waren.
>
> (3) (...) [Es] muss gewährleistet sein, dass die zum Zeitpunkt der Erzeugung des qualifizierten Zeitstempels gültige gesetzliche Zeit unverfälscht in diesen aufgenommen wird.
>
> (4) Sicherheitstechnische Veränderungen an technischen Komponenten (...) müssen für den Nutzer erkennbar werden.“

Zertifizierungsdiensteanbieter sind dazu veranlasst, nur geprüfte Signaturanwendungskomponenten mit Bestätigung oder Herstellererklärung (vgl. dazu Abschnitt 3.2.2) einzusetzen. Auch wenn es Anwendern freigestellt ist, ob sie gleichermaßen geprüfte Signaturanwendungskomponenten benutzen, ändert dies nichts am Status der qualifizierten elektronischen Signatur (s. [16, S.13]). Besonders bei Massensignaturen (vgl. Abschnitt 3.2.4) ist es aufgrund der erhöhten Missbrauchsgefahr empfohlen, auf sichere Signaturanwendungskomponenten zurückzugreifen.
Es wird an dieser Stelle darauf hingewiesen, dass die Prüfung von Signaturen nach dem Kettenmodell zu erfolgen hat (s. auch Antwort auf Frage 14 in den FAQ der Bundesnetzagentur [17]).
Eine Übersicht von geprüften und bestätigten Signaturanwendungskomponenten findet sich unter [18].

3.2.4 Massensignatur

Während bei normalen Signaturanwendungen für jede qualifizierte Signatur eine Identifikation notwendig ist, versteht man unter der *Massensignatur* die automatisierte Erzeugung von qualifizierten Signaturen.
Massensignaturen werden vor allem dort eingesetzt, wo es zu unwirtschaftlich wäre, für jede Signatur den Schlüsselinhaber um Erlaubnis zu fragen. Typische Anwendungen sind Zeitstempeldienste, elektronische Rechnungsstellung oder die elektronische Archivierung von schriftlichen Unterlagen (vgl. [34]).

Im Signaturgesetz [6] werden automatisierte Signaturerstellungen an keiner Stelle erwähnt, da es hauptsächlich die Belange der Zertifizierungsdiensteanbieter regelt, jedoch nicht die Nutzung der Signaturschlüssel der Anwender (vgl. [16, S. 82]).
Diesbezüglich stehen die Anforderungen an eine Signaturanwendungskomponente (s. Abschnitt 3.2.3) in keinen Widerspruch zur automatischen Erzeugung elektronischer Signaturen (s. [45, S. 188-190]).
Erst die Begründung zur Signaturverordnung [5] nimmt zu § 15 Abs. 2 SigV Stellung in Bezug auf Massensignaturen:

> „Insbesondere bei der automatischen Erzeugung von Signaturen ('Massensignaturen') muss sichergestellt sein, dass Signaturen nur zu dem voreingestellten Zweck (z. B. Signaturen zu Zahlungsanweisungen bei Großanwendern) und durch eine zuvor geprüfte und abgenommene Anwendung vorgenommen werden können."

Nun bleibt zu klären, wie „geprüft und abgenommen" verstanden werden kann. Laut Auskunft auf die Frage an das Bundesamt für Sicherheit in der Informationstechnik (BSI)[4], ob für die Prüfung eines Zeitstempeldienstes eine Herstellererklärung ausreichend sei, muss der Zeitstempeldienst geprüft und durch eine Stelle nach § 18 bestätigt worden sein[5]. Weiterführende Erklärungen zu „geprüft und abgenommen" finden sich in [34].
Gerade bei der automatischen Signaturerstellung muss eine hohe Sicherheit gewährleistet sein, damit nach § 15 Abs. 2 SigV nur die berechtigte Person signieren kann. Die Begründung zur Signaturverordnung [5] schlägt z. B. vor, die Signaturerstellungseinheit nur für eine feste Anzahl von Signaturen oder nur für ein „Zeitfenster" freizuschalten.
Unter der Antwort auf Frage 18 in den FAQ der Bundesnetzagentur [17] findet sich außerdem Folgendes:

> „Trotz Verwendung dieser technischen Hilfsmittel werden die Erklärungen aus den signierten Dokumenten dem Absender persönlich zugerechnet. Daher sollte bei derartigen 'automatisch' erstellten Signaturen immer ein besonderer Schutz gegen Missbrauch implementiert werden. Dieser Schutz sollte sich an dem Aktivierungszeitraum orientieren, was von einem verschlossenen Stahlschrank für Karte und Kartenleser, bis hin zur TrustCenter Umgebung reichen kann."

In technischer Hinsicht entspricht ein Zeitstempeldienst einem Massensignatursystem, den ein Zertifizierungsdiensteanbieter bereitstellt. Dafür ist nach §§ 4-14 SigG ein Sicherheitskonzept zu erstellen (vgl. Abschnitt 3.2.2). Je nach Art der getroffenen Vorkehrungen muss entschieden werden, wie die Signaturkarten freigeschaltet werden können.
Es bleibt zu bedenken, dass ein qualifizierter Zeitstempel nicht mit einer qualifizierten Signatur gleichzusetzen ist. Ein qualifizierter Zeitstempel trägt zwar eine (qualifizierte) Signatur, ist aber keine qualifizierte Signatur (Willenserklärung des Signaturschlüsselinhabers) im Sinne des Gesetzes (vgl. [31]).

---

[4] `http://www.bsi.de`

[5] Ingo Hahlen vom BSI schrieb am 26. 09. 2006: „Mit 'Geprüft und abgenommen' ist eine Bestätigung nach §18 SigG gemeint. Die Herstellererklärung kommt wohl in der Begründung an dieser Stelle nicht vor, wobei seitens der Bundesnetzagentur aber schon gewisse Ansprüche an die Herstellererklärung gestellt werden. Die Prüfung muss den Zeitstempeldienst umfassen, wobei die zur Signatur der Zeitstempel verwendeten Karten geprüfte und bestätigte SSCDs sein müssen, die in einer Auflagen-konformen Betriebsumgebung eingesetzt werden müssen."

3.2.5 Langzeitarchivierung elektronisch signierter Dokumente

In den meisten Anwendungsgebieten müssen Dokumente unterschiedlich lange aufbewahrt werden. Dies gilt also auch für elektronisch signierte Dokumente.
Damit die Signatur auch nach längerer Zeit erfolgreich überprüft werden kann, ergeben sich einerseits die Problematik der Archivierung der Dokumente an sich und andererseits die Problematik, die mit der Sicherheitseignung der kryptographischen Algorithmen einhergeht. Auf den letzten Punkt und die damit verbundenen Maßnahmen wird im Folgenden näher eingegangen.

3.2.5.1 Aufbewahrungsdauer

Gemäß § 14 SigV beträgt die Gültigkeitsdauer eines qualifizierten Zertifikats höchstens fünf Jahre. Nach § 4 SigV muss es nach dem angegebenen Gültigkeitszeitraum noch mindestens fünf weitere Jahre in einem Zertifikatsverzeichnis geführt werden. Handelt es sich um ein qualifiziertes Zertifikat mit Anbieterakkreditierung, so muss es noch mindestens 30 Jahre abrufbereit gehalten werden, was übrigens von der Bundesnetzagentur garantiert wird (s. Abschnitt 3.2.2).
Da die rechtliche Aufbewahrungsdauer von Dokumenten durchaus 30 Jahre oder mehr betragen kann, bieten qualifizierte Signaturen mit Anbieterakkreditierung die höchste Rechtssicherheit, da außerdem nur für sie die „Sicherheitsvermutung des Signaturgesetzes“ gilt (vgl. [51, S. 22f] und [1] zur Begründung von § 15 SigG).

3.2.5.2 Archivierung

In [49] wird auf die Problemstellungen von langfristigen Archivierungen von Dokumenten eingegangen. Demnach sind u. a. folgende Punkte zu beachten:

1. Sichere Speicherung: Damit ist die Lebensdauer der Datenträger gemeint, auf denen die elektronischen Dokumente gespeichert werden. Da nicht garantiert werden kann, dass die Speichermedien die Daten dauerhaft fehlerfrei speichern, muss also gegebenenfalls überprüft werden, ob sie auf neue Medien abgebildet werden müssen.

2. Interpretierbarkeit: Hierbei geht es um die langfristige Lesbarkeit von den verwendeten Speichermedien und Datenformaten. Einerseits muss sichergestellt sein, dass man das Medium langfristig lesen kann und dass das Format eine Präsentation der Originals zulässt. In [51, S. 77-79] findet sich eine Bewertung von geeigneten (Signatur-) Datenformaten.

3. Authentizität und Integrität: Bei elektronischen Dokumenten muss gewährleistet werden, dass die archivierten Daten nicht verändert wurden und dass ihr Urheber zweifelsfrei nachgewiesen werden kann. Siehe dazu auch Abschnitt 3.2.5.4.

4. Referenzierbarkeit / Suche: Nachdem ein Dokument archiviert wurde, muss es auch wieder auffindbar sein.

3.2.5.3 Sicherheitseignung kryptographischer Algorithmen

Da die digitale Signatur auf Hashfunktionen und asymmetrischen Signaturverfahren aufbaut, muss gewährleistet sein, dass die verwendeten Algorithmen auch sicher sind (vgl. hierzu Abschnitt 2.1).
In der Anlage 1 I.2 zur SigV wird aus diesem Grund die Sicherheitseignung näher spezifiziert. Die Bundesnetzagentur hat jährlich in Zusammenarbeit mit einem Expertengremium eine Liste

mit geeigneten Algorithmen und ihrer dazugehörigen Parameter zu bestimmen. Demnach ist die Eignung für einen Zeitraum von mindestens sechs Jahren abzuschätzen und gegeben, wenn „innerhalb des bestimmten Zeitraumes nach dem Stand von Wissenschaft und Technik eine nicht feststellbare Fälschung von qualifizierten elektronischen Signaturen oder Verfälschung von signierten Daten mit an Sicherheit grenzender Wahrscheinlichkeit ausgeschlossen werden kann."
Außerdem gilt § 6 SigG, nach dem Zertifizierungsdienstleister Antragssteller darauf hinzuweisen hat, dass „Daten mit einer qualifizierten elektronischen Signatur bei Bedarf neu zu signieren sind, bevor der Sicherheitswert der vorhandenen Signatur durch Zeitablauf geringer wird."
Eine Übersicht geeigneter kryptographischer Algorithmen findet sich in [19].

3.2.5.4 Neusignierung

Vor dem Hintergrund der Sicherheitseignung der kryptographischen Algorithmen müssen Daten mit einer qualifizierten elektronischen Signatur nach § 17 SigV vor Ablauf der Eignung *übersigniert*, d. h. neu signiert werden. Die neue (qualifizierte) Signatur muss „mit geeigneten neuen Algorithmen oder zugehörigen Parametern erfolgen, frühere Signaturen einschließen und einen qualifizierten Zeitstempel tragen."
Im Rahmen des „ArchiSig"-Projekts[6] wurde diese Regelung folgendermaßen ausgelegt (vgl. [51, S. 28-31]):

1. Soll der Beweiswert von qualifizierten elektronischen Signaturen erhalten werden, ist eine erneute qualifizierte Signatur mit einem qualifizierten Zeitstempel, wie in § 17 SigV vorgesehen, notwendig. Es ist nicht ausreichend, die Daten auf einmal beschreibbaren Datenträgern zu speichern oder z. B. bei einem Notar zu hinterlegen.

2. Ziel und Zweck der erneuten Signatur ist die Sicherstellung von Integrität und Authentizität eines bereits signierten Dokuments, es handelt sich also um keine Willenserklärung, die die persönliche Signatur z. B. eines Archivars erforderlich machen würde.

3. Für die Neusignierung der Daten ist ein qualifizierter Zeitstempel ausreichend, sofern er eine qualifizierte elektronische Signatur enthält. Eine weitere qualifizierte Signatur (im Sinne einer Willenserklärung) bietet keinen Mehrwert an Sicherheit.

4. Sofern nur das asymmetrische Verschlüsselungsverfahren in seiner Sicherheitseignung gefährdet ist, ist es ausreichend, „allein die Signaturen des elektronischen Dokuments erneut mit einem Zeitstempel zu versehen und damit neu zu signieren" [51, S. 30]. Ist allerdings (auch) der eingesetzte Hashalgorithmus nicht mehr geeignet, so ist die „Berechnung eines neuen Hashwerts der gesamten Daten mit einem neuen sicherheitsgeeigneten Hashalgorithmus und ein erneuter Zeitstempel unter Einbeziehung einer erneuten qualifizierten elektronischen Signatur [notwendig]." [51, S. 30]

5. Die erneute Signatur muss *vor* Ablaufen der Sicherheitseignung der verwendeten Algorithmen und ihrer Parameter mit neuen, sicherheitsgeeigneten Algorithmen erfolgen.

[6] Im Projekt „ArchiSig" werden Archivierungskonzepte und -technologien aufgegriffen und dahingehend erweitert, dass sie die sichere und beweiskräftige Langzeitarchivierung digital erzeugter und signierter Daten über 30 Jahre und mehr ermöglichen.
http://www.archisig.de

6. „Die erneute elektronische Signatur muss mindestens die gleiche Qualitätsstufe haben wie die Ausgangssignatur, um deren Qualität zu erhalten.“ [51, S. 30] Eine qualifizierte elektronische Signatur mit Anbieterakkreditierung muss also mit einem qualifizierten Zeitstempel eines akkreditierten Zertifizierungsdiensteanbieters erneuert werden. Handelt es sich um eine qualifizierte elektronische Signatur, so bleibt es freigestellt, ob der Zeitstempel von einem akkreditierten Zertifizierungsdiensteanbieter ausgestellt wird oder nicht.

7. Die erneute Signatur muss alle vorherigen Signaturen (auch Mehrfachsignaturen oder bereits erneuerte Signaturen) umfassen. Auf diese Weise bleibt die rechtliche Beweiskraft erhalten und nimmt sogar zu, da ein nachträgliches Löschen einer zu den Daten gehörende Signatur erkennbar wird.

8. Die erneute elektronische Signatur kann auch mehrere signierte Dokumente umschließen.

3.2.5.5 Verifikationsdaten

Neben der Neusignierung gibt es außerdem noch den Aspekt der Signaturprüfung zu beachten. Zum einen wird bei der Verifikation einer Signatur die mathematische Korrektheit überprüft, zum anderen aber auch das verwendete Nutzerzertifikat.
Da eine dauerhafte Signaturprüfung nicht ohne weiteres möglich ist (vgl. Abschnitt 3.2.5.1), gilt es zu bedenken, inwieweit man den Beweiswert der Signatur sicherstellen kann, da es durchaus an klaren Vorgaben fehlt. Einerseits stellt sich die Frage, welche Daten denn für den Nachweis der Authentizität benötigt werden und andererseits, wie ein dauerhafter Zugriff darauf gewährleistet werden kann.
Der Artikel von Tielemann et al. in [51] gibt auf diese Frage Anhaltspunkte, wie man bei der Bestimmung von Verifikationsdaten vorgehen kann. Als Verifikationsdaten können beispielsweise herangezogen werden:

1. Zertifikatsketten, um die Echtheit der verwendeten Zertifikate zu belegen. Darunter fallen hauptsächlich Zertifikate von Anwendern, Zertifizierungsstellen, Zeitstempel- und Verzeichnisdiensten sowie Wurzelinstanzen (bei qualifizierten Zertifikaten ist dies die Bundesnetzagentur).

2. Nachweise der Gültigkeit der Zertifikate zum Prüfzeitpunkt, also z. B. OCSP-Antworten oder CRLs. Man beachte jedoch, dass nur OCSP-Abfragen einen Nachweis der Gültigkeit des Zertifikats nach SigG ermöglichen.[7]

3. Zeitstempel, sofern vorhanden.

4. Erneute Signaturen mit ihren Zertifikatsketten und den dazugehörigen Gültigkeitsabfragen.

Bei qualifizierten Signaturen mit (deutscher) Anbieterakkreditierung wird davon ausgegangen, dass folgende Verifikationsdaten für eine erfolgreiche und beweiskräftige Signaturprüfung ausreichend sind:

[7] „Nach § 5 Abs. 2 SigV i. V. m. § 5 Abs. 1 Satz 3 SigG darf ein Zertifikat vom Zertifizierungsdiensteanbieter erst dann in ein Zertifikatsverzeichnis aufgenommen werden, wenn der Signaturschlüsselinhaber den Erhalt der sicheren Signaturerstellungseinheit gegenüber dem Zertifizierungsdiensteanbieter bestätigt hat. Sperrlisten führen jedoch nur gesperrte Zertifikate auf, eine Aussage über die Gültigkeit von Zertifikaten ermöglichen sie damit grundsätzlich nicht.“ [51, S. 99]

1. Anwender- und Attributzertifikate mit Zertifikatskette bis zur Wurzelinstanz der Bundesnetzagentur

2. qualifizierter Zeitstempel eines akkreditierten Zertifizierungsdiensteanbieters mit Zertifikatskette bis zur Wurzelinstanz der Bundesnetzagentur

3. gültige OCSP-Antworten (akkreditiert signiert) mit Zertifikatskette bis zur Wurzelinstanz der Bundesnetzagentur

4. gültige OCSP-Antworten (akkreditiert signiert) oder CRLs auf Attributzertifikate mit Zertifikatskette bis zur Wurzelinstanz der Bundesnetzagentur.

Für einfach qualifizierte Signaturen lässt sich derzeit (Anfang 2007) noch keine pauschale Aussage für die benötigten Verifikationsdaten machen.

3.2.6 Technische Umsetzung

Im Maßnahmenkatalog für digitale Signaturen [50] werden technische Anforderungen und Maßnahmen für Zertifizierungsstellen und technische Komponenten beschrieben, um als praktische Hilfe für die Umsetzung des alten Signaturgesetzes [2] und der dazugehörigen Signaturverordnung [3] zu dienen.
Aufgrund des empfehlenden Charakters des Maßnahmenkatalogs müssen die genannten Maßnahmen nicht unbedingt erfüllt werden, geben aber einige Ansätze, wie sie umgesetzt werden können.
Ein Abschnitt darin beschreibt ebenfalls einen Zeitstempeldienst, dessen Schutzbedarf als sehr hoch anzusehen ist (vgl. [50, S. 13]). In diesem Zusammenhang werden folgende Sicherheitsziele genannt:

- Vertraulichkeit. Der private Schlüssel eines Zeitstempeldienstes muss besonders geschützt werden, da er „uneingeschränkt vertraulich“ ist.

- Integrität. Der Zeitstempeldienst gibt eine verbindliche Auskunft zu der Existenz eines Dokuments und muss daher auch uneingeschränkt nachprüfbar sein.

- Verfügbarkeit. Die zu signierenden Dokumente können zeitkritisch sein, wenn z. B. Fristen eingehalten werden müssen.

Für eine Liste der genauen Anforderungen und die daraus abgeleiteten Maßnahmen sei auf genannten Maßnahmenkatalog verwiesen. Folgende Themen werden darin behandelt:

- Zugriffsschutz (z. B. Betriebssystem, Zugangswege, ...)

- Maßnahmen zum Schutz der Zeit (Verwendung einer Funk- und Referenzuhr)

- Maßnahmen zur Zeistempel-Sicherheit (z. B. Sicherheit des Signaturschlüssels, Protokollierung, ...)

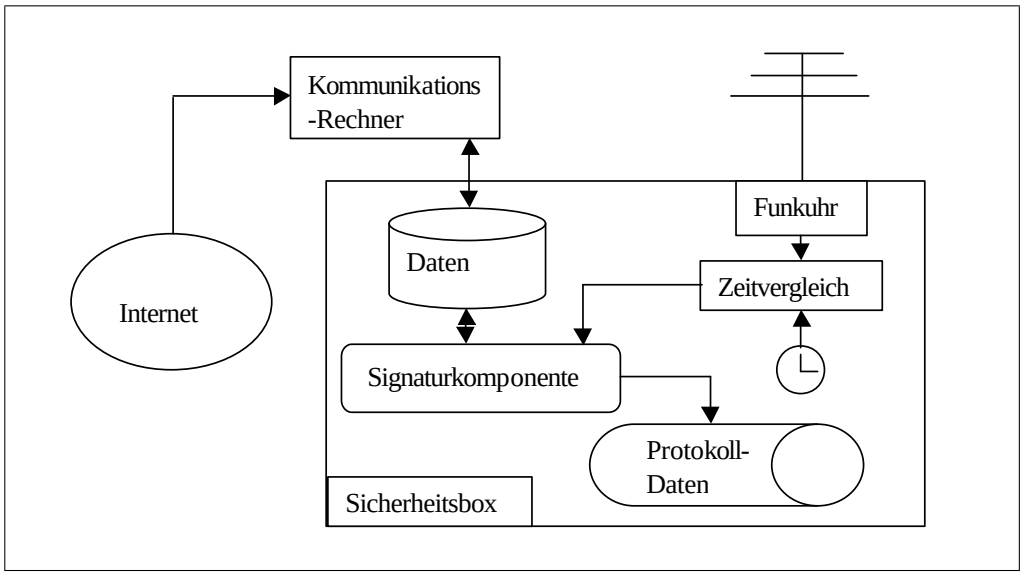

Abbildung 3.3: Komponenten eines Zeitstempeldienstes [50, S. 163]

3.2.6.1 Lösungsvorschläge

Im Maßnahmenkatalog [50] wurden ebenfalls Lösungsvorschläge für einen Zeitstempeldienst erarbeitet, der die entsprechenden Maßnahmen aus den vorigen Abschnitten umsetzt und in Abbildung 3.3 dargestellt ist:
„Der Zeitpunkt (Datum, Stunde, Minute, ggf. Sekunde und Zeitzone) wird durch einen Funkempfänger (DCF77, 77kHz) festgestellt und automatisch zusammen mit den zu stempelnden Daten signiert. Die Form der Daten ist für den Zeitstempeldienst irrelevant. Es kann sich um Signaturen, Hashwerte oder vollständige Dateien handeln. Parallel wird durch eine lokale IT-gestützte Referenzuhr die Korrektheit der empfangenen Zeit überprüft. Alternativ kann zur Bestimmung der Referenzzeit auch ein GPS-Empfänger o.ä. benutzt werden. Wenn zwischen der Funkuhr und der Referenzuhr eine Abweichung festgestellt wird, wird der Zeitstempeldienst abgeschaltet und eine Warnmeldung erzeugt.
Die Verwendung eines sekundengenauen Zeitstempels ist nur möglich, wenn die technisch bedingten Abweichungen zwischen dem Funkempfänger und der Referenzuhr im Bereich von Millisekunden liegen. Generell darf die zulässige Abweichung zwischen den Uhren maximal die Hälfte der vom Zeitstempeldienst-Betreiber garantierten Antwortzeit liegen. Diese Antwortzeit sollte nicht mehr als eine Minute betragen.
Der Zeitstempeldienst ist eigener Teilnehmer der zugeordneten ZS[8]. Hierdurch wird der Schaden, der bei einer Kompromittierung des Zeitstempel-Signaturschlüssels entsteht, auf eine ZS begrenzt. Das Zertifikat des Zeitstempeldienstes wird im Verzeichnisdienst abrufbar gehalten. Auch im Falle, daß er nur organisatorisch zur ZS gehört (im Rahmen der Pflichtdienstleistung), wird ein eigener Signaturschlüssel verwendet, der von der zugehörigen ZS zertifiziert wird.
Um einen Zeitstempel zu bekommen, verschickt der Benutzer seine Daten über eine öffentlich zugängliche Telekommunikationseinrichtung an einen Kommunikations-Rechner beim Zeitstempel-Anbieter. Hierbei kann es sich z. B. um einen HTTP-Proxy mit stark eingeschränkter Funktionalität handeln. Der Telekommunikations-Zugang muß so abgesichert sein, daß keine Angriffe auf den Kommunikations-Rechner möglich sind. Der benutzte Protokoll-Stack muß dann mit

[8]Zertifizierungsstelle (ZS). Gemeint ist eine Zertifizierungsinstanz.

entsprechenden Packet-Filter Eigenschaften versehen werden, die keine zusätzliche Nutzung ermöglichen.

Der Kommunikations-Rechner ist zusätzlich über ein gesichertes Protokoll mit einer speziellen Sicherheitsbox verbunden. Hierbei kann es sich z. B. um ein Store-and-Forward Verfahren handeln, bei dem der Kommunikations-Rechner die empfangenen Daten auf einer Festplatte ablegt, von wo aus sie in regelmäßigen Abständen von der Zeitstempel-Sicherheitsbox abgeholt werden. (...) Darüber hinaus kann die Sicherheitsbox den aktuellen Zeitpunkt feststellen und diesen mit der Referenzzeit vergleichen. Änderungen am Funkempfänger oder der Referenzuhr werden als Manipulationen der Sicherheitsbox betrachtet. Dieser Zeitpunkt zusammen mit den übertragenden Daten und einer laufenden Nummer werden in der Box signiert. Anschließend wird das Ergebnis über den Kommunikations-Rechner an den Benutzer zurückgegeben. Dieser muß überprüfen, ob die mit einem Zeitstempel versehenen Daten mit den verschickten übereinstimmen, ob der Zeitstempel korrekt ist und ob die vom Zeitstempeldienst verwendete Zeit plausibel ist.

Um im Verdachts- oder Manipulationsfall die Überprüfung eines Zeitstempels zu ermöglichen, wird jede Nutzung des Zeitstempeldienstes auf einem einmal beschreibbarem Medium protokolliert, und es können zusätzlich verkettete Listen der erzeugten Zeitstempel verwendet werden. Der Zeitstempel-Signaturschlüssel muß mindestens einmal jährlich gewechselt werden.

Der Zugang zu dem Zeitstempel-Rechner ist nur für vertrauenswürdiges Personal nach entsprechender Identifizierung und Authentisierung möglich.

Bei der Prüfung eines Zeitstempels muß geprüft werden, ob der verwendete Zeitstempel- Signaturschlüssel zum Zeitpunkt der Prüfung nicht gesperrt ist.“ [50, S. 162f]

3.3 Standards

Nachdem im vorigem Abschnitt die gesetzlichen Rahmenbedingungen von Zeitstempeldiensten erklärt wurden, wird in diesem Abschnitt die technische Umsetzung näher beschrieben.

Ausschlaggebend ist das Time Stamp Protocol (s. Abschnitt 3.3.1) und der Cryptographic Message Syntax (s. Abschnitt 3.3.2), die das Format der Zeitstempel spezifizieren.

Von weiterer Bedeutung ist das Time Stamping Profile (s. Abschnitt 3.3.3) sowie die ISIS-MTT Spezifikation (s. Abschnitt 3.3.4), die bereits genannte Standards profilieren.

Schließlich wird in Abschnitt 3.3.5 auf Richtlinien für Zeitstempeldienste eingegangen. Dabei geht es hauptsächlich um die Gewährleistung des Beweiswertes von ausgestellten Zeitstempeln.

3.3.1 Time Stamp Protocol

In RFC 3161 [9] wird das *Time Stamp Protocol (TSP)* beschrieben, das von allen Zertifizierungsdiensteanbietern für die Ausstellung von qualifizierten Zeitstempeln verwendet wird (vgl. [16]).

Das TSP beschreibt die Formate von der Anfrage an eine TSA und deren Antwort sowie einige sicherheitstechnische Anforderungen an einen Zeitstempeldienst. Im Folgenden wird ein Blick auf die einzelnen Datenstrukturen geworfen, die allesamt ASN.1[9] kodiert werden.

[9] Die Abstract Syntax Notation One (ASN.1) erlaubt es, Datenstrukturen zu definieren und sie einheitlich in ein Transportformat zu kodieren.

3.3.1.1 TimeStampReq

Abbildung 3.4 gibt einen Überblick über die Daten, die während einer Zeitstempelanfrage (`TimeStampReq`) an den Server geschickt werden:

- `version`
  beschreibt die Version der TimeStampReq-Datenstruktur
- `MessageImprint`
  enthält den Hashwert der Daten (`hashedMessage`), die mit einem Zeitstempel versehen werden sollen. In `hashAlgorithm` steht die benutzte Hashfunktion.
- `reqPolicy`
  gibt die optionale Policy an, unter der der Zeitstempel ausgestellt werden soll.
- `nonce`
  enthält eine optionale Zufallszahl, mit der überprüft werden kann, dass der erzeugte Zeitstempel auch zur Anfrage gehört. Dies ist wichtig, wenn der Client keine zuverlässige Zeitquelle hat.
- `certReq`
  ist das Flag `certReq` gesetzt, so muss das Signaturzertifikat des Zeitstempeldienstes in der Antwort enthalten sein.
- `extensions`
  in diesem Feld könnten weitere Informationen der Anfrage mitgegeben werden. Es werden aber keine näheren Angaben dazu gemacht.

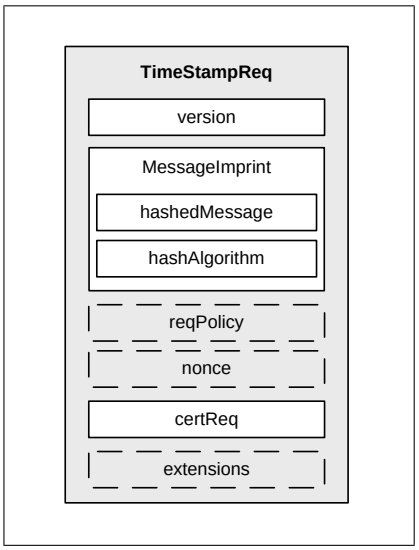

Abbildung 3.4: Zeitstempel-Anfrage (vgl. [16, S. 60])

3.3.1.2 TimeStampResp

Die Zeitstempelantwort (`TimeStampResp`) wird, wie in Abbildung 3.5 angedeutet, vom Server zurückgegeben. Sie besteht immer aus einer Statusangabe und kann durch Fehlerinformationen ergänzt werden (s. auch RFC 2510 [10]). Bei erfolgreicher Antwort wird des Weiteren

auch der gewünschte Zeitstempel (`timeStampToken`) mitgeliefert. Er enthält die signierten Zeitstempeldaten (`TSTInfo`) in einer `SignedData`- und `SignerInfo`-Struktur, die im Cryptographic Message Syntax (s. Abschnitt 3.3.2) beschrieben werden.

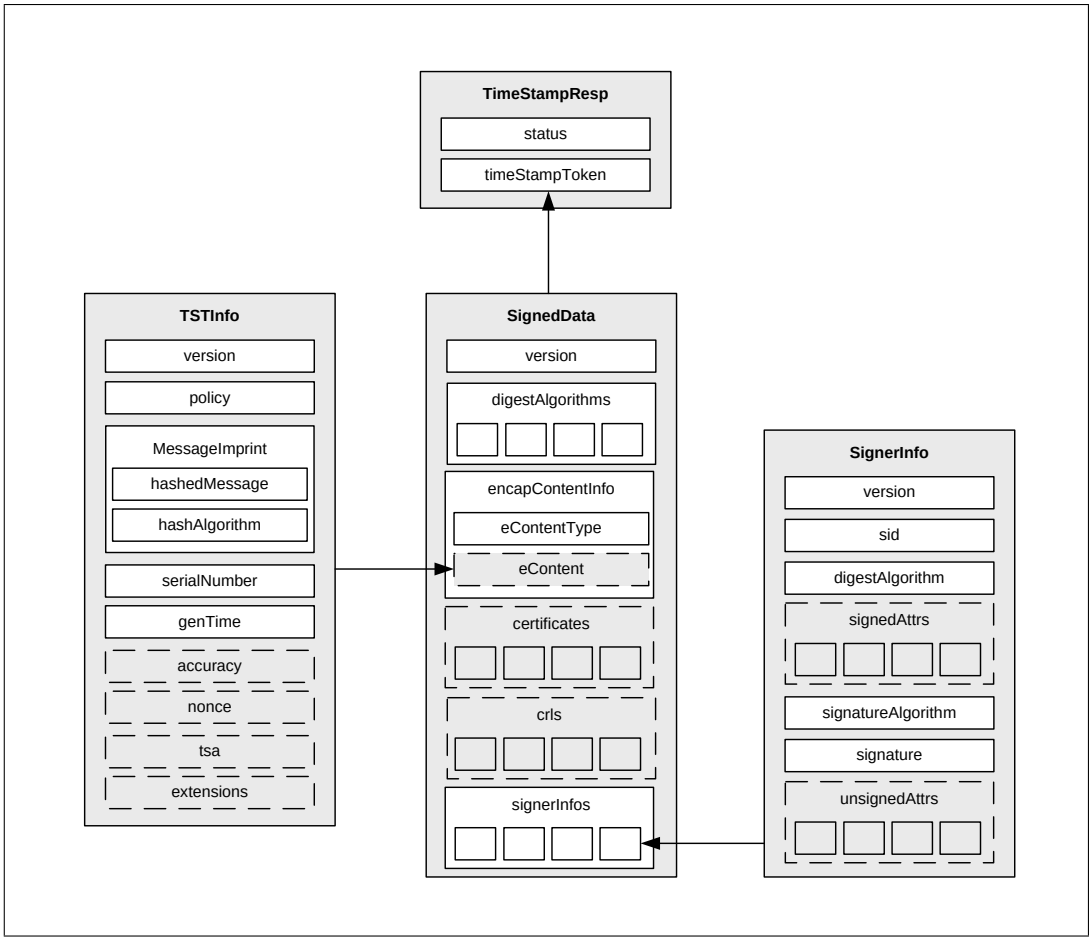

Abbildung 3.5: Zeitstempel-Antwort (vgl. [16, S. 61,70])

3.3.1.3 TSTInfo

Die `TSTInfo`-Struktur schließt folgende Informationen ein:

- `version`
  beschreibt die Version der `TSTInfo`-Struktur
- `policy`
  gibt die Policy an, unter der der Zeitstempel ausgestellt wurde.
- `MessageImprint`
  enthält den Hashwert der Daten (`hashedMessage`), die mit einem Zeitstempel versehen wurden. In `hashAlgorithm` steht die benutzte Hashfunktion.
- `serial`
  gibt die eindeutige Seriennummer des Zeitstempels an.

- `genTime`
  enthält den Zeitpunkt, mit dem die Daten verknüpft wurden.
- `accuracy`
  gibt optional die Genauigkeit der Zeitquelle in Sekunden, Millisekunden und Mikrosekunden an.
- `ordering`
  gibt an, ob die Zeitstempel unabhängig von der Genauigkeit des Zeitgebers geordnet werden können. Standardmäßig wird dieses Feld ignoriert.
- `nonce`
  enthält die Zufallszahl, die eventuell bei einer Anfrage mitgegeben wurde.
- `tsa`
  in diesem Feld steht optional der Name des Zeitstempeldienstes, der mit dem Signaturzertifikat übereinstimmen muss.
- `extensions`
  in diesem Feld könnten weitere Informationen der Anfrage mitgegeben werden. Es werden aber keine näheren Angaben dazu gemacht.

3.3.1.4 Überprüfung des Zeitstempels

Sendet ein Client eine Zeitstempel-Anfrage an einen Zeitstempeldienst, so muss er die Antwort auf seine Korrektheit überprüfen. War die Anfrage erfolgreich (ersichtlich am `status`-Flag), so müssen anschließend alle anderen relevanten Felder verifiziert werden.
Auf jeden Fall muss geprüft werden, dass der zurückgegebene Zeitstempel auch zu der Anfrage passt und dass er auch vom gewünschten Zeitstempeldienst mit einer akzeptablen Policy ausgestellt wurde. So müssen z. B. die Hashwerte der Daten in `hashedMessage` übereinstimmen. Verfügt der Client über eine zuverlässige Zeitquelle, so kann sie mit der Zeitangabe im Zeitstempel verglichen werden. Ansonsten muss die bei der Anfrage erzeugte Zufallszahl `nonce` mit der Antwort übereinstimmen. Außerdem muss die Signatur des Zeitstempels verifiziert werden (s. Abschnitt 3.3.2.3).

3.3.1.5 Transport und Authentifizierung

Das Time Stamp Protocol [9] schlägt mehrere Transportwege vor, um Zeitstempel zu beantworten und zu übertragen. Folgende Möglichkeiten werden genannt:

- E-Mail
- Dateibasiert
- Sockets
- HTTP

Da es insbesondere bei mandantenfähigen Zeitstempeldiensten gewünscht ist, die Clients zu identifizieren, muss auf andere Protokolle zurückgegriffen werden, weil die Zeitstempel-Anfrage (`TimeStampReq`) keine Authentifizierungsinformationen enthalten.
Es wird daher vorgeschlagen, Client-Authentifikation mit CMS (s. Abschnitt 3.3.2 und Signatur-Interoperabilitätsspezifikation [15]) oder TLS [23] durchzuführen.

3.3.2 Cryptographic Message Syntax

Im für das Time Stamp Protocol (s. Abschnitt 3.3.1) relevanten Cryptographic Message Syntax (CMS) [36] werden verschiedene weitverbreitete Signaturformate spezifiziert. Er wurde vom PKCS#7 Standard [52] abgeleitet und inzwischen in der neusten Fassung [37] veröffentlicht.

3.3.2.1 SignedData und SignerInfo

Für den Zeitstempeldienst sind vor allem die Datenstrukturen `SignedData` und `SignerInfo` (s. Abbildung 3.5) von Interesse, die im Folgenden genauer beschrieben werden.
Die SignedData-Struktur besteht aus folgenden Feldern:

- `version`
  Die Versionsnummer der benutzten Struktur um die Kompatibilität zwischen den verschiedenen CMS-Versionen zu gewährleisten.
- `digestAlgorithms`
  enthält alle für die Signatur verwendeten Hashfunktionen um die Überprüfung der Signatur zu erleichtern.
- `encapContentInfo`
  enthält die Felder `eContentType` und `eContent`, die den Inhalt näher spezifizieren, auf den sich die Signatur bezieht. Bei einem Zeitstempel ist der Inhalt die ASN.1-kodierte `TSTInfo`-Struktur.
- `certificates`
  kann Zertifikatsketten enthalten, die für die Signaturprüfung verwendet werden können
- `crls`
  kann Sperrinformationen (CRLs oder OCSP-Antworten) für die Zertifikate enthalten
- `signerInfos`
  enthält eine Reihe von Signaturen vom Typ `SignerInfo`, die aus folgenden Feldern zusammengesetzt ist:
  - `version`
    spezifiziert die Version der `SignerInfo`-Struktur.
  - `sid`
    enthält Daten, um das Signaturzertifikat eindeutig zu identifizieren, beispielsweise über den Issuer und die Seriennummer des Zertifikats.
  - `digestAlgorithm`
    gibt an, welche Hashfunktion zum Erzeugen der Signatur verwendet wurde.
  - `signedAttrs`
    enthält verschiedene Attribute, die in die Signatur einfließen. Siehe dazu Abschnitt 3.3.2.2 für eine Liste der verwendeten Attribute.
  - `signatureAlgorithm`
    gibt den verwendeten Signaturalgorithmus an.
  - `signature`
    enthält den Signaturwert.

  - `unsignedAttrs`
    kann weitere Attribute enthalten, die allerdings keinen Einfluss auf die Signatur haben.

3.3.2.2 Attribute

In einer Zeitstempel-Antwort fließen mehrere Attribute in die Signatur ein. Diese sollen hier näher besprochen werden:

- `ContentType`
  Das `ContentType`-Attribut gibt den Typ der signierten Daten in der `SignedData`-Struktur an.
- `MessageDigest`
  Das `MessageDigest`-Attribut enthält den Hashwert der zu signierenden Daten aus dem Feld `eContent`.
- `SigningCertificate`
  Mit dem `SigningCertificate`-Attribut aus RFC 2634 [35] kann zusätzlich der SHA-1-Hashwert der Zertifikatskette des Signaturzertifikates mitsigniert werden, um den späteren Austausch der Zertifikate zu verhindern.
- `OtherSigningCertificate`
  Das `OtherSigningCertificate`-Attribut aus [25] ist identisch mit dem `SigningCertificate`-Attribut, nur kann man zusätzlich andere Hashverfahren verwenden.

3.3.2.3 Erzeugen und Überprüfen von Signaturen

Wie bereits oben erwähnt, befindet sich die eigentliche Signatur in der `SignerInfo`-Struktur und wird in zwei Schritten erzeugt:

1. Berechnen des Hashwertes
   Falls keine signierten Attribute vorhanden sind, so wird der Inhalt von `eContent` gehasht. Ansonsten wird von allen ASN.1-kodierten signierten Attributen der Hashwert berechnet. Außerdem müssen in diesem Fall mindestens die Attribute `ContentType` und `MessageDigest` vorhanden sein.
2. Signieren
   Der berechnete Hashwert wird anschließend mit dem privaten Schlüssel des Signierers signiert und in das `signature`-Feld geschrieben.

Für die Verifikation der Signatur wird analog dazu vorgegangen. Zusätzlich muss die Zertifikatskette auf ihre Gültigkeit überprüft werden.

3.3.3 Time Stamping Profile

Das vom European Telecommunications Standards Institute (ETSI)[10] entworfene *Time Stamping Profile* [26] basiert auf dem Time Stamp Protocol (s. Abschnitt 3.3.1) und beschreibt, welche Anforderungen ein Server und ein Client in Bezug auf das Anfordern und Erzeugen von Zeitstempeln erfüllen muss.

[10] `http://www.etsi.org`

3.3.3.1 Anforderungen an einen Timestamping-Client

In Bezug auf eine Zeitstempel-Anfrage sollte der Client die Hashalgorithmen SHA-1 oder RIPEMD-160 benutzen.
Bei der Zeitstempel-Antwort müssen die Felder `accuracy` und `nonce` unterstützt werden. Erweiterungen können ignoriert werden. Als Signaturalgorithmus muss SHA1withRSA mit einer Schlüssellänge von mindestens 1024 bit unterstützt werden.

3.3.3.2 Anforderungen an den Timestamping-Server

Bei der Anfrage müssen auf der Seite des Servers jeweils die Felder `nonce` und `certReq` unterstützt werden. Erweiterungen können ignoriert werden. Ebenfalls muss der Server die Hashalgorithmen SHA-1, RIPEMD-160 und MD5 erkennen.
Beim Erzeugen des Zeitstempels werden folgende Anforderungen gestellt:

- Die Zeitangabe `genTime` wird auf eine Genauigkeit von einer Sekunde beschränkt.
- Die Genauigkeit `accuracy` muss mindestens eine Sekunde betragen.
- Das Feld `ordering` muss fehlen oder auf `false` gesetzt werden.
- Es müssen keine Erweiterungen generiert werden. Falls doch, dann dürfen sie nicht kritisch sein.

Der Name der Timestamping-Servers sollte aus einer Untermenge von folgenden Attributen bestehen:

- `countryName`
  identifiziert das Land, in dem der Zeitstempeldienst angemeldet ist.
- `stateOrProvinceName`
  identifiziert den Staat oder die Provinz des Zeitstempeldienstes (optional).
- `organizationName`
  gibt den offiziellen Namen des Zeitstempeldienstes an.
- `commonName`
  gibt den Namen der Zeitstempeleinheit (Signaturzertifikat) an.

Für die Signatur sollen sowohl die Hashalgorithmen SHA-1, RIPEMD-160 und MD5 als auch das Signaturverfahren SHA1withRSA unterstützt werden.
Als Transportprotokolle müssen sowohl ein Online- und ein Store-and-Forward[11]-Verfahren eingesetzt werden. Ein Transport über HTTP sollte, wie in RFC 3161 [9] beschrieben, unterstützt werden.

[11] Mit Store-and- Forward bezeichnet man eine Vermittlungstechnik, bei der die empfangenen Daten zwischengespeichert werden und nur bei Integrität an den Empfänger weitergeleitet werden.

3.3.4 ISIS-MTT Spezifikation

Die ISIS-MTT Spezifikation [60] beschäftigt sich mit allen relevanten elektronischen Signaturen bis hin zur qualifizierten elektronischen Signatur. Außerdem wird versucht, Kompatibilität und Interoperabilität zu den internationalen Standards zu erreichen.
ISIS-MTT konforme Zeitstempeldienste müssen das Time Stamp Protocol (s. Abschnitt 3.3.1) anwenden, das vom ETSI weiter profiliert wurde (s. Abschnitt 3.3.3). Als Transportprotokoll soll HTTP unterstützt werden, da es einfach zu implementieren ist und für Firewalls und Proxyserver keinerlei Probleme darstellt.

3.3.5 Policy requirements for time-stamping authorities

In den *Policy requirements for time-stamping authorities* [24] werden hauptsächlich Richtlinien für Zeitstempeldiensteanbieter beschrieben, um die Beweiskraft der ausgestellten Zeitstempel nachweislich sicherzustellen.
Zu diesem Standard konforme Zeitstempeldienste dürfen die Policy-Identifikationsnummer

> itu-t(0) identified-organization(4) etsi(0) time-stamp-policy(02023)
> policy-identifiers(1) baseline-ts-policy (1)

benutzen. Dieser Wert findet sich außerdem in der Zeitstempel-Antwort im Feld `policy` wieder.
Basierend auf diesen Standard können auch eigene Richtlinien abgeleitet werden. Ein Beispiel findet sich in [13].
Eine TSA ist konform zu diesem Standard, falls sie:

- Angaben zu folgenden Punkten macht:
  - Zuverlässigkeit, um einen Zeitstempeldienst betreiben zu können
  - Allgemeine Geschäftsbedingungen
  - Schlüsselmanagement
  - Ausstellen von Zeitstempeln, insbesondere die Aufnahme der korrekten Zeit
  - Management und Betrieb des Zeitstempeldienstes, insbesondere Sicherheitsaspekte
- ihren Pflichten zur Einhaltung der Angaben sowie gegenüber der Benutzer (z. B. Ausfallzeiten) gerecht wird.

3.4 Lösungen

In Abschnitt 3.3 wurde vorgestellt, wie genau ein Zeitstempel auszusehen hat. Ebenso wurde erklärt, wie ein Zeitstempel angefordert und verifiziert werden kann.
In diesem Abschnitt werden zwei weiterführende Konzepte vorgestellt. In Abschnitt 3.4.1 wird auf intervall-qualifizierte Zeitstempel zur Kostensenkung bei der Anforderung von Zeitstempeln verwiesen, bevor in Abschnitt 3.4.2 das „ArchiSig“-Konzept erläutert wird. Dieses Konzept sieht eine SigG-konforme Neusignierung zur Langzeitarchivierung elektronisch signierter Dokumente vor.

3.4.1 Intervall-qualifizierte Zeitstempel

Die Ausstellung von qualifizierten Zeitstempeln ist u. a. durch bereits beschriebene Sicherheits- und Überprüfungsmaßnahmen mit Kosten verbunden. Diese Kosten werden natürlich an die Kunden weitergeleitet und können abhängig von der Anzahl der ausgestellten Zeitstempel in die Höhe steigen. In [33] beschreibt D. Hühnlein einen Weg, die anfallenden Kosten mit *intervall-qualifizierten* Zeitstempeln zu reduzieren.

3.4.2 „ArchiSig"-Konzept

Das im Rahmen des „ArchiSig"-Projekts entwickelte Konzept (vgl. auch [51, S. 86-91]) zur beweiskräftigen Archivierung von elektronischen Dokumenten befindet sich auf dem Weg der internationalen Standardisierung durch die ltans-Arbeitsgruppe[12] und wird im aktuellen Draft [32] näher definiert.
Das „ArchiSig"-Konzept hat mehrere Vorteile im Vergleich zu anderen Konzepten, wie z. B. von ETSI [25] oder der Signatur-Interoperabilitätsspezifikation [14], insbesondere was die Performanz und die Wirtschaftlichkeit der erneuten Signatur angeht.
Die Idee von „ArchiSig" ist es, die Signaturerneuerung durch ein zentrales Archivierungssystem durchführen zu lassen, bei der man mehrere Dokumente gleichzeitig mit einem Zeitstempel versehen kann.

3.4.2.1 Initiale Archivzeitstempelung

Ein Dokument muss bei seiner Archivierung spätestens vor Ablauf der Sicherheitseignung der verwendeten kryptographischen Algorithmen mit einem Zeitstempel versehen werden. Dabei wird das Dokument gehasht und zusammen mit anderen Hashwerten ein so genannter Hashbaum aufgebaut, wie er in Abbildung 3.6 zu sehen ist. Dabei berechnet sich der Wert eines Knotens durch die Konkatenation und anschließende Hashwertberechnung der Kindknoten. Die Wurzel des Baums wird anschließend mit dem so genannten Archivzeitstempel versehen.
Als Neusignatur im rechtlichen Sinne kann der reduzierte Archivzeitstempel (s. Abbildung 3.7) herangezogen werden. Dafür muss der Hashbaum nicht komplett neu aufgebaut werden, sondern man kann die bereits berechneten Zwischenschritte verwenden, um den Hashwert $h_{1.7}$ zu rekonstruieren.
Um die Beweiskraft des Archivzeitstempel zu wahren, muss dieser wiederum erneut signiert werden. Dafür gibt es zwei Verfahren, auf die in den beiden nächsten Abschnitten eingegangen wird.

3.4.2.2 Neusignierung eines Zeitstempels

Bei Verlust der Sicherheitseignung der kryptographischen Algorithmen des Archivzeitstempels ist es ausreichend, nur die vorhandenen Archivzeitstempel neu zu signieren - vorausgesetzt, das verwendete Hashverfahren ist noch sicher.
Wie in Abbildung 3.8 dargestellt, wird einfach ein zweiter Archivzeitstempel erzeugt, der den alten einschließt. Der reduzierte Archivzeitstempel rA_2 ist dann die neue Signatur nach § 17 SigV.
Der Vorteil dieses Verfahrens ist, dass es sehr effizient und kostengünstig realisierbar ist.

[12] Long-Term Archive and Notary Services (ltans) `http://ltans.edelweb.fr`

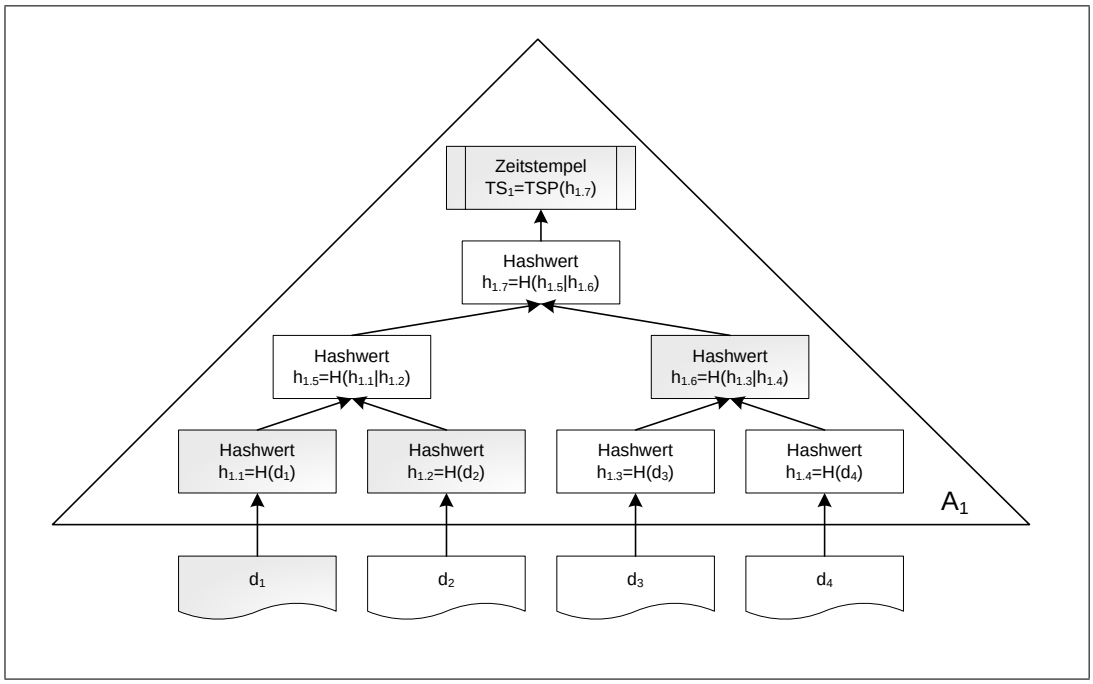

Abbildung 3.6: Hashbaum mit Archivzeitstempel (vgl. [51, S. 87])

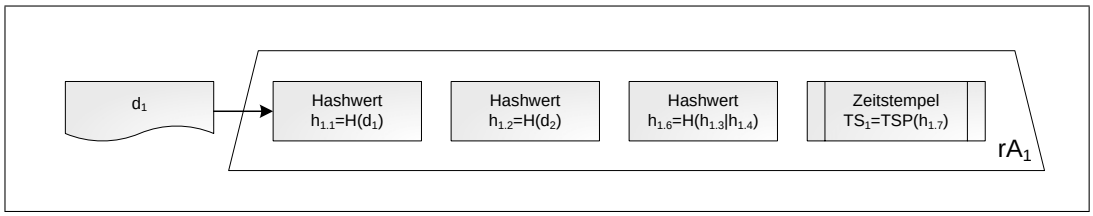

Abbildung 3.7: Reduzierter Archivzeitstempel (vgl. [51, S. 87])

3.4.2.3 Neusignierung eines Hashbaums

Falls das verwendete Hashverfahren unsicher wird, müssen bei der erneuten Signatur der komplette Hashbaum aber auch alle referenzierten Dokumente berücksichtigt werden.
Wie in Abbildung 3.9 verdeutlicht, werden die vorhandenen reduzierten Archivzeitstempel rA_1 und rA_2 sowie das initiale Dokument d_1 mit dem neuen sicherheitsgeeigneten Hashalgorithmus gehasht.
Der neue Archivzeitstempel A_3 kann wieder, wie bereits oben beschrieben, reduziert werden. Der so entstandene reduzierte Archivzeitstempel entspricht wieder der erneuten Signatur des Dokuments.
Auch dieses Verfahren ist konform zu § 17 SigV, da der Archivzeitstempel jeweils das signierte Dokument und alle vorherigen Signaturen umfasst. Allerdings ist die unvermeidbare Neusignierung des Hashbaums nicht sehr performant, da auf alle Dokumente zugegriffen werden muss.

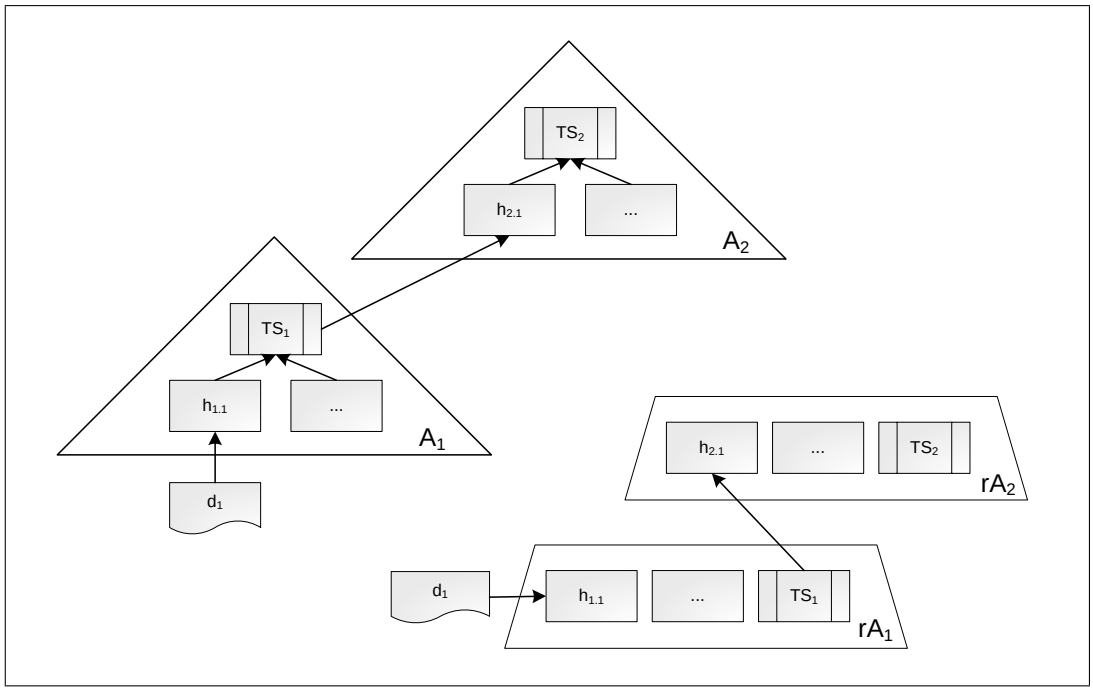

Abbildung 3.8: Erneuerung des Zeitstempels (vgl. [51, S. 89])

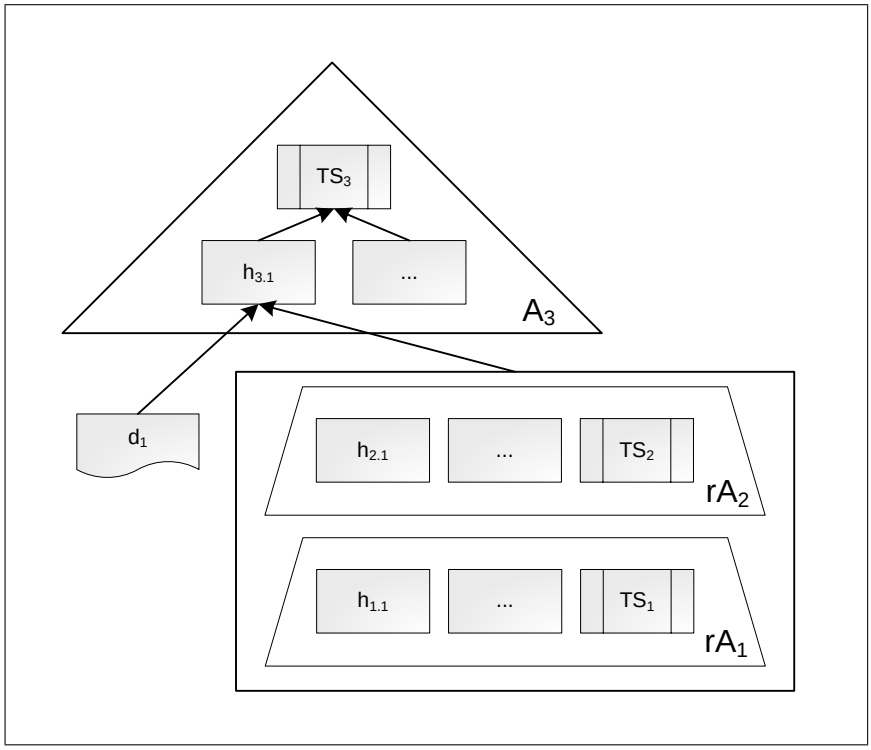

Abbildung 3.9: Erneuerung des Hashbaums (vgl. [51, S. 90])

Kapitel 4

Realisierung

4.1 Anforderungen

An den im Rahmen dieser Diplomarbeit entwickelten Zeitstempeldienst werden, neben den bereits genannten Vorgaben aus Abschnitt 3.2.6, noch andere funktionale Anforderungen gestellt:

Mandantenfähigkeit Der Zeitstempeldienst soll für mehrere (virtuelle) ZDAs Zeitstempel erzeugen können. Sämtliche Belange der Zeitstempelerzeugung und Signatur bzgl. kryptographischer Verfahren und SSEE-Anbindung erfolgen getrennt nach Mandanten. Insbesondere sollen die Mandanten unabhängig voneinander fortgeschritten oder qualifiziert signieren können.

Robustheit Der Zeitstempeldienst muss hochverfügbar sein. Da dies nicht nur mit Softwaremitteln erreichbar ist, müssen zumindest einige Rahmenbedingungen erfüllt werden, damit man ein hochverfügbares System aufsetzen kann.

- Der Zustand des Zeitstempeldienstes muss jederzeit festgestellt werden können. Sollte der Server in einen Zustand geraten, in dem keine weiteren Zeitstempel mehr ausgestellt werden dürfen, so muss dies erkennbar sein.
- Der Zeitstempeldienst muss robust gegenüber falschen Zeitstempel-Anfragen sein. Es dürfen also keine Ressourcen blockiert werden, die ein ordnungsgemäßes Arbeiten des Servers verhindern.

Revisionssichere Protokollierung Der Zeitstempeldienst muss alle Vorfälle im Server protokollieren, insbesondere muss er eine Liste der ausgestellten Zeitstempel führen.

Austauschbarkeit Der Server soll möglichst frei konfigurierbar sein, besonders in Hinblick auf die verwendeten kryptographischen Verfahren.

Erweiterbarkeit Der Server soll besonders in Bezug auf die Signaturkomponente für andere Dienste erweiterbar sein. So könnte der bestehende OCSP-Server oder Timestamping-Server sowie andere Massensignaturdienste auf ein- und demselben Rechner laufen.

Technische Vorgaben Der Zeitstempeldienst soll als EJB-Anwendung auf dem JBoss Application Server[1] realisiert werden. Die Anbindung der Signaturerstellungseinheiten erfolgt mit

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

dem von FlexSecure entwickelten Cardman-Modul. Ebenso soll das hauseigene Logging-Modul für die Protokollierung benutzt werden. Das (De-)Kodieren von ASN.1-Strukturen soll basierend auf dem Codec-Modul[2] des Fraunhofer Instituts[3] realisiert werden. Die Anbindung der kryptographischen Algorithmen erfolgt über den FlexiProvider[4].

Testclient Es ist ein Werkzeug zu entwickeln, das beispielhaft Dokumente neu signiert. An ein Dokument angebrachte Zeitstempel sollen anschließend verifiziert werden können.

Tests Schließlich sollen Tests durchgeführt werden, die die Funktionalität des Zeitstempeldienstes abdecken und seine Performanz und Robustheit gewährleisten.

4.2 Designentscheidungen

Für die technische Realisierung des Zeitstempeldienstes müssen einige Designentscheidungen getroffen werden, um genannte Anforderungen zu erfüllen.

Der Zeitstempeldienst wird in der Programmiersprache Java entwickelt und ist damit plattformunabhängig. Er wurde erfolgreich unter Linux und Solaris getestet und wird später auch auf diesen Betriebssystemen zum Einsatz kommen.

Die Absicherung des Servers wird in dieser Diplomarbeit nicht näher betrachtet. Dies betrifft die Umgebung, in der dieser sich befindet, das Betriebssystem an sich sowie den darauf aufgesetzten JBoss. Weiterführende Literatur findet sich z. B. in der Dokumentation des JBoss Application Servers [41]. Es wird bei der Entwicklung des Timestamping-Servers davon ausgegangen, dass die Systemzeit mit der gesetzlich gültige Zeit synchronisiert wird. Maßnahmen, bei denen beispielsweise der Server bei einer Zeitabweichung abgeschaltet wird, sind nicht Gegenstand dieser Arbeit.

Die Anbindung von sicheren Signaturerstellungseinheiten erfolgt mit Kartenlesern über die PC/SC[5]-Schnittstelle, über die das Cardman-Modul die Verbindung zu den Smartcards herstellt. In der technischen Realisierung wurden Kartenterminals der Firma Kobil Systems GmbH[6] angeschlossen. Der Einsatz einer Sicherheitsbox oder anderer Hardware Security Module ist vorerst nicht geplant.

Anders wie im Time Stamping Profile (s. Abschnitt 3.3.3) gefordert, wurde für diesen Server kein Store-and-Forward-Protokoll implementiert. Stattdessen wurde von den im Time Stamp Protocol (s. Abschnitt 3.3.1) vorgeschlagen Methoden HTTP und HTTPS berücksichtigt. Wie in Listing 4.2 dargestellt können Zeitstempel auch direkt über den Zugriff auf die entsprechenden EJB-Komponenten angefordert werden.

Der Timestamping-Server wurde serverseitig mit dem JDK 1.5 entwickelt, um die Vorteile von EJB 3 nutzen zu können. Eingesetzt wurde der JBoss Application Server 4.0.4. Alle Klassen und Datenstrukturen, die clientseitig eingesetzt werden, sind kompatibel zu JDK 1.4.

Da sich in Bezug auf Massensignaturen mehrere Schritte ähneln, wurde im Rahmen dieser Diplomarbeit zusätzlich ein Framework für Massensignaturanwendungen entwickelt, die über den Anwendungsbereich von Zeitstempeln hinausgehen.

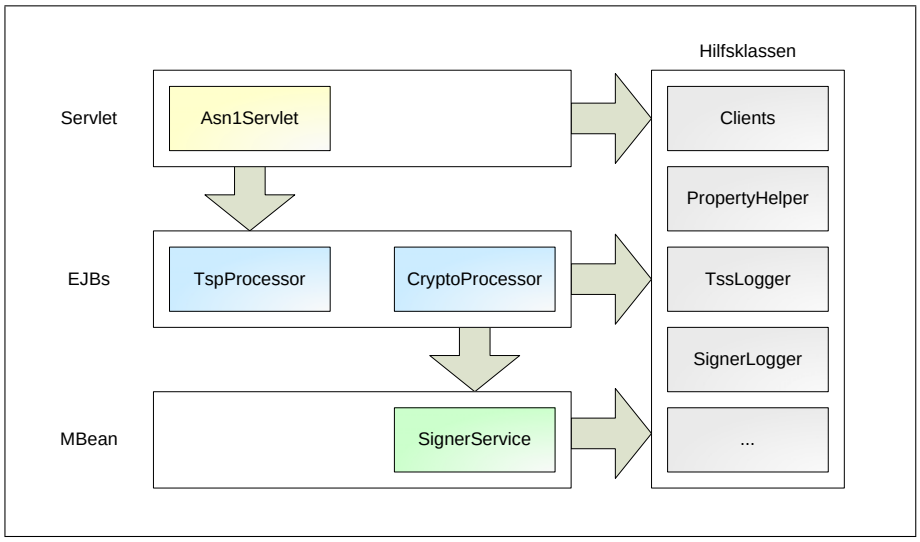

Abbildung 4.1: Vereinfachte Architektur des Timestamping-Servers

4.2.1 Architektur

In Abbildung 4.1 findet sich ein vereinfachter Überblick über die Architektur und Abläufe des Timestamping-Servers, der sich im Prinzip aus drei Schichten zusammensetzt. Dabei greift die höhere Schicht stets auf die unter ihr liegende zu.
Eine Zeitstempel-Anfrage wird in der Regel durch das `Asn1Servlet` entgegengenommen, das sie nach ersten Überprüfungen an EJBs weiterreicht (s. Abschnitt 4.2.3).
Die Anfrage wird im EJB `TspProcessor` bearbeitet, in dem auch die Zeitstempel-Antwort erzeugt wird. Im `CryptoProcessor` werden anschließend die CMS-Datenstrukturen erstellt, um die Antwort signieren zu können (s. Abschnitt 4.2.4).
Für die eigentliche Signatur sowie die Verwaltung der Signaturerstellungseinheiten wird auf die Signaturkomponente `SignerService` zurückgegriffen (s. Abschnitt 4.2.5).
Des Weiteren gibt es noch andere Hilfsklassen, die z. B. die Protokollierung (s. Abschnitt 4.2.8) übernehmen und nicht unbedingt an eine einzige Schicht gebunden sind.
Die in diesem Zusammenhang auftretenden Designentscheidungen werden im Folgendenen näher ausgeführt.

4.2.2 Datenstrukturen

Nachdem in Abschnitt 3.3 alle für Zeitstempel relevanten Standards und Datenstrukturen beschrieben wurden, soll an dieser Stelle detailliert werden, inwieweit diese Vorgaben umgesetzt wurden. Alle ASN.1-Datenstrukturen wurden basierend auf dem Fraunhofer Codec enwickelt.
In Tabelle 4.1 wird dargestellt, welche Felder bei einer Zeitstempel-Anfrage `TimeStampReq` (s. Abschnitt 3.3.1.1) unterstützt werden.
Die Zeitstempel-Antwort `TimeStampResp` wird nach Abschnitt 3.3.1.2 gebildet. Im Falle eines Fehlers wird die Fehlerinformation mitgeliefert. Die `TSTInfo`-Struktur, die in der Antwort

[2] `http://www.semoa.org`
[3] `http://www.igd.fhg.de`
[4] `http://www.flexiprovider.de`
[5] PC/SC ist eine einheitliche Schnittstelle zwischen PC und Smartcards `http://www.pcscworkgroup.com`
[6] `http://www.kobil.de`

FELD	WERT
version	v1(1)
hashAlgorithm	alle vom FlexiProvider unterstützen Hashalgorithmen
hashedMessage	s. Abschnitt 3.3.1.1
reqPolicy	s. Abschnitt 3.3.1.1
nonce	s. Abschnitt 3.3.1.1
certReq	s. Abschnitt 3.3.1.1
extensions	Erweiterungen werden nicht unterstützt. Sollten Erweiterungen vorhanden sein, wird eine Fehlermeldung zurückgegeben.

Tabelle 4.1: TimeStampReq

signiert wird, enthält die in Tabelle 4.2 beschriebenen Werte.

FELD	WERT
version	v1(1)
policy	konfigurierbar
messageImprint	s. Abschnitt 3.3.1.3
serialNumber	s. Abschnitt 3.3.1.3
genTime	basiert auf der Systemzeit
accuracy	konfigurierbar
ordering	nicht verwendet
nonce	s. Abschnitt 3.3.1.3
tsa	wird nicht generiert aber verarbeitet [a]
extensions	nicht unterstützt

Tabelle 4.2: TSTInfo

[a] Laut RFC 3161 [9] erfolgt die Identifizierung durch das `SigningCertificate`-Attribut. Das Feld `tsa` bietet an dieser Stelle keinen Mehrwert und hätte eine Wiederverwendbarkeit des `CryptoProcessor` zunichte gemacht, da er die `TSTInfo`-Stuktur kennen müsste.

In Tabelle 4.3 werden die Werte der erzeugten `SignedData`-Struktur aus Abschnitt 3.3.2 beschrieben.
Schließlich wird die `SignerInfo`-Struktur erzeugt, wie sie in Tabelle 4.4 detailliert wird.

4.2.3 Annahme von Anfragen

Die Annahme von Zeitstempel-Anfragen wird von einem Servlet übernommen, das sie entgegennimmt, sie dekodiert und an die Businesslogik der EJBs weiterleitet, wo letztendlich der Zeitstempel erzeugt wird. Anschließend leitet das Servlet die Antwort an den Client weiter.
Da das Servlet bei dieser Funktionsweise nicht wissen muss, ob es sich um eine Zeitstempel-Anfrage handelt oder nicht, wurde ein generisches Servlet entwickelt, das sich auch in anderen Kontexten benutzen lässt. Dafür muss es in besonderem Maße konfigurierbar sein.
Das so entwickelte `Asn1Servlet` ist auf eine `POST`-Anfrage einer ASN.1-kodierten Nachricht ausgerichtet und führt, wie in Abbildung 4.2 dargestellt, eine Überprüfung des HTTP-Headers durch. Ist die `ContentLength` zu lang oder entspricht der `ContentType` nicht der erwarteten Anfrage, so wird sie abgewiesen. Anschließend wird versucht, die ASN.1-kodierte Anfrage zu dekodieren und an die EJBs weiterzuleiten sowie das Ergebnis wieder zu kodieren

FELD	WERT
version	v3(3)
digestAlgorithms	s. Abschnitt 3.3.2
eContentType	id-ct-TSTInfo
eContent	die kodierte `TSTInfo`-Struktur
certificates	s. Abschnitte 3.3.1.1 und 3.3.2
crls	nicht benutzt
signerInfos	genau eine `SignerInfo`-Struktur

Tabelle 4.3: SignedData

FELD	WERT
version	v1(1)
sid	Identifikation des Signaturzertifikats mit Issuer und Seriennummer
digestAlgorithm	konfigurierbar
signedAttrs	• `ContentType` mit Wert id-ct-TSTInfo • `MessageDigest` mit Hashwert des Feldes `eContent` • `SigningCertificate` mit Hashwert und Identifikationsdaten (Issuer und Seriennumer) für jedes Zertifikat der Zertifikatskette
signatureAlgorithm	ergibt sich aus `digestAlgorithm` und öffentlichem Schlüssel des Signaturzertifikats
signautre	s. Abschnitt 3.3.2
unsignedAttrs	nicht benutzt

Tabelle 4.4: SignerInfo

und an den Client zurückzuschicken. Sollte ein Fehler im Servlet aufgetreten sein, ist es möglich sowohl einen HTTP-Statuscode (Standardverhalten) oder eine ASN.1-kodierte Fehlermeldung zurückzugeben.
Das Servlet lässt sich auf zwei verschiedene Arten konfigurieren:

1. Über *Java-Properties*[7]. Dadurch kann das Servlet sofort und ohne Änderungen im Quellcode eingesetzt werden. Es wurde darüberhinaus darauf geachtet, dass sich auch mehrere unterschiedlich konfigurierte Instanzen in einem Container befinden können. Eine Übersicht und Erklärung aller Eigenschaften befindet sich in Anhang B.
   Der zentrale Zugriff auf die Properties erfolgt mit der Hilfsklasse `PropertyHelper`.

2. Über Vererbung. Falls bereits beschriebene Eigenschaften nicht ausreichen oder ASN.1-kodierte Fehlermeldungen zurückgegeben werden sollen, ist es möglich, sein eigenes Servlet von `Asn1Servlet` abzuleiten, um es um die gewollte Funktionalität zu erweitern. Dieses Vorgehen sollte jedoch eine Ausnahme bleiben.

[7] Unter Java-Properties versteht man einen einfachen Mechanismus, um Java-Anwendungen dateibasiert konfigurieren zu könnnen. Weitere Informationen finden sich in der Java-API [58].

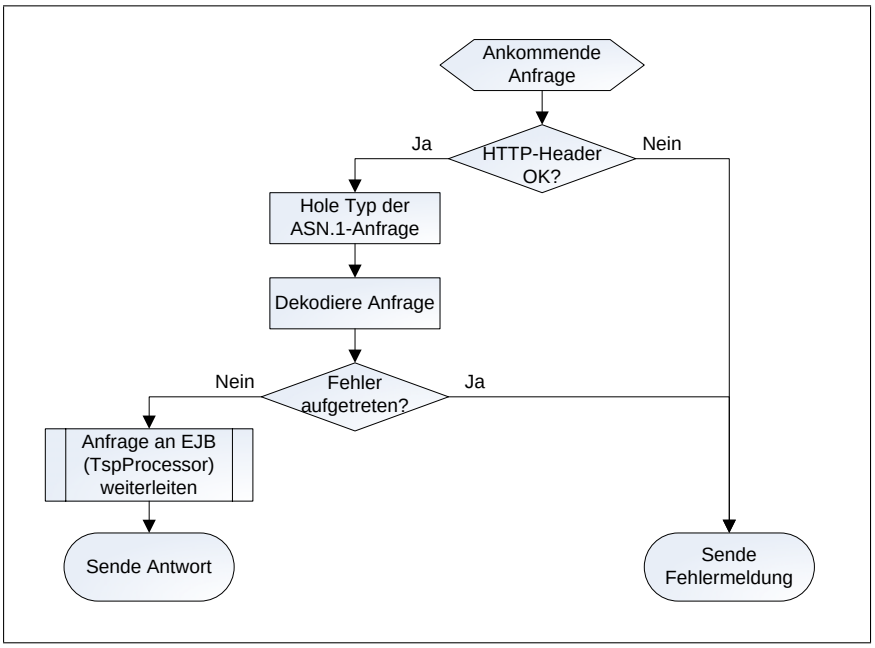

Abbildung 4.2: Flussdiagramm des Asn1Servlets

4.2.4 Business-Logik

Die Business-Logik des Timestamping-Servers setzt sich aus zwei *Stateless Session-Beans* zusammen.

4.2.4.1 TspProcessor

Der `TspProcessor` nimmt die bereits dekodierte `TimeStampReq`-Anfrage entgegen und baut die `TimeStampResp`-Antwort auf. Dabei werden einige Überprüfungen vorgenommen, wie in Abbildung 4.3 graphisch verdeutlicht wird. So wird geprüft,

- ob das Hashverfahren, mit dem die Daten gehasht wurden, akzeptiert wird. Dadurch kann ausgeschlossen werden, dass Zeitstempel mit bereits unsicheren Hashalgorithmen ausgegeben werden. Alle akzeptierten Verfahren können beliebig konfiguriert werden.
- ob die Policy, mit der der Zeitstempel ausgestellt werden soll, akzeptiert wird. Alle akzeptierten Policys können beliebig konfiguriert werden.
- dass keine Erweiterungen in der Anfrage mitgeschickt wurden.

Die Seriennummer der Zeitstempel muss in einer Datenbank gespeichert werden um ihre Eindeutigkeit zu gewährleisten.
Zuerst wurden zur Persistenz der Seriennummer die Verwendung von Entities und Transaktionen in Betracht gezogen, um auf eine breitere Palette von Java EE zurückzugreifen.
Während Entities für die Persistenz von (komplexen) Datenstrukturen ausgelegt sind, wird für die Seriennummer nur eine fortlaufende Zahl benötigt. Folglich lässt sich der Einsatz von Entities nicht rechtfertigen, wodurch sich die klassische Alternative der Datenbankanbindung über JDBC anbietet.
Dabei gibt es zwei Dinge zu beachten:

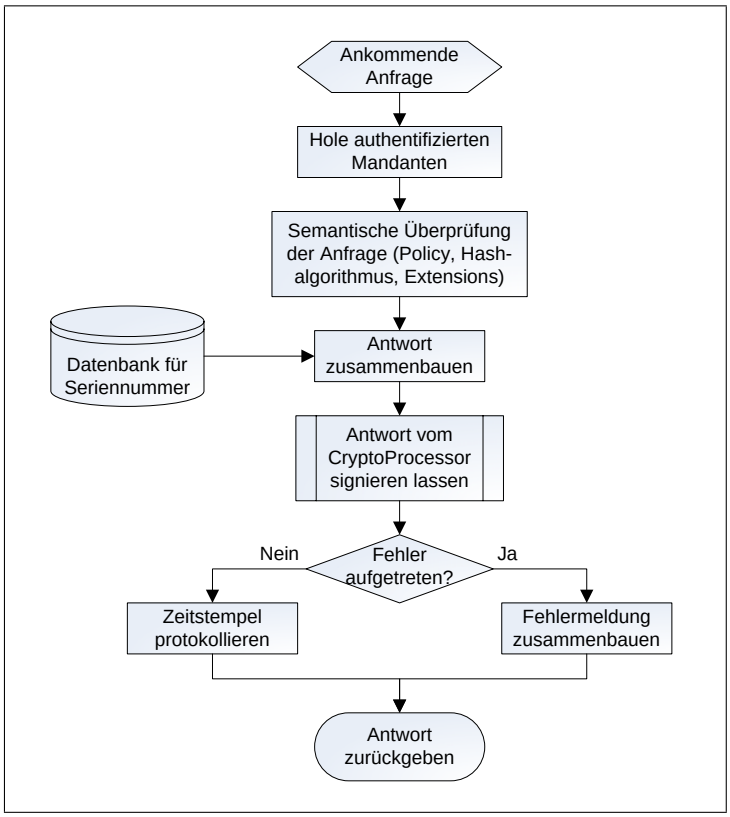

Abbildung 4.3: Flussdiagramm des TspProcessors

1. Es darf keine Abhängigkeit von einer bestimmten Datenbank geben.

2. Für eine fortlaufende Nummer sind am Besten *Sequenzen* geeignet, da ein eindeutiger Wert in einem atomaren Schritt zurückgegeben wird. Somit ist sichergestellt, dass bei zwei gleichzeitigen Transaktionen auch immer zwei unterschiedliche Werte zurückgeliefert werden. Allerdings sind Sequenzen nicht standardisiert und somit stark von der Datenbank abhängig, was in Widerspruch zu Punkt 1 steht.

Die einfachste Lösung ist in diesem Fall, den SQL-Code direkt aus der Konfigurationsdatei zu beziehen, um jeweils den nächsten Wert der Seriennummer zu beziehen. Auf diese Weise kann die Datenbankabfrage stets manuell einer geänderten Umgebung angepasst werden. Diese Lösung wurde auch implementiert.
Wie bereits in Abschnitt 2.3.5 angedeutet, wäre es möglich, Java Enterprise-Anwendungen auch ohne EJB zu entwickeln bzw. EJBs mit Spring zu konfigurieren.
In oben beschriebenen Fall würde sich jedoch der Konfigurationsaufwand des Applikationsservers nicht gegen die implementierte Variante rechnen, da Session-Beans nicht direkt, sondern von einem Container verwaltet werden. Außerdem sollte die Kompatibilität mit bereits existierendem OCSP-Server gewahrt werden, man hätte nämlich die Schnittstellen der EJBs anpassen müssen.
Standardmäßig wird die in JBoss integrierte HSQL-Datenbank[8] verwendet, die jedoch für produktive Arbeitsumgebungen nicht geeignet ist.

[8] `http://hsqldb.org`

4.2.4.2 CryptoProcessor

Während im `TspProcessor` zeitstempelspezifische Datenstrukturen bearbeitet wurden, kümmert sich der `CryptoProcessor` um die Objekte des Cryptographic Message Syntax (s. Abschnitt 3.3.2) und bringt an ihnen die Signatur an, die in der Signaturkomponente (s. Abschnitt 4.2.5 erzeugt wird.
In Abbildung 4.4 wird der interne Ablauf des `CryptoProcssor` in Form eines Flussdiagramms dargestellt.

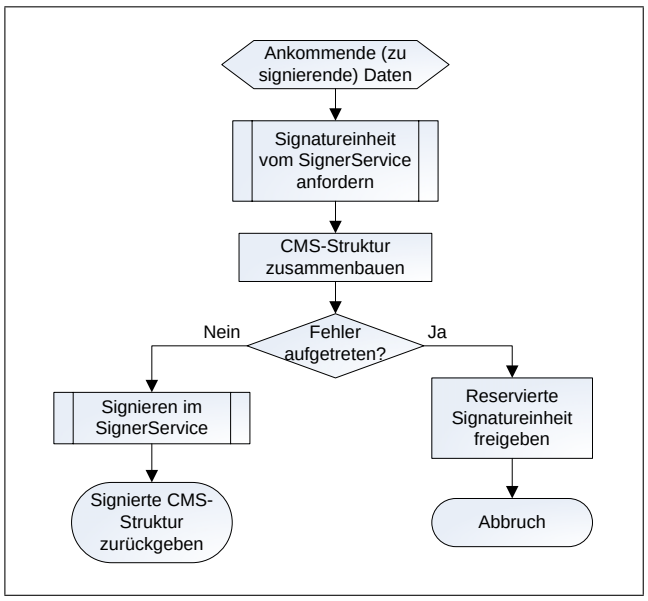

Abbildung 4.4: Flussdiagramm des CryptoProcessors

In folgendem Listing 4.1 werden bereits die folgenden Abschnitte vorweggenommen, um ein vereinfachtes Beispiel davon zu geben, wie von Seiten der EJBs Signaturen erzeugt werden können.

Listing 4.1: Erzeugen einer Signatur

```
import de.flexsecure.flexiTrust.tss.jmx.SignerService;
import de.flexsecure.flexiTrust.tss.jmx.SignerData;
import de.flexsecure.flexiTrust.tss.jmx.clients.Client;

import javax.ejb.Stateless;
import javax.ejb.Local;
import javax.annotation.EJB;

@Local({ CryptoProcessor.class })
@Stateless
public class CryptoProcessorBean implements CryptoProcessor
{

  // die Signaturkomponente wird automatisch injiziert
  @EJB
  private SignerService signerService;
```

```
  // eine Business-Methode um die Daten in data zu signieren
  public byte[] signData(byte[] data)
  {
    // hole den authentifizierten Mandanten
    Client client = (Client)
      AuthUtil.getAuthenticatedPrincipal(Client.class);

    // reserviere eine freie Signatureinheit
    SignerData signer =
      signerService.getSigner(client.getId());

    // das Signaturverfahren
    String sigAlg = ...;

    // Erzeugen der Signatur
    byte[] signature = signerService.signData(data, sigAlg,
       signer);

    return signature;
  }
}
```

4.2.5 Signaturkomponente

Da EJBs nicht für die Verwaltung von Ressourcen ausgelegt sind, muss insbesondere die Anbindung der Kartenleser anders gelöst werden. Für diese Aufgabe bieten sich die Java Management Extensions an (s. Abschnitt 2.3.4).
Die Signaturkomponente `SignerService` wird in Form eines MBeans realisiert, das im Gegensatz zu EJBs nur einmal im Server instanziiert wird. Das letztendliche Ansprechen der Kartenleser erfolgt über das von FlexSecure entwickelte Cardman-Modul, das gleichzeitig Signatureinheiten über Hard- und Softtoken verwalten kannn.
In einem Thread wird im `SignerService` in regelmäßigen Intervallen nach neuen Karten gesucht und diese anschließend einem Mandanten zugeordnet. Dafür wird das Zertifikat auf der Karte mit den in einem Verzeichnis des Mandanten abgelegten Zertifikaten verglichen. Nach einer erfolgreichen Aktivierung (s. Abschnitt 4.2.10) und Prüfung der Zertifikatskette wird die Karte zu einer Liste von verfügbaren Signaturkarten hinzugefügt.
Bei einer Signaturanfrage wird, wie in Abbildung 4.5 verdeutlicht, eine freie Signatureinheit des Mandanten in die Liste der aktuell verwendeten Signaturkarten hinzugefügt. Dabei wird sichergestellt, dass das Signaturzertifikat noch gültig ist und dass bei gleichzeitigen Anfragen so lange gewartet wird, bis wieder eine Signatureinheit frei wird. Ebenso ist es möglich, diese Wartezeit über eine angefragte höhere Priorität zu verkürzen. Dies wird im bestehenden OCSP-Server in Anspruch genommen. Folglich werden die Anfragen nicht unbedingt nach dem FIFO-Prinzip bearbeitet.
Nach erfolgreicher Signatur wird die Signatureinheit wieder aus der Liste der verwendeten Signaturkarten gelöscht.
Wie bereits angesprochen, können Dienste über JMX überwacht werden. JBoss stellt hier-

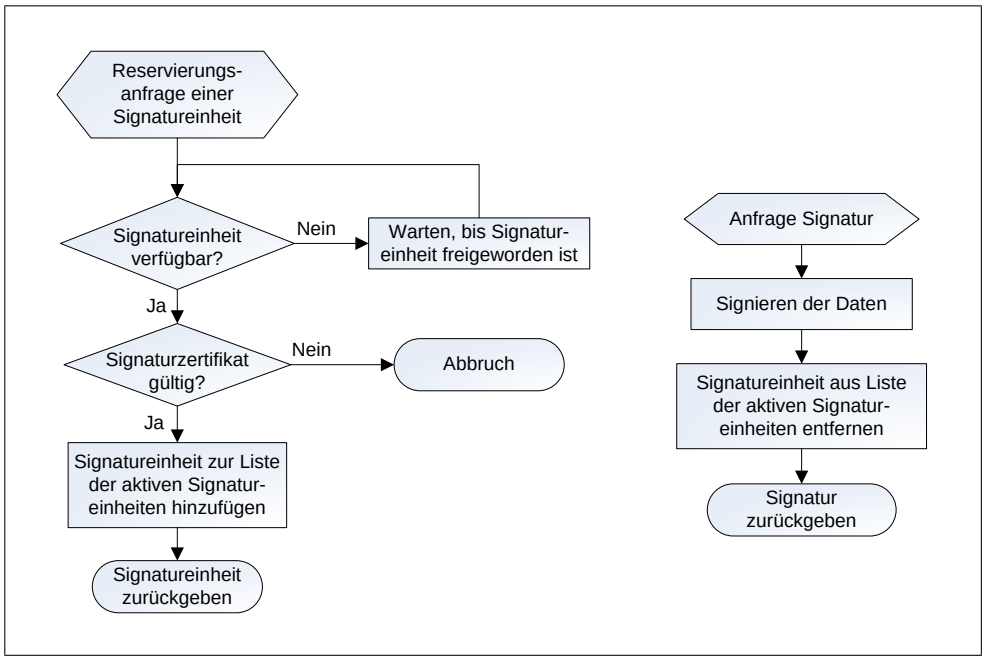

Abbildung 4.5: Flussdiagramme des SignerService

für eine spezielle JMX Console bereit, die eine Verwaltung des JBoss Application Servers über einen Webbrowser ermöglicht. Wie in Abbildung 4.6 dargestellt, kann der Zustand des `SignerService` ebenfalls über JMX abgefragt werden.
Die Signaturkomponente gibt Auskunft über folgende Eigenschaften:

- ob der Dienst gestartet ist,
- alle verfügbarenen Signaturereinheiten mit Zuordnung zu den Mandanten,
- alle aktuell verwendeten Signaturereinheiten mit Zuordnung zu den Mandanten,
- weitere Informationen über die aktivierten Signatureinheiten.

Abbildung 4.7 zeigt den Aufbau des `SignerService` und die Verzahnung mit der Mandantenverwaltung. Auch in den weiteren Abschnitten wird dieses Diagramm noch eine Rolle spielen.

4.2.6 Mandantenverwaltung

Die Mandantenverwaltung des Timestamping-Servers wurde im Laufe der Zeit mehrmals überarbeitet. Das Ziel ist es, alle Mandanten des Zeitstempeldiensts zu verwalten, jedoch gleichzeitig auf zusätzliche Dienste vorbereitet zu sein. Der bereits existierende OCSP-Server hat gezeigt, dass die Anforderungen an die Datenhaltung der Mandanten von einem Dienst zum anderen stark variieren können.
Eine solche Problemstellung kann durch Dependency Injection gelöst werden.
Dazu wird die Annahme getroffen, dass sich alle Mandanten auf einige wenige Gemeinsamkeiten reduzieren lassen: das Signieren von Daten. Dafür werden nur Informationen benötigt, die auch von der Signaturkomponente benutzt werden, nämlich Signatur- und Ausstellerzertifikate sowie die hinterlegten Daten für das Pinsharing. Wie in Abbildung 4.7 verdeutlicht, besitzt die

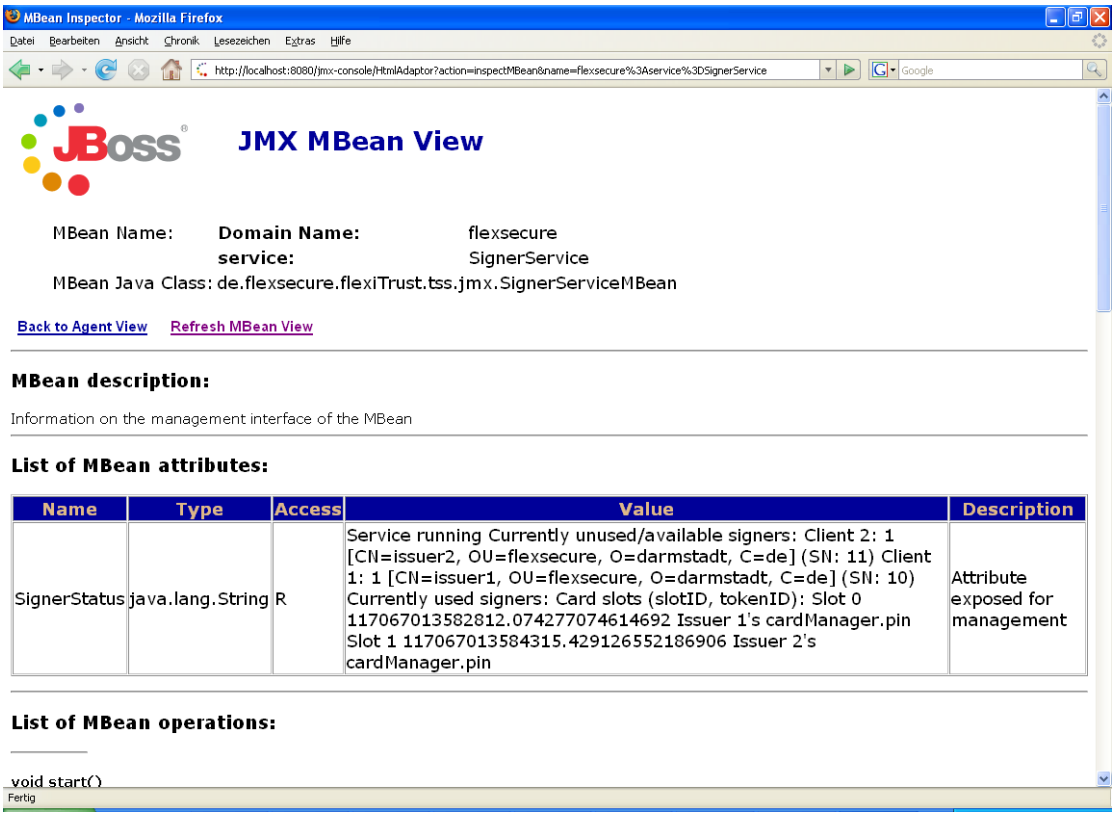

Abbildung 4.6: JMX Console

entwickelte Klasse `Client` für diese Informationen Eigenschaften, denen die Pfadangaben zu diesen Daten gesetzt werden können.

Während der Initialisierungsphase des `Client`-Objekts werden Signatur- und Ausstellerzertifikate eingelesen. Die Pinsharing-Informationen werden erst von der Signaturkomponente bei der Aktivierung der Signaturerstellungseinheiten verwendet (s. Abschnitt 4.2.10).

Für dienstspezifische Zusatzdaten können von der Klasse `Client` eigene Klassen abgeleitet werden. Im Zusammenhang mit dem entwickelten Timestamping-Server ist so z. B. die Klasse `TssClient` entstanden, von der wiederum zwei Klassen bezüglich der Authentifzierung (s. Abschnitt 4.2.7) abgeleitet wurden.

Die erste Lösung sah eine Konfiguration der Mandanteninformationen in XML-Format vor, anhand derer die `Client`-Objekte durch eine Factory erstellt und konfiguriert wurden. Dabei wurde das Parsen der XML-Dateien selbst vorgenommen, was bei evtl. komplexeren Datenstrukturen durchaus aufwendig werden kann.

Aus diesem Grund fiel die Entscheidung letztendlich auf das Spring-Framework, insbesondere, weil die verantwortlichen Klassen der Mandantenverwaltung manuell an nur einer einzigen Stelle initialisiert werden. Deshalb kann die XML-Konfigurationsdatei direkt eingelesen werden, ohne dass ein EJB-Container konfiguriert werden muss, wie es in Abschnitt 4.2.4.1 der Fall gewesen wäre.

Durch den Einsatz von Spring können also komplexe Objektabhängigkeiten von `Client`-Klassen aufgelöst werden, ohne auch nur eine einzige Zeile selbst programmiert zu haben.

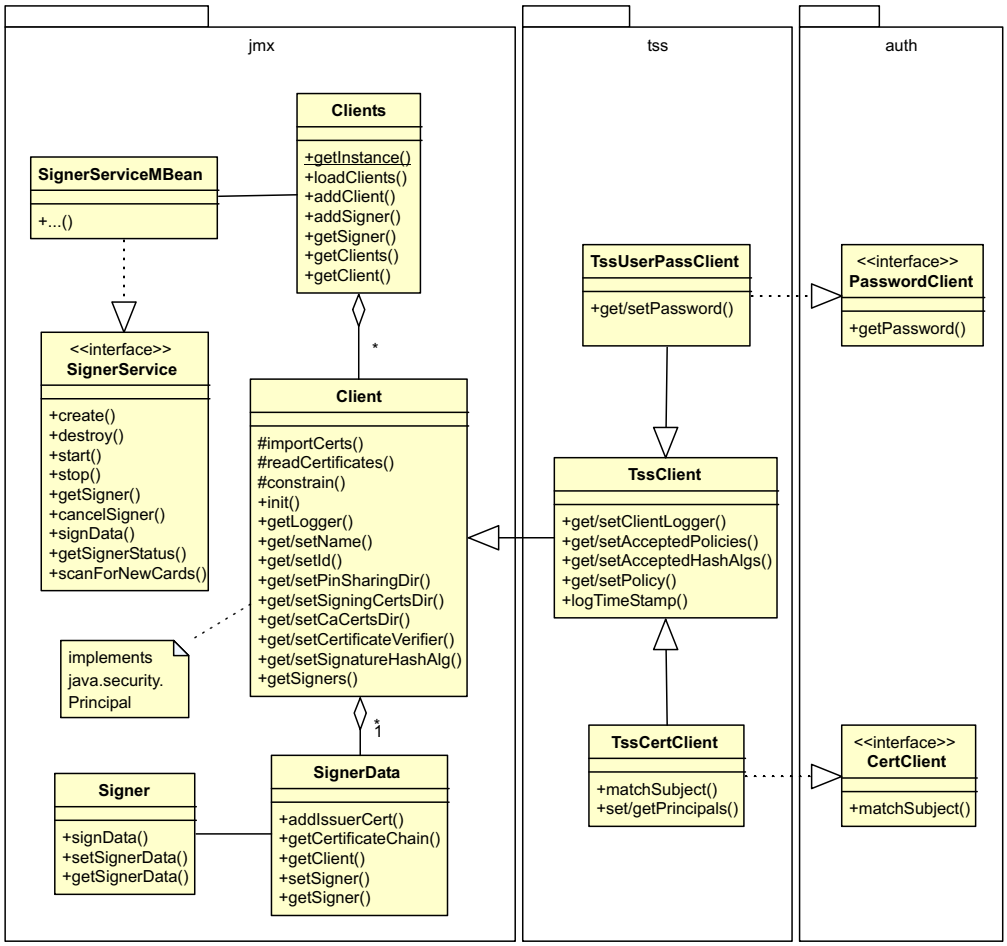

Abbildung 4.7: UML-Diagramm des SignerService inkl. der Mandantenverwaltung

4.2.7 Authentifizierung

Bei einem mandantenfähigen Zeitstempeldienst ist es von großem Interesse, jeder Zeitstempel-Anfrage einen Mandanten zuzuordnen. Je nach Anwendungsfall und Geschäftsmodell können ihnen die ausgestellten Zeitstempel in Rechnung gestellt werden. Aus diesem Grund muss gewährleistet werden, dass nur durch erfolgreiche Identifikation des Antragssteller ein Zeitstempel ausgestellt werden kann.
Der entwickelte Timestamping-Server unterstützt standardmäßig Client-Authentifizierung

- mit Benutzername und Passwort (z. B. Basic[9]) über HTTP
- mit Client-Zertifikaten über HTTPS.

Da JBoss für die Authentifizierung JAAS benutzt, erfolgt die Identifikation der Mandanten mit drei Login-Modulen, die hintereinandergeschaltet sind. Sie ist also erst dann erfolgreich, wenn sich der Mandant entweder mit Benutzernamen und Passwort oder mit einem Client-Zertfikat am Server angemeldet hat.

[9] *Basic* ist eine Methode für Authentifizierung über HTTP und ist in RFC 2617 [30] spezifiziert.

Für die Authentifizierung per Benutzername und Passwort wurde das `PasswordLoginModule` entwickelt.
Für die Authentifizierung mit Client-Zertifikaten wurde die Verwendung des Auth-Moduls (s. [44]) geprüft, jedoch war es nicht mit JBoss kompatibel.[10] Außerdem stellt Auth eine rollenbasierende Authentifizierung bereit, die für den Timestamping-Server nicht benötigt wird.
Die entwickelte Lösung sieht die folgende Vorgehensweise vor, die generell in drei Etappen abläuft:

1. Anfordern des Client-Zertifikats durch den Webserver und Prüfung des Zertifikats auf Vertrauenswürdigkeit.

2. Login mit dem `BaseCertLoginModule` von JBoss, das das Zertifikat für die weitere Verwendung bereitstellt.

3. Login mit dem eigens entwickelten `CertLoginModule`, das schließlich dem Zertifikat einen Mandanten zuordnet. Dabei kann ein Mandant durchaus mehrere Client-Zertifikate akzeptieren. So ist es beispielsweise möglich, dass ein Mandant alle Zertifikate eines bestimmten Issuers annimmt. Es sind aber auch eindeutige Zuordnungen mit Issuer und Seriennummer des Zertifikats vorgesehen. Siehe dazu auch die Konfigurationsmöglichkeiten im Anhang B.

Das entwickelte Framework stellt, wie in Abbildung 4.7 veranschaulicht, verschiedene Interfaces bereit, damit auch andere Dienste diese Form der Authentifizierung mit minimalem Aufwand mitbenutzen können. Die Mandanten-Datenstrukturen (s. Abschnitt 4.2.6) müssen dafür nur die Interfaces `PasswordClient` oder `CertificateClient` implementieren. Dadurch, dass die Basisklasse für alle Mandanten `Client` das Interface `java.security.Principal` implementiert, kann ein Subject direkt mit einem Mandanten assoziiert werden.
Die genaue Methode der Authentifizierung wird in der Regel im Deployment-Deskriptor des Servlets spezifiziert. Auf diese Weise kann schon frühzeitig entschieden werden, ob die Benutzer ausreichende Rechte haben, den Zeitstempeldienst zu nutzen.
Hat sich ein Client erfolgreich am Servlet angemeldet, kann man nicht davon ausgehen, dass die aufgerufenen EJBs das Subject des Servlets übernehmen. Dies ist nur der Fall, wenn das Servlet als auch die betreffenden EJBs unter der gleichen *Security-Domain*[11] ausgeführt werden. Dies lässt sich in Form von Deployment-Deskriptoren oder auch von JBoss bereitgestellten Annotationen realisieren.
Soll der Zeitstempeldienst von anderen Anwendungen eingesetzt werden, kann auf die EJBs auch direkt zugegriffen werden. Denkbar wäre u. a. ein Massensignatursystem, das die erzeugten Signaturen gleich mit einem Zeitstempel versieht. Das direkte Ansprechen von EJBs hat neben der verbesserten Performanz[12] auch eine verbesserte Fehlerbehandlung als Vorteil. Anwendungen können *Java-Exceptions* selbst behandeln und dadurch gezielt auf spezielle Fehler reagieren.
Listing 4.2 gibt ein Beispiel, wie man eine Verbindung mit den EJBs herstellt und wie man außerdem die Authentifizierung über Client-Zertifikate sicherstellt.

[10] Das `SSLClientCertificateLoginModule` des Auth-Moduls verwendet einen `CertificateCallback`, um an die Client-Zertifikate zu kommen. Für JBoss ist dieser allerdings unbekannt.

[11] Eine Security-Domain ist eine Art Richtlinie, die für Authentifizierung und Authorisierung verwendet wird. Dazu gehört z. B. die Angabe der Login-Module.

[12] Die (De-)Kodierung von Datenstrukturen entfällt z. B. bei lokalem Zugriff komplett.

Listing 4.2: Authentifierung mit JNDI

```
import java.io.InputStream;
import java.io.FileInputStream;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateFactory;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;

import de.flexiprovider.core.FlexiCoreProvider;
import de.flexsecure.flexiTrust.tss.tsp.TimeStampReq;
import de.flexsecure.flexiTrust.tss.tsp.TimeStampResp;
import de.flexsecure.flexiTrust.tss.ejb.TspProcessor;

public class JndiClient
{
  public static void main(String[] args) throws Exception
  {
    // Lade den Security Provider für Hashwertberechnung
    Security.addProvider(new FlexiCoreProvider());

    // Client-Zertifikat laden
    InputStream inStream = new FileInputStream("client1.der");
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
    X509Certificate cert =
      (X509Certificate)cf.generateCertificate(inStream);
    inStream.close();

    // Authentifikation vorbereiten
    Properties env = new Properties();
    env.setProperty(Context.INITIAL_CONTEXT_FACTORY,
      "org.jboss.security.jndi.JndiLoginInitialContextFactory");
    env.setProperty(Context.PROVIDER_URL,
      "jnp://localhost:1099/");
    env.put(Context.SECURITY_CREDENTIALS, cert);
    env.put(Context.SECURITY_PRINCIPAL,
      cert.getSubjectDN().toString());

    InitialContext context = new InitialContext(env);

    //mache Lookup
    TspProcessor processor = (TspProcessor)
      context.lookup("tss-ejb/TspProcessorBean/remote");

    // sende Anfrage
    TimeStampReq req = new TimeStampReq();
    req.getMessageImprint().calculateHash("".getBytes(), "SHA");
```

```
    TimeStampResp response =
      processor.processTimeStampRequest(req);
    ...
  }
}
```

4.2.8 Protokollierung

Für die Protokollierung wird die von FlexSecure verwendete `AdLogger`-Komponente[13] benutzt. Sie kommt an drei verschiedenen Stellen zum Einsatz:

- Der `SignerLogger` protokolliert im `SignerService` das Einlesen der Konfigurationsdaten und vor Allem die Ereignisse der Kartenleser.
- Der `TssLogger` schneidet quer durch die komplette Architektur. Er loggt alle Vorgänge während einer Zeitstempel-Anfrage vom `Asn1Servlet` bis hin zur Signatur im `SignerService`.
- Zuletzt besitzt jeder (Zeitstempel-)Mandant einen eigenen Logger, der die ausgestellten Zeitstempel pro Mandant festhält. Dabei werden die Vorgaben aus dem Maßnahmenkatalog [50] beachtet, so dass folgende Felder protokolliert werden:
  - ID des Mandanten
  - Name des Mandanten
  - Zeitpunkt der Zeitstempelerzeugung
  - Seriennummer des Zeitstempels
  - Signatur
  - Ausstellendes Zertifikat mit Isser und Seriennummer

Alle Logger unterscheiden sich jeweils nur bei ihrer Initialisierung, da jeder Logger anders konfiguriert werden kann und soll. Um Fehlerquellen und Redundanz im Code zu vermeiden, wurde die Initialisierung über Java-Properties in eine eigene Klasse `UtilLogger` ausgelagert. Die Logger-Parameter werden in der Form `loggerName.propertyName` angegeben und müssen bei jeder Initialisierung der Logger eingelesen werden. Ziel war es, eine Klasse zu entwickeln, der man nur den `loggerName` injizieren muss. Eine Liste aller zulässigen Parameter findet sich in Anhang B.
Listing 4.3 verdeutlicht, dass es jetzt nur noch eine Zeile Code benötigt, um ein neues und konfiguriertes Logger-Objekt zu erstellen.

Listing 4.3: SignerLogger

```
import java.util.logging.Logger;

public final class SignerLogger
{

  public static Logger getInstance()
```

[13] Die `AdLogger`-Komponente ist eine Erweiterung der Java Logging Api (s. [58])

```
  {
    return UtilLogger.getInstance(Constants.P_LOGGER_NAME);
  }

}
```

Es sei angemerkt, dass man die AdLogger-Komponente auch mit Dependency Injection nutzen kann. Dies kommt, wie bereits in Abschnitt 4.2.6 angedeutet, bei der Mandantenverwaltung zum Einsatz.

4.2.9 Client

Zur Demonstration des Zeitstempeldienstes wurde zudem auch ein Client in Form eines Java-Applets entwickelt. Es können somit Daten mit einem Zeitstempel versehen werden, der anschließend anschaulich verifiziert werden kann. Sollte es sich bei den Daten um ein signiertes Dokument handeln, so wird die Signatur nicht gesondert nachgeprüft. Diese Funktionalität wird bereits durch das Codec-Modul bereitgestellt.
Die Neusignierung von Dokumenten inklusive der Verifikationsdaten kann mit dem in Abschnitt 3.4.2 beschriebenen „ArchiSig“-Konzept über Hashbäume erfolgen. Mit der erarbeiteten Implementierung können mehrere Dateien oder gar Verzeichnisse mit einem einzigen Zeitstempel versehen werden. Ebenso kann auf diese Weise die Folge verschiedener Zeitstempeln verifiziert werden. Nicht berücksichtigt wurden die angesprochenen reduzierten Archivzeitstempel, da dies über den Rahmen eines Test-Clients hinausgegangen wäre.
Das Applet ist standardmäßig unter der Adresse `http://servername:8080/tss` erreichbar.
Um auf Dateien zugreifen und Zeitstempel abspeichern zu können, muss das Applet signiert sein. Beim Laden muss man deshalb einigen Sicherheitswarnungen zustimmen, sonst wird es nicht ausgeführt.

4.2.9.1 Zeitstempel anfordern

Wie in Abbildung 4.8 dargestellt, kann man auf der ersten Registerkarte des Applets einen Zeitstempel zu beliebigen Daten anfordern. Es kann sich dabei um eine Datei, ein Verzeichnis oder um durch ein Semikolon getrennte Dateien oder Verzeichnisse handeln.
Anschließend muss man sich aus der Combobox einen der verfügbaren Hashalgorithmen aussuchen, mit dem die Daten anschließend gehasht werden.
Die Felder „User“ und „Password“ sind nur dann von Bedeutung, wenn der Timestamping-Server eine HTTP-Authentifizierung über Basic verlangt. Es wird automatisch erkannt, wenn der Server das Client-Zertifikat für die Authentifizierung anfordert. Für diesen Fall muss das Zertifikat in die Zertifikatsverwaltung von Java importiert werden.
Im Feld „URL“ kann schließlich die Adresse des Zeitstempeldienstes angegeben werden. Es steht dazu eine kleine Auswahlliste zur Verfügung. Eventuell muss die Adresse noch angepasst werden.
Standardmäßig ist der in dieser Diplomarbeit entwickelte Timestamping-Server unter folgenden Adressen erreichbar:

- `http://servername:8080/tss/tsp-servlet`
- `https://servername:8443/tss-clientcert/tsp-servlet`

Durch Klick auf „Obtain“ wird schließlich der Zeitstempel eingeholt und mit der Dateierweiterung, wie im Time Stamp Protocol vorgeschlagen (s. Abschnitt 3.3.1), `.tsr` gespeichert.

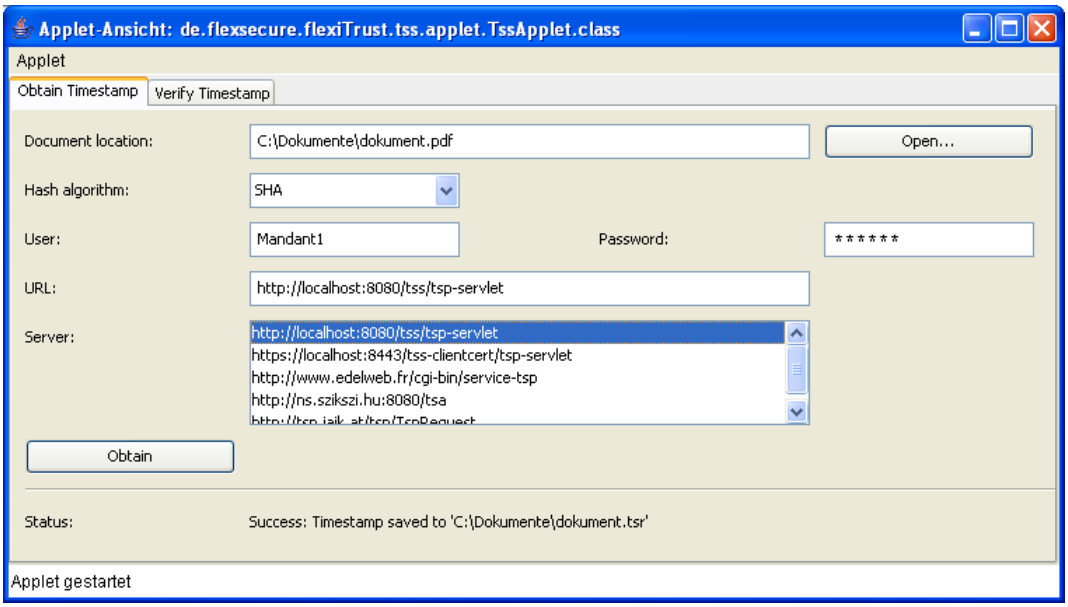

Abbildung 4.8: Anfordern eines Zeitstempels

Da im Anschluss an diese Arbeit andere Anwendungen auf dieses Applet aufbauen werden, soll an dieser Stelle ein Blick in den Quellcode geworfen werden. Listing 4.4 zeigt, wie man einen Zeitstempel zu mehreren Dateien einholen kann.

Listing 4.4: Anfordern eines Zeitstempels

```
// das das Verfahren, mit dem die Daten gehasht werden sollen
String hashAlg = ...;

// erstelle den Hashbaum
HashTree tree = new HashTree(hashAlg);

// die zu zeitstempelnden Dateien
String[] filenames = ...;

// werden dem Hashbaum hinzugefügt
for(int i = 0; i < filenames.length; i++)
{
  File file = new File(filenames[i]);
  tree.add(file);
}

// initialisiere die Zeitstempel-Anfrage
TimeStampReq req = new TimeStampReq();
MessageImprint imprint = req.getMessageImprint();
imprint.setHashedMessage(tree.getHashValue(),
  tree.getHashAlgorithm());
```

```
req.generateRandomNonce();
req.setCertReq(true);

// URL des Timestamping-Servers
String url = ...;

// HTTP Authentifizierung mit Username und Passwort
String username = ...;
String password = ...;

// Basic-Authentifizierung
ClientAuthenticator auth =
  = new ClientBaseAuthenticator(username, password);

// sende die Zeitstempel-Anfrage
TimeStampResp response =
  ClientUtils.sendTssRequest(url, auth, req);

// überprüfen, dass der Zeitstempel OK ist
TimeStampVerifier verifier = new TimeStampVerifier(response,
   req);
verifier.verify();

// und speichere den Zeitstempel
String filename = ...;
FileOutputStream fos = new FileOutputStream(filename);
fos.write(response.getEncoded());
```

4.2.9.2 Zeitstempel verifizieren

Unter der zweiten Registerkarte des Applets (s. Abbildung 4.9) kann man die bereits ausgegebenen Zeitstempel verifizieren.
Nach Eingabe des Speicherortes von Dokument (es kann sich wie bei der Anfrage von Zeitstempeln um mehrere Dokumente handeln) und Zeitstempel, wird die Überprüfung durch einen Klick auf „Verify“ angestoßen. Dabei wird angezeigt, ob der Zeitstempel

- syntaktisch korrekt ist und die Signatur verifiziert werden konnte und
- auch zu angegebenem Dokument passt.

Wie in Abbildung 4.10 dargestellt, öffnet ein Klick auf „Show“ ein neues Fenster, in dem sämtliche Inhalte des zu prüfenden Zeitstempels angezeigt werden.
Listing 4.5 zeigt, wie die Überprüfung von archivierten Zeitstempeln aussehen könnte.

Listing 4.5: Verifizieren eines Zeitstempels

```
// der Dateiname des Zeitstempels
String filename = ...;

// Zeitstempel laden
```

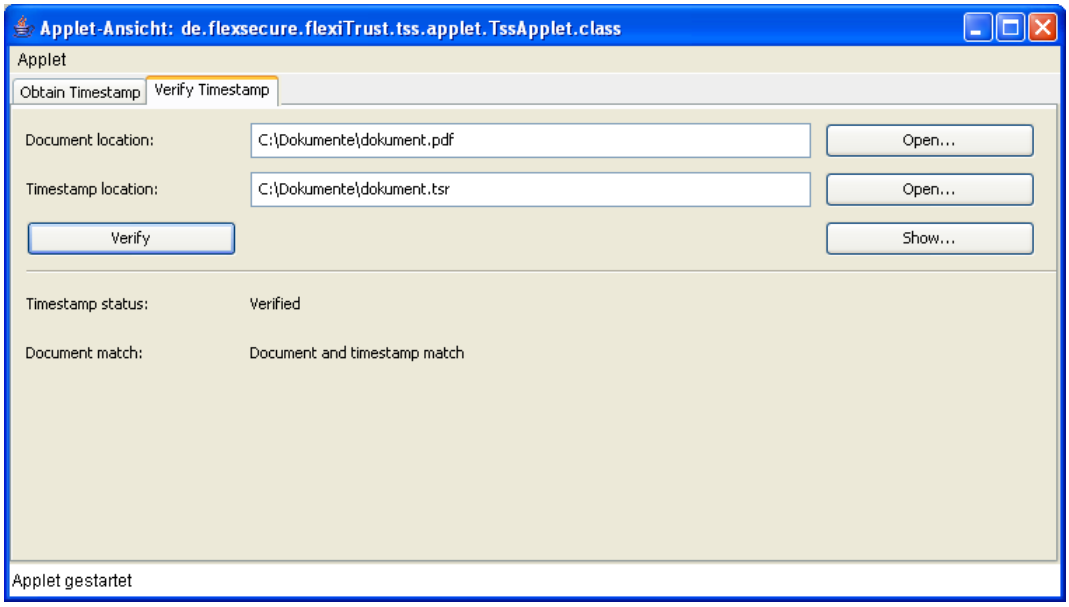

Abbildung 4.9: Verifikation des Zeitstempels

```
FileInputStream timestampIn =
  new FileInputStream(filename);
TimeStampResp tsr = new TimeStampResp(timestampIn);
timestampIn.close();

// und verifizieren
try
{
  TimeStampVerifier verifier = new TimeStampVerifier(tsr);
  verifier.verify();
}
catch (VerificationException e)
{
  // die Verifikation war nicht erfolgreich
  ...
}

// hole Hashalgorithmus und Hashwert aus Zeitstempel
MessageImprint messageImprint =
  timestamp.getTimeStampToken().getTSTInfo().getMessageImprint();
String digestAlg =
  messageImprint.getHashAlgorithm().getAlgorithmName();
byte[] digest = messageImprint.getHashedMessage().getByteArray();

// erstelle den Hashbaum
HashTree tree = new HashTree(hashAlg);
```

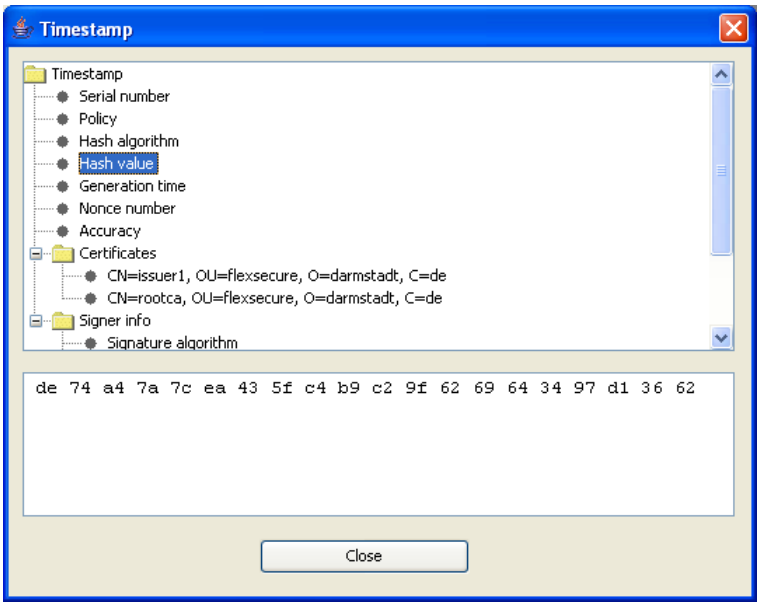

Abbildung 4.10: Detailansicht des Zeitstempels

```
// die zu zeitstempelnden Dateien
String[] filenames = ...;

// werden dem Hashbaum hinzugefügt
for(int i = 0; i < filenames.length; i++)
{
  File file = new File(filenames[i]);
  tree.add(file);
}

// die beiden Hashwerte auf Gleichheit überprüfen
if (Arrays.equals(digest, tree.getHashValue()))
{
  // der Zeitstempel passt zu den Daten
}
```

Für die Implementierung des Prüfalgorithmus wurde dabei nach den Standards TSP und CMS vorgegangen (s. Abschnitte 3.3.1.4 und 3.3.2.3). Es gilt dabei die Einschränkung, dass die Zertifikatskette unabhängig davon überprüft werden muss, da evtl. Sperrinformationen vorliegen können.
In der Praxis sollten folgende zwei Varianten für die Verifikation benutzt werden:

1. Die Zeitstempel-Antwort wird auf Korrektheit überprüft und mit der Anfrage verglichen (s. Listing 4.4).

2. die Zeitstempel-Antwort wird nur auf Korrektheit überprüft. Wenn der Zeitstempel z. B. archiviert wurde, ist die Anfrage nicht mehr verfügbar (s. Listing 4.5).

4.2.10 Sicherheitsmerkmale

An dieser Stelle sollen einige Sicherheitsvorkehrungen beschrieben werden, die an bestehenden OCSP-Server angelehnt sind (vgl. [28]).
Jede Signaturkarte muss anfangs durch eine PIN freigeschaltet werden, denn ohne erfolgreiche Aktivierung der Einheiten kann keine Signatur erstellt werden. Da das Personal des Trustcenters in der Regel nicht Eigentümer des Signaturschlüssels ist, muss das Erzeugen einer qualifizierten Signatur an dieses delegiert werden. Dafür müssen einige Sicherheitsvorkehrungen getroffen werden, die im Folgenden beschrieben werden:

1. Die Signaturkarten werden mit dem Vier-Augen-Prinzip freigeschaltet. Das heißt, dass sich zwei Benutzer mit Smartcards und PIN-Eingabe dem System gegenüber autorisieren müssen.

2. Die PINs der Signaturkarten sind verschlüsselt im System abgelegt. Weder die Bediener, Schlüsselinhaber oder Administratoren kennen die PINs.

3. Die PINs der Signaturkarten werden mit einem „m aus n“ Algorithmus geteilt. Dafür wird das Secret-Sharing-Protokoll nach Shamir (s. [12]) verwendet. Anschließend werden die Geheimnisteile mit den öffentlichen Schlüsseln der Bedienerkarten verschlüsselt.

4. Die Signaturkomponente und die Mandantenverwaltung verifiziert die Gültigkeit der Zertifikatskette der Signaturkarten. Auf diese Weise können nicht unbemerkt Zertifikate ausgetauscht werden.

4.3 Tests

Im Rahmen dieser Diplomarbeit wurden die wesentlichen Kernfunktionalitäten des Timestamping-Servers mit Tests abgedeckt.

4.3.1 Funktionale Tests

Die funktionalen Tests wurden mit JUnit[14] durchgeführt, wodurch sichergestellt wurde, dass die oben beschriebenen Komponenten einwandfrei arbeiten. JUnit ist ein Testframework für Java-Programme, das sich insbesondere für das automatisierte Testen von Klassen oder Methoden eignet. Für eine Einführung in JUnit sei auf [61] verwiesen.
In diesem Zusammenhang wurde getestet, dass

- die Authentifizierung der Mandanten korrekt funktioniert.
  Dabei wurde die Identifikation der Mandanten über ein Client-Zertifikat wegen den unterschiedlichen Konfigurationsmöglichkeiten besonders geprüft.

- bei der Signaturkomponente keine Signiereinheiten „verloren“ gehen.
  Da es im `SignerService` zwei Listen mit den jeweils verfügbaren und verwendeten Signatureinheiten gibt, muss sichergestellt sein, dass jede reservierte Einheit auch wieder freigegeben wird.

- das Hashen mehrerer Dokumente über einen Hashbaum korrekt aufgebaut wird.

[14] `http://www.junit.org`

Die Funktionsweise der EJBs wurde hauptsächlich an den ausgegebenen Zeitstempeln getestet. Hierfür wurden Anfragen generiert, die Fehlermeldungen produzieren, z. B. wenn die Policy oder das Hashverfahren nicht akzeptiert wurde. Fehlermeldungen aus der darunterliegenden Signaturkomponente können durch Einstellungen provoziert werden. Dazu kann man Mandanten erzeugen, die über keine einzige Signatureinheit verfügen oder die den Signaturalgorithmus nicht unterstützen. In allen Fällen wurde eine adäquate Fehlermeldung zurückgegeben.
Im Rahmen des Challenge PKI Projekts[15] wurde eine Testumgebung für Zeitstempel-Clients entwickelt, die bereits 54 vorgefertigte Testfälle zur Verfügung stellt. Es ist außerdem möglich, diese Testdatenbank nach belieben zu erweitern. Weitere Informationen zur Installation finden sich unter [39] und [40].
Wie bereits in Abschnitt 4.2.9.2 beschrieben, wurde für das Verifizieren der Zeitstempel eine Komponente geschrieben, die gegen die Testumgebung des Challenge PKI Projekts getestet wurde. Damit ist der entwickelte Verifizierer überprüfbar streng. Außerdem wurden mit der Prüfkomponente folgende Zeitstempeldienste erfolgreich verifiziert:

- OpenTSA[16]
- EdelWeb Experimental Time Stamping Service[17]
- Demo Time Stamping Authority des Instituts für Angewandte Informationsverarbeitung und Kommunikationstechnologie (IAIK) der TU Graz[18]
- DigiStamp[19]

Die Zeitstempel, die vom Timestamping-Server dieser Diplomarbeit ausgestellt wurden, konnten anschließend von folgenden Clients verifiziert werden:

- OpenTSA
- Evaluierungsversion des IAIK Crypto Toolkits der TU Graz[20]
- Bouncy Castle Crypto API[21]

4.3.2 Lasttest

Zusätzlich zu den funktionellen Tests wurden auch Lasttests durchgeführt. Dafür wurde das Werkzeug JMeter[22] verwendet, um das Verhalten des Timestamping-Servers unter Last zu untersuchen.
Die Wahl fiel auf dieses Werkzeug, da es intuitiv zu bedienen ist. Gleichzeitig ist JMeter in hohem Maße konfigurierbar und erweiterbar: Neben bereits vielen vorgefertigten Klassen für z. B. HTTP- oder Datenbankanfragen gibt es die Möglichkeit, eigene Anfragen zu definieren. Weitere Informationen zu JMeter finden sich unter [11].
Wie in Abbildung 4.11 dargestellt, wurde eine eigene Zeitstempel-Anfrage entwickelt, die sich bezüglich mehrerer Optionen konfigurieren lässt.

[15] http://www.jnsa.org/mpki
[16] http://www.opentsa.org
[17] http://timestamping.edelweb.fr
[18] http://tsp.iaik.at
[19] http://www.digistamp.com/tech.htm
[20] http://jce.iaik.tugraz.at
[21] http://www.bouncycastle.org
[22] http://jakarta.apache.org/jmeter

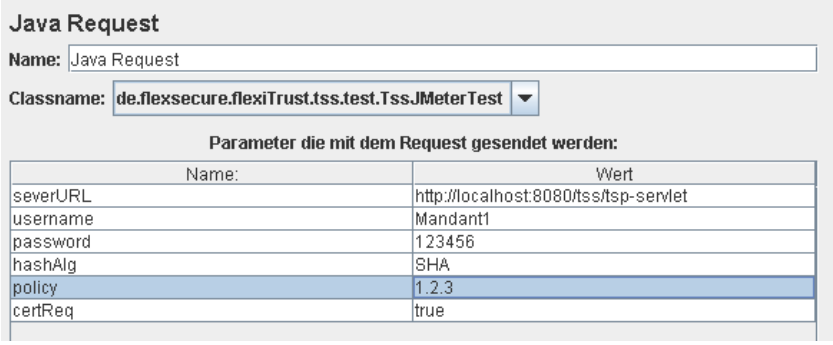

Abbildung 4.11: Einstellungen der JMeter-Zeitstempel-Anfrage

In Abbildung 4.12 wurde das Verhalten des Servers mit JMeter graphisch ausgewertet. Dabei ist vor allem die Durchsatzrate (grüne Linie) von Interesse. Auf dem Testserver[23] konnten bei 300 gleichzeitigen Threads durchschnittlich 450 Zeitstempel pro Minute ausgestellt werden.
Mit dem Benutzen von Signaturkarten beschränkt sich der Durchsatz auf ca. 30 Zeitstempel pro Minute und Karte. Diese Größe kann aber durch den Anschluss von mehreren Kartenlesern skaliert werden.
An den vereinzelten Punkten und der hohen Abweichung (rote Linie) erkennt man, dass einige Anfragen sofort bearbeitet werden, andere erst nach einer Wartezeit von etwa einer Minute. Dies liegt daran, dass die Signaturanfragen nach Thread-Priorität beantwortet werden und nicht nach dem FIFO-Prinzip (s. auch Abschnitt 4.2.5).
Ebenso wurde zum Testen der Performanz des Servers ein *Profiler* verwendet. Ein Profiler ist ein Werkzeug, mit dem man die reale Ausführungszeit jeder durchlaufenen Methode messen kann. In diesem Zusammenhang wurde der JBoss Profiler[24] benutzt.

4.4 Zusammenfassung

In diesem Abschnitt soll beschrieben werden, welche Schritte genau notwendig sind, um bestehendes Framework zu erweitern.

1. Je nach Anwendungsfall müssen neue Datenstrukturen entwickelt werden, die auf dem Fraunhofer Codec aufbauen.

2. Sollen ASN.1-kodierte Anfragen über HTTP verschickt werden, kann das `Asn1Servlet` verwendet werden. Es empfiehlt sich, nur neue Klassen davon abzuleiten, wenn es auch wirklich benötigt wird, z. B. wenn mehrere EJBs angesprochen werden müssen.

3. Die EJBs sind in der Regel von Grund auf neu zu entwickeln. Dabei ist zu beachten, dass das EJB, das vom `Asn1Servlet` angesprochen wird, das Interface `Asn1Processor` implementieren muss.

4. Je nach benötigten Datenstrukturen für die Mandantenverwaltung muss eine weitere Klasse von `Client` abgeleitet werden. Soll außerdem eine Authentifizierung über Benutzername und Passwort bzw. über Client-Zertifikate möglich sein, muss diese Klasse die Interfaces `PasswordClient` bzw. `CertificateClient` implementieren.

[23] 2,4 GHz, 1 GB RAM, Signatur über Softtoken mit SHA1withRSA 1024 Bit
[24] `http://labs.jboss.org/portal/jbossprofiler`

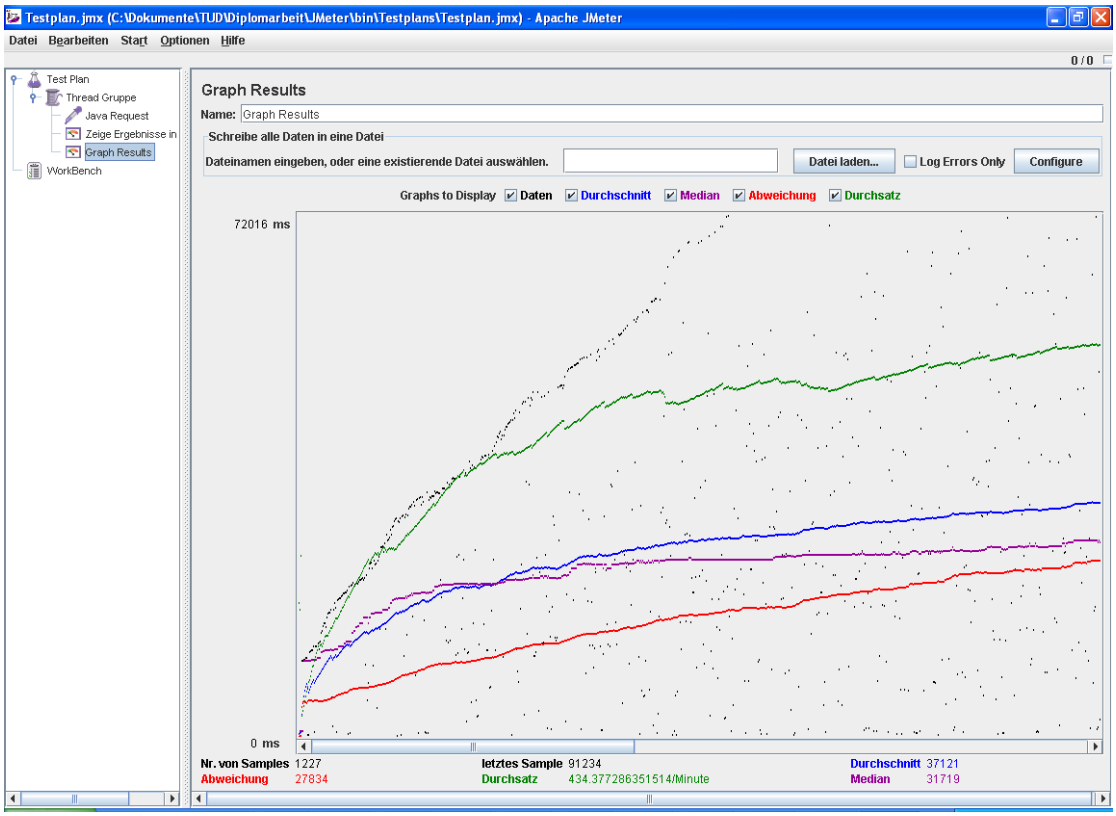

Abbildung 4.12: Lastverhalten des Timestamping-Servers

5. Zuletzt müssen besagte Komponenten konfiguriert werden. In Anhang B finden sich Beispiele von Konfigurationsdateien.

Kapitel 5

Fazit

In dieser Diplomarbeit wurden die wesentlichen Anforderungen an Zeitstempeldienste, die qualifizierte Zeitstempel gemäß des Signaturgesetzes [6] ausstellen, herausgestellt. Besonders im qualifizierten Bereich werden hohe Sicherheitsanforderungen an Zertifizierungsdiensteanbieter von Seiten der Gesetzgebung gestellt. Dies betrifft die eingesetzten Server sowie ihr Umfeld einschließlich des Personals.

Ebenso wurden verschiedene Anwendungsfälle (Massensignatur, Langzeitarchivierung, intervallqualifizierte Zeitstempel) und technische Lösungsmöglichkeiten diskutiert.

Bedingt durch den Verlust der Sicherheitseignung von kryptographischen Algorithmen müssen Signaturen von Zeit zu Zeit mit einem Zeitstempel erneuert werden. Auf rechtlicher Seite konnten zu dieser Problematik noch kaum Erfahrungen gesammelt werden. Das „ArchiSig“-Projekt erarbeitete speziell für die Langzeitarchivierung von elektronisch signierten Dokumenten ein Konzept, das durch simulierte Fallstudien vor Gericht rechtlich bewertet wurde.

Der Schwerpunkt dieser Arbeit bildet die Entwicklung eines mandantenfähigen Timestamping-Servers zur Ausstellung von qualifizierten Zeitstempeln. Es wurden Anforderungen, die sich zum Teil aus den gesetzlichen Vorgaben ergeben, gesammelt und umgesetzt. Während der Konzeptionsphase wurden verschiedene Designentscheidungen getroffen, wobei viel Wert auf die Funktionalität und Robustheit als auch auf die Erweiterbarkeit gelegt wurde. Entstanden ist ein modular aufgebauter Timestamping-Server, dessen Komponenten auch in anderen Kontexten eingesetzt werden können. Gerade im Bereich der Massensignatur ergeben sich Anwendungsgebiete wie z. B. OCSP-Server oder die elektronische Rechnungsstellung, für die das entwickelte mandantenfähige Signatursystem eine Grundlage bietet.

Einige Komponenten können ohne Änderung im Quellcode von anderen Diensten mitbenutzt werden. Dazu zählen das Servlet zur Annahme von ASN.1-kodierten Daten sowie die Signaturkomponente zur Erzeugung von rechtsgültigen Signaturen.

Ebenso ist die Mandantenverwaltung durch den Einsatz des Spring-Frameworks voll konfigurierbar. Bereitgestellte Mandanten-Klassen können mit minimalem Aufwand an eigene Bedürfnisse angepasst werden, wobei bestehende Schnittstellen zur Authentifizierung von Mandanten integriert werden können.

Die Erzeugung der Zeitstempel erfolgt in der Business-Logik, die mit Hilfe von EJBs realisiert wird und die in einem anderen Kontext neu entwickelt werden muss.

Die Architektur des Servers stützt sich auf den Einsatz von etablierten Frameworks (Java EE, Spring) und Design Patterns. Java EE hat den Vorteil, dass es sich um einen weit verbreiteten Standard handelt und besonders für sicherheitskritische Anwendungen nötige Schnitt-

stellen anbietet. Insbesondere vereinfacht die aktuelle EJB 3.0-Spezifikation die Entwicklung von Multitier-Architekturen spürbar. Diese Arbeit gibt ein Beispiel, wie Spring in Java EE-Anwendungen integriert werden kann und wie Abhängigkeiten zwischen Objekten über Dependency Injection aufgelöst werden können. Aufgrund der angesprochenen Erweiterbarkeit des Servers wurde die Grundlage für weitere Arbeiten geschaffen, denen die vollständige Bandbreite der Java EE-Welt offen steht.
Schließlich wurden der Timestamping-Server und seine Komponenten verschiedenen Tests unterzogen, um die Funktionalität als auch Interoperabilität, d. h. die Konformität zu internationalen technischen Standards, zu gewährleisten. Die Leistungsfähigkeit des Servers wurde mit Lasttests bestätigt.
Damit der Zeitstempeldienst bzw. das entwickelte Framework auch in der Praxis eingesetzt werden dürfen, sind laut Gesetzgeber einige Formalitäten nötig. Dazu gehört u. a. die Ausarbeitung eines Sicherheitskonzepts und eine technische Prüfung.

Anhang A

Installation

A.1 Benötigte Software

Im Folgenden werden die Softwarekomponenten beschrieben, die für das erfolgreiche Erstellen des Timestamping-Servers benötigt werden.
Die erforderliche Software setzt sich zusammen aus den Libraries, die im CVS zu finden sind sowie anderen Softwarepaketen, die im Internet erhältlich sind.
Aus dem CVS müssen die folgenden drei Module mit den angegebenen jar-Archiven ausgecheckt werden:

```
/
├── all
│   ├── codec.jar
│   └── FlexiProvider.jar
├── dist
│   ├── cardman.jar
│   ├── fs_codec.jar
│   ├── fs_util.jar
│   ├── leanca.jar
│   ├── p11.jar
│   ├── pass_sharing.jar
│   ├── pin_sharing.jar
│   └── secret_sharing.jar
└── tss-ejb
```

Diese beiden Module *ant* und *dist* müssen sich im gleichen Verzeichnis wie das Modul *tss-ejb* befinden.
Darüberhinaus werden folgende Softwarepakete benötigt:

- Sun JDK 1.5 (`http://java.sun.com/javase`)
- JBoss 4.0.4 (`http://labs.jboss.com/portal/jbossas`)
  Für diese Version muss für EJB 3-Support der Installer verwendet werden.
- Ant 1.6.x (`http://ant.apache.org`)
- Spring Spring 2.0.x (`http://www.springframework.org`)

A.2 Verzeichnisstruktur

Das Modul *tss-ejb* besteht aus folgenden Verzeichnissen

```
/tss-ejb
├── build ..... Das Verzeichnis, in dem die erstellten Archive abgelegt werden
├── conf ..... Die einzelnen Konfigurationsdateien
├── data ..... Das Datenverzeichnis für die Mandanten
│   ├── mandant1
│   │   ├── cacerts
│   │   └── signercerts
│   └── mandant2
│       ├── cacerts
│       └── signercerts
├── doc ..... Dokumentation
├── etc
│   ├── certificates ..... Einige selbsterzeugte Zertifikate
│   └── deploymentdescriptors ..... Einige Deployment-Deskriptoren
├── lib ..... Verzeichnis, in dem einige externe jar-Archive liegen
├── src ..... Das Verzeichnis mit den Quellcodedateien
└── web ..... Die Webseite, die mit dem Timestamping-Server deployed wird
```

A.3 Deployment

Das Ausführen des Ant-Build-Skript erzeugt folgende Archive, die in das Deploy-Verzeichnis von JBoss gelegt werden:

```
tss-ejb.ear
├── META-INF
│   └── application.xml
├── beans.jar ..... Darin befinden sich alle kompilierten Serverklassen
│   └── de
│       └── ..
├── web-basic.war
│   ├── WEB-INF
│   │   ├── web.xml
│   │   └── jboss-web.xml
│   ├── codec.jar
│   ├── index.htm
│   ├── FlexiProvider.jar
│   └── tssapplets.jar
└── web-clientcert.war
    └── WEB-INF
        ├── web.xml
        └── jboss-web.xml
```

Es sei bemerkt, dass die Archive `codec.jar` und `tssapplets.jar` signiert werden müssen. Zurzeit geschieht dies über das Server-Zertifikat.

A.4 Umgebungsvariablen

A.4.1 Kompilieren

Für den Erstellungsprozess werden folgende Umgebungsvariablen benötigt:

JAVA_HOME Der Pfad zum Java JRE/JDK

JBOSS_HOME Der Pfad von JBoss

A.4.2 Ausführen

Listing A.1 gibt an, wie der Timestamping-Server gestartet werden kann. Folgende Variablen werden definiert:

FT_CONFIGDIR Das Verzeichnis, in dem die verschiedenen Konfigurationsdateien liegen

FT_DATADIR Das Basisverzeichnis, in dem sich die Daten der Mandanten befinden

JAVA_OPTS Mit dieser Variable werden die beschriebenen Optionen der JVM mitgegeben, darunter auch zum Pfad der Konfigurationsdatei.

LD_LIBRARY_PATH Die Anbindung von Kartenlesern geschieht über Nicht-Java-Bibliotheken. Sollten diese nicht in Standardverzeichnissen liegen, müssen die ensprechenden Verzeichnisse in dieser Umgebungsvariable definiert werden.

Listing A.1: start.sh

```
#!/bin/bash

export JAVA_HOME=/usr/local/tssServer/jdk1.5.0_10
export JBOSS_HOME=/usr/local/tssServer/jboss-4.0.4.GA
export FT_CONFIGDIR=$JBOSS_HOME/server/default/conf/tss-ejb
export FT_DATADIR=/usr/local/tssServer/data

export JAVA_OPTS="-DFT_DATADIR=$FT_DATADIR -DFT_CONFIGDIR=
    $FT_CONFIGDIR -DconfigFile=$FT_CONFIGDIR/config.properties"

$JBOSS_HOME/bin/run.sh
```

Anhang B

Konfiguration

Im Rahmen des Timestamping-Servers gibt es mehrere Konfigurationsdateien, die im Einzelnen näher beschrieben werden.

B.1 config.properties

Properties-Dateien können, wie in Tabelle B.1 beschrieben, in andere Properties-Dateien eingebunden werden. Dadurch können Erweiterungen des Servers modular verwaltet werden.

EIGENSCHAFT	BESCHREIBUNG
include	Bindet die durch ";" getrennten Properties-Dateien ein. Dabei wird wiederum überprüft, ob andere Dateien rekursiv einzufügen sind.

Tabelle B.1: Allgmeine Konfiguration

Listing B.1: config.properties

```
include=${FT_CONFIGDIR}/tss.properties;${FT_CONFIGDIR}/signer.
    properties
```

B.2 Logger

Insgesamt gibt es zwei Logger. Der `TssLogger` wird in den eigentlichen Funktionen des Zeitstempeldienstes verwendet. Der `SignerLogger` kommt im `SignerService` zum Einsatz und loggt z. B. die Ereignisse der Kartenleser (u. a. das Erkennen von neuen Karten).
Der `SignerLogger` wird analog Tabelle B.2 konfiguriert (das Präfix lautet `signerLogger`).

B.3 Asn1Servlet

Da mehrere Asn1Servlets mit unterschiedlichen Eigenschaften in der gleichen VM laufen müssen, ist das Präfix der Properties standardmäßig der Servletname, der in der Datei web.xml definiert wurde (hier `TspServlet`). Tabelle B.3 beschreibt die möglichen Konfigurationen.

EIGENSCHAFT	BESCHREIBUNG
tssLogger.logLevel	Log-Level (TRACE, DEBUG, INFO, ...)
tssLogger.logRotationSize	Maximale Größe der Log-Dateien
tssLogger.logRotationCount	Maximale Anzahl der Log-Dateien
tssLogger.logFile	Dateiname und Pfad der Log-Datei
tssLogger.logToConsole	Falls true, wird zusätzlich auf die Konsole geloggt
tssLogger.logPasswords	Falls true, wird versucht, alle Passwörter unkenntlich zu machen

Tabelle B.2: Konfiguration der Logger

EIGENSCHAFT	BESCHREIBUNG
TspServlet.requestContentTypes	Die durch Komma (,) getrennten Werte repräsentieren die akzeptierten ContentTypes der Anfrage. Eingehende ContentTypes, die hier nicht aufgelistet sind, werden abgewiesen.
TspServlet.responseContentType	ContentType der Antwort
TspServlet.requestType	Der vollständige Klassennamen der ASN1-Struktur, in der die Anfrage kodiert wurde.
TspServlet.processor	Der JNDI-Lookup-Name des EJBs, das die Anfrage weiterverarbeitet
TspServlet.logger	Der Name des Loggers
TspServlet.maxContentLength	Maximale ContentLength. Liegt die eingehende ContentLength über diesem Wert, so wird die Anfrage abgewiesen

Tabelle B.3: Konfiguration des Asn1Servlets

B.4 Signaturkomponente

Listing B.2: signer.properties

```
#-------------------------------------------
# Logging
#-------------------------------------------

signerLogger.logLevel=DEBUG
signerLogger.logRotationSize=0
signerLogger.logRotationCount=1
signerLogger.logFile=signer.log
signerLogger.logToConsole=true
signerLogger.logPasswords=false

#-------------------------------------------
# SignerService
#-------------------------------------------

cardManager.drivers=de.flexsecure.flexiTrust.cardman.
    SoftCardDriver:p12file=issuer1.p12,certfile=issuer1.der,
    tokenID=Issuer 1's cardManager.pin=123456;de.flexsecure.
```

EIGENSCHAFT	BESCHREIBUNG
cardManager.drivers	Treiber-String für Cardman-Bibliothek
cardManager.hotplug	Hotplugging für Signaturkarten ein-/ausschalten
cardManager.pin	PIN, falls kein PIN-Sharing verwendet wird
cardManager.smKey	SM-Key, falls kein PIN-Sharing verwendet wird
cardManager.scanInterval	Intervall, in dem nach neuen Karten gescannt wird. Standardwert 1000 (= 1s)
clients.configurationFile	Dateiname der Konfigurationsdatei der Mandanten
clients.signatureHashAlg	Der Hash-Algorithmus, der für die Signatur der Zeitstempel benutzt werden soll

Tabelle B.4: Konfiguration der Signaturkomponente

```
    flexiTrust.cardman.SoftCardDriver:p12file=issuer2.p12,
    certfile=issuer2.der,tokenID=Issuer 2's cardManager.pin
    =123456
#cardManager.drivers=de.flexsecure.flexiTrust.cardman.
    starcosapdu.StarcosApduCTApiCardDriver:JceProvider=FlexiCore
    ,CtInterfaceLib=ct,CtInterfacePorts=1,appfile=${FT_CONFIGDIR
    }/card_dgn_issuer.xml
cardManager.smKey=5DAD7ABF808002D3F24F9E80804C8C3D
cardManager.pin=123456

#-----------------------------------------
# Client configuration
#-----------------------------------------

clients.configurationFile=${FT_CONFIGDIR}/clients.xml
clients.signatureHashAlg=SHA
```

B.5 Business-Objekte

Listing B.3: tss.properties

```
#-----------------------------------------
# Asn1Servlet
#-----------------------------------------

TspServlet.requestContentTypes=application/timestamp-query
TspServlet.responseContentType=application/timestamp-reply
TspServlet.requestType=de.flexsecure.flexiTrust.tss.tsp.
    TimeStampReq
TspServlet.processor=tss-ejb/TspProcessorBean/local
TspServlet.logger=tssLogger

#-----------------------------------------
# Logging
```

EIGENSCHAFT	BESCHREIBUNG
tspProcessor.acceptedPolicies	Durch Komma (,) getrennte Werte der akzeptierte Policies. Steht in einem Request eine Policy, die nicht hier aufgeführt wird, wird die Anfrage zurückgewiesen
tspProcessor.defaultPolicy	Wird im Request keine Policy angegeben, so wird die Response mit der Defaultpolicy ausgestellt
tspProcessor. acceptedHashAlgorithms	Durch Komma (,) getrennte Werte der akzeptierten Hash-Algorithmen. Wurde in einem Request ein anderer Algorithmus benutzt, der nicht hier aufgelistet ist, wird die Anfrage zurückgewiesen
tspProcessor.accuracy.seconds	Der Sekundenanteil der Genauigkeit des Zeitgebers
tspProcessor.accuracy.millis	Der Millisekundenanteil der Genauigkeit des Zeitgebers (0-999)
tspProcessor.accuracy.micros	Der Mikrosekundenanteil der Genauigkeit des Zeitgebers (0-999)
tspProcessor.sql.create	SQL-Statement, um eine Sequenz für die Seriennummer der Zeitstempel zu erstellen. Wird beim Deployen ausgeführt, falls vorhanden
tspProcessor.sql.drop	SQL-Statement, um eine Sequenz zu droppen. Nicht empfehlenswert
tspProcessor.sql.query	SQL-Statement, um den nächsten Wert der Sequenz abzufragen
tspProcessor.sql.datasource	Lookup-Name für die Datenquelle. Standardwert java:/DefaultDS

Tabelle B.5: Konfiguration der EJBs

```
#-------------------------------------------------

tssLogger.logLevel=DEBUG
tssLogger.logRotationSize=0
tssLogger.logRotationCount=1
tssLogger.logFile=tss.log
tssLogger.logToConsole=true
tssLogger.logPasswords=false

#-------------------------------------------------
# TspProcessor
#-------------------------------------------------

tspProcessor.acceptedPolicies=1.2.3
tspProcessor.defaultPolicy=1.2.3
tspProcessor.acceptedHashAlgorithms=RIPEMD160,SHA,SHA224,SHA256
    ,SHA384,SHA512
tspProcessor.accuracy.seconds=1
tspProcessor.accuracy.millis=0
tspProcessor.accuracy.micros=0
```

```
tspProcessor.sql.create=CREATE SEQUENCE tss_sequence_seq_id AS
    INTEGER START WITH 1 INCREMENT BY 1; CREATE TABLE
    tss_sequence (tss_sequence_seq_id INTEGER NOT NULL); INSERT
    INTO tss_sequence VALUES (1);
#tspProcessor.sql.drop=DROP SEQUENCE tss_sequence_seq_id; DROP
    TABLE tss_sequence;
tspProcessor.sql.query=SELECT NEXT VALUE FOR
    tss_sequence_seq_id FROM tss_sequence;
```

B.6 Mandantenverwaltung

Die in Tabelle B.6 angegebenen Eigenschaften entsprechen eins zu eins den Eigenschaften der entsprechenden Java-Klassen.
Die Konfiguration der Klassen geschieht per XML-Dateien über das Spring-Framework (`http://www.springframework.org`). In Listing B.4 werden zwei Mandanten erstellt, der Eine mit einer Authentifizierung über Benutzername und Passwort und der Andere mit Authentifizierung über Client-Zertifikat.

Listing B.4: clients.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>

<bean id="Mandant1" class="de.flexsecure.flexiTrust.tss.utils.
    TssUserPassClient" init-method="init">
  <property name="name" value="Mandant1" />
  <property name="id" value="1" />
  <property name="signerCertsDir" value="${FT_DATADIR}/mandant1
      /signercerts" />
  <property name="caCertsDir" value="${FT_DATADIR}/mandant1/
      cacerts" />
  <property name="clientLogger" ref="Mandant1Logger" />
  <property name="password" value="123456" />
  <property name="certificateVerifier" ref="CHAIN" />
  <!--<property name="pinSharingDir" value="${FT_DATADIR}/
      mandant1/pinshares" />-->
</bean>

<bean id="Mandant2" class="de.flexsecure.flexiTrust.tss.utils.
    TssCertClient" init-method="init">
  <property name="name" value="Mandant2" />
  <property name="id" value="2" />
  <property name="signerCertsDir" value="${FT_DATADIR}/mandant2
      /signercerts" />
```

EIGENSCHAFT	BESCHREIBUNG
id	Eindeutige ID des Mandanten
name	Eindeutiger Name des Mandanten
certificateVerifier	Je nach Gültigkeitsmodell kann ein Verifizierer gesetzt werden, mit dem die Zertifikate überprüft werden sollen. Standardmäßig wird nach dem Kettenmodell geprüft.
signerCertsDir	Verzeichnis, in dem die Signaturzertifikate liegen
caCertsDir	Verzeichnis, in dem die CA-Zertifikate liegen
pinSharingDir	Verzeichnis, in dem die PIN-Shares liegen
signatureHashAlg	Das Hashverfahren, das für die Signatur verwendet wird. Wenn die Eigenschaft nicht gesetzt ist, wird die Eigenschaft clients.signatureHashAlg verwendet.
clientLogger	Ein Logger-Objekt, mit dem die ausgestellten Zeitstempel für jeden Mandanten geloggt werden
policy	Die Policy des Zeitstempeldienstes. Wenn die Eigenschaft nicht gesetzt ist, wird die Eigenschaft tspProcessor.defaultPolicy verwendet.
acceptedPolicies	Die Policys, die vom Zeitstempeldienst von Anfragen akzeptiert werden. Wenn die Eigenschaft nicht gesetzt ist, wird die Eigenschaft tspProcessor.acceptedPolicies verwendet.
acceptedHashAlgs	Die akzeptierten Hashverfahren des Zeitstempeldienstes. Wenn die Eigenschaft nicht gesetzt ist, wird die Eigenschaft tspProcessor.acceptedHashAlgorithms verwendet.
password	Das Passwort um einen Mandanten zu authentifizieren (bei Authentifizierung über Namen und Passwort)
principals	Eine Liste von Principals für die Authentifizierung über Client-Zertifikate. Ein Principal besteht aus (jeweils ein inklusives Oder) **subject** Ein regulärer Ausdruck, dem der Subject des Client-Zertifikats entsprechen muss **issuer** Ein regulärer Ausdruck, dem der Issuer des Client-Zertifikats entsprechen muss **serial** Eine Seriennummer, die der Seriennummer des Client-Zertifikats entsprechen muss

Tabelle B.6: Eigenschaften der Mandantenklassen

```
  <property name="caCertsDir" value="${FT_DATADIR}/mandant2/
     cacerts" />
  <!--<property name="pinSharingDir" value="${FT_DATADIR}/
     mandant2/pinshares" />-->
  <property name="clientLogger" ref="Mandant2Logger" />
  <property name="certificateVerifier" ref="CHAIN" />
  <property name="principals">
    <list>
      <bean class="de.flexsecure.flexiTrust.tss.jmx.auth.
         CertPrincipal">
        <property name="subject" value="CN=client1,␣OU=
           flexsecure,␣O=darmstadt,␣C=de" />
      </bean>
    </list>
  </property>
</bean>

<bean id="Mandant1Logger" class="de.flexsecure.flexiTrust.
   logging.AdLogger" factory-method="setLogger">
  <constructor-arg value="Mandant1Logger" />
  <constructor-arg value="de.flexsecure.flexiTrust.tss" />
  <constructor-arg><null /></constructor-arg>
  <constructor-arg ref="DebugInfoLevel" />
  <constructor-arg value="${FT_DATADIR}/mandant1/mandant.log" /
     >
  <constructor-arg value="0" />
  <constructor-arg value="1" />
  <constructor-arg value="false" />
  <constructor-arg value="false" />
</bean>

<bean id="Mandant2Logger" class="de.flexsecure.flexiTrust.
   logging.AdLogger" factory-method="setLogger">
  <constructor-arg value="Mandant2Logger" />
  <constructor-arg value="de.flexsecure.flexiTrust.tss" />
  <constructor-arg><null /></constructor-arg>
  <constructor-arg ref="DebugInfoLevel" />
  <constructor-arg value="${FT_DATADIR}/mandant2/mandant.log" /
     >
  <constructor-arg value="0" />
  <constructor-arg value="1" />
  <constructor-arg value="false" />
  <constructor-arg value="false" />
</bean>

<bean id="DebugInfoLevel" class="de.flexsecure.flexiTrust.
   logging.AdLevel" factory-method="resolveLevel">
  <constructor-arg value="INFO" />
```

```
</bean>

<bean id="SHELL" class="de.flexsecure.flexiTrust.tss.jmx.
    certutils.CertificateVerifierFactory" factory-method="
    createCertifiaceVerifier">
  <constructor-arg value="SHELL" />
</bean>

<bean id="CHAIN" class="de.flexsecure.flexiTrust.tss.jmx.
    certutils.CertificateVerifierFactory" factory-method="
    createCertifiaceVerifier">
  <constructor-arg value="CHAIN" />
</bean>

</beans>
```

B.7 JBoss

B.7.1 server.xml

Konfiguration von JBoss für HTTP(S)

Listing B.5: server.xml

```
<Server>

   <Service name="jboss.web"
      className="org.jboss.web.tomcat.tc5.StandardService">

      <!-- A HTTP/1.1 Connector on port 8080 -->
      <Connector port="8080" address="${jboss.bind.address}"
         maxThreads="250" strategy="ms" maxHttpHeaderSize="8192
            "
         emptySessionPath="true"
         enableLookups="false" redirectPort="8443" acceptCount=
            "100"
         connectionTimeout="20000" disableUploadTimeout="true"/
            >

      <!-- SSL/TLS Connector configuration -->
      <Connector port="8443" address="${jboss.bind.address}"
           maxThreads="100" strategy="ms" maxHttpHeaderSize="
              8192"
           emptySessionPath="true"
           scheme="https" secure="true" clientAuth="false"
           keystoreFile="${FT_CONFIGDIR}/tomcat.keystore"
           keystorePass="123456" sslProtocol = "TLS" />
```

```
        ...

    </Service>

</Server>
```

Für den Betrieb für HTTPS muss ein Keystore (entweder JKS oder PKCS12) vorbereitet werden. Weitere Anweisungen finden sich unter `http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html`

B.7.2 login-config.xml

In der Datei login-config.xml können die Login-Module konfiguriert werden. Standardmäßig sind `PasswordLoginModule`, `BaseCertLoginModule` `CertLoginModule` und aktiviert.
Die Login-Module `PasswordLoginModule` und `CertLoginModule` authentifizieren die Mandanten über ihren Namen und das Passwort bzw. über ein Client-Zertifikat, wie es in clients.xml definiert wurde.
`BaseCertLoginModule` kümmert sich um die Client-Authentifizierung über SSL. Dabei sind zwei Dinge zu beachten: Die Client-Zertifikate müssen von einer vertrauenswürdigen CA ausgegeben worden sein. Dazu muss man evtl. die Zertifikatskette in den Truststore des Servers importieren:

```
keytool -import -keystore $JAVA_HOME/lib/security/cacerts
  -storepass changeit -alias rootca -file rootca.pem
  -trustcacerts
```

Der TSS hat darüber hinaus einen eigenen Client-Zertifikat-Verifier, der alle Client-Zertifikate akzeptiert (ansonsten würden sie anhand des Truststores der Security-Domain validiert werden). Weitere Informationen finden sich unter `http://wiki.jboss.org/wiki/Wiki.jsp?page=BaseCertLoginModule`

Listing B.6: login-config.xml

```
<?xml version='1.0'?>
<!DOCTYPE policy PUBLIC
  "-//JBoss//DTD JBOSS Security Config 3.0//EN"
  "http://www.jboss.org/j2ee/dtd/security_config.dtd">

<policy>

  <application-policy name="tss-ejb">
    <authentication>

      <!-- Login via username and password-->
      <login-module code="de.flexsecure.flexiTrust.tss.jmx.auth
          .PasswordLoginModule" flag="sufficient" />

      <!-- Login via client certs-->
      <login-module code="org.jboss.security.auth.spi.
          BaseCertLoginModule" flag="requisite">
```

```
        <module-option name="password-stacking">useFirstPass</
            module-option>
        <module-option name="verifier">de.flexsecure.flexiTrust
            .tss.utils.AllX509CertificateVerifier</module-option
            >
      </login-module>

      <!-- Match the client certs-->
      <login-module code="de.flexsecure.flexiTrust.tss.jmx.auth
          .CertLoginModule" flag="required">
        <module-option name="password-stacking">useFirstPass</
            module-option>
      </login-module>

    </authentication>
  </application-policy>

  ...

</policy>
```

B.7.3 standardjboss.xml

Die Datei `standardjboss.xml` enthält die Konfigurationen für die EJB-Container, z. B. die Anzahl der Instanzen von Stateless Session Beans (nur diese werden im TSS verwendet)

Listing B.7: standardjboss.xml

```
<jboss>
  <container-configurations>

    ...

    <container-configuration>
      <container-name>Standard Stateless SessionBean</container
          -name>
      ...
      <container-pool-conf>
        <MaximumSize>100</MaximumSize>
      </container-pool-conf>
    </container-configuration>

  ...

  </container-configurations>
</jboss>
```

B.7.4 Datenquellen

Standardmäßig arbeitet der TSS mit der eingebauten HSQL-Datenbank. Vorlagen für andere Datenbanken befinden sich im Verzeichnis `JBOSS_HOME/docs/examples/jca` und müssen anschließend angepasst ins Deploy-Verzeichnis gelegt werden. Zuletzt muss der Eintrag `tspProcessor.sql.datasource` in der Datei *tss.properties* angepasst werden.

Literaturverzeichnis

[1] *Begründung zum Entwurf eines Gesetzes über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften in der Fassung des Kabinettbeschlusses vom 16. August 2000*.

[2] *Gesetz zur digitalen Signatur (Signaturgesetz - SigG)*. Artikel 3 des Informations- und Kommunikationsdienste-Gesetz (IuKDG) vom 22. Juli 1997, BGBl. I S. 1870-1872, 1997.

[3] *Verordnung zur digitalen Signatur (Signaturverordnung - SigV), vom 22. Oktober 1997*. Artikel 3 des Informations- und Kommunikationsdienste-Gesetz (IuKDG) vom 22. Juli 1997, BGBl. I Nr. 70, 27. Oktober 1997, S. 2498, 1997.

[4] *Richtlinie 1999/93/EG des Europäischen Parlaments und des Rates vom 13. Dezember 1999 über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen* . Amtsblatt der Europäischen Gemeinschaften, Januar 2000.

[5] *Begründung zum Entwurf einer Verordnung zur elektronischen Signatur in der Fassung des Kabinettbeschlusses vom 24. 10. 2001*. 2001.

[6] *Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften, vom 16.05.2001*. BGBl 2001 Teil I Nr. 22, Mai 2001.

[7] *Verordnung zur elektronischen Signatur (Signaturverordnung - SigV) vom 16. November 2001*. BGBl 2001 Teil I Nr. 59, November 2001.

[8] *Erstes Gesetz zur Änderung des Signaturgesetzes (1. SigGÄndG), vom 4. Januar 2005*. BGBl 2001 Teil I Nr. 1, Januar 2005.

[9] Adams, C., P. Cain, D. Pinkas, and R. Zuccherato: *Time-stamp protocol (TSP)*. RFC 3161, Internet Engineering Task Force, August 2001.

[10] Adams, C. and S. Farrell: *Certificate Management Protocols*. RFC 2510, Internet Engineering Task Force, March 1999.

[11] Apache Software Foundation: *Apache JMeter - User's Manual*. `http://jakarta.apache.org/jmeter/usermanual/index.html`. Stand 18. 01. 2007.

[12] Buchmann, J.: *Einführung in die Kryptographie*. Springer, 3. Auflage, 2003.

[13] Bundesamt für Eich- und Vermessungswesen: *Policy für den sicheren Zeitstempeldienst des BEV*. http://www.signatur.rtr.at/repository/csp-bev-cp-10-20061201-de.pdf, November 2006. Version 1.0.

[14] Bundesamt für Sicherheit in der Informationstechnik: *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV, Signatur-Interoperabilitätsspezifikation SigI, Abschnitt A2 Signatur*. http://www.bsi.de/esig/basics/techbas/interop/bsi/sigi-a2.pdf. Version 6.1, Stand 30. Juni 1999.

[15] Bundesamt für Sicherheit in der Informationstechnik: *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV, Signatur-Interoperabilitätsspezifikation SigI, Abschnitt A4 Zeitstempel*. http://www.bsi.de/esig/basics/techbas/interop/bsi/sigi-a4.pdf. Version 3.0, Stand 31. März 1999.

[16] Bundesamt für Sicherheit in der Informationstechnik: *Grundlagen der elektronischen Signatur*. SecuMedia, 2006.

[17] Bundesnetzagentur: *FAQ*. http://www.bundesnetzagentur.de/enid/Elektronische_Signatur/FAQ_pm.html. Stand 16. 01. 2007.

[18] Bundesnetzagentur: *Produkte für qualifizierte elektronische Signaturen*. http://www.bundesnetzagentur.de/enid/Elektronische_Signatur/Produkte_pi.html. Stand 16. 01. 2007.

[19] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen) vom 2. Januar 2006*. Bundesanzeiger Nr. 58, S. 1913-1915, März 2006. http://www.bundesnetzagentur.de/media/archive/5951.pdf.

[20] Comment Ça Marche: *Introduction à J2EE*. http://www.commentcamarche.net/j2ee/j2ee-intro.php3. Stand 15. 02. 2007.

[21] Cooper, J. W.: *The Design Patterns Java Companion*. http://www.patterndepot.com/put/8/JavaPatterns.htm. Stand 20. 02. 2007.

[22] Coté: *JAAS in Action by Coté*. http://www.jaasbook.com. Stand 13. 02. 2007.

[23] Dierks, T. and C. Allen: *The TLS Protocol*. RFC 2246, Internet Engineering Task Force, January 1999.

[24] European Telecommunications Standards Institute (ETSI): *Electronic Signatures and Infrastructures (ESI); Policy requirements for time-stamping authorities*. 102 023 V1.2.1, January 2001.

[25] European Telecommunications Standards Institute (ETSI): *Electronic Signatures and Infrastructures (ESI), CMS Advanced Electronic Signatures (CAdES)*. TS 101 733 V1.6.3, September 2005.

[26] European Telecommunications Standards Institute (ETSI): *Time stamping profile*. TS 101 861 V1.3.1, January 2006.

[27] Fielding, R. *et al.*: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Internet Engineering Task Force, June 1999.

[28] FlexSecure: *FlexiTrust Trustcenter Software Version 3.0 OCSP-Responder*, 2004.

[29] Fowler, M.: *Inversion of Control Containers and the Dependency Injection pattern*. `http://www.martinfowler.com/articles/injection.html`. Stand 20. 01. 2007.

[30] Franks, J. *et al.*: *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617, Internet Engineering Task Force, June 1999.

[31] Giessmann, E. G.: *Elektronische Signaturen Technik der Zukunft*. Signatur-Workshop der Regulierungsbehörde für Telekommunikation und Post, 16.- 17. September 2003. `http://www.bundesnetzagentur.de/media/archive/2409.pdf`.

[32] Gondrom, T., R. Brandner, and U. Pordesch: *Evidence Record Syntax (ERS)*. IETF Internet draft draft-ietf-ltans-ers-09, Internet Engineering Task Force, January 2007.

[33] Hühnlein, D.: *Intervall-qualifizierte Zeitstempel*. In: Horster, P. (Herausgeber): *Tagungsband „Elektronische Geschäftsprozesse“*, Seiten 431–445. IT-Verlag, 2004. `http://213.68.205.170/download/fachartikel/iq-zeitstempel.pdf`.

[34] Hühnlein, D. und Y. Knosowski: *Aspekte der „Massensignatur“*. In: Horster, P. (Herausgeber): *Tagungsband „D-A-CH Security“*, Seiten 293–307. IT-Verlag, 2003. `http://inetorg.alphasystems.com/papers/Aspekte_Massensignatur_Nachdruck.pdf`.

[35] Hoffman, P.: *Enhanced Security Services for S/MIME*. RFC 2634, Internet Engineering Task Force, June 1999.

[36] Housley, R.: *Cryptographic Message Syntax*. RFC 2630, Internet Engineering Task Force, June 1999.

[37] Housley, R.: *Cryptographic Message Syntax (CMS)*. RFC 3852, Internet Engineering Task Force, July 2004.

[38] Housley, R., W. Polk, W. Ford, and D. Solo: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3280, Internet Engineering Task Force, April 2002.

[39] IPA/JNSA Challenge PKI Test Suite: *Challenge PKI Test Suite 2.0 Document*. `http://www.jnsa.org/mpki/ts`, 2004.

[40] Japan Network Security Association (JNSA): *Challenge PKI 2003 - The Timestamp-protocol Interoperability Test Suite*. `http://www.jnsa.org/mpki/2003/index.html`, July 2004.

[41] JBoss: *JBoss Application Server Documentation Library*. `http://labs.jboss.com/portal/jbossas/docs`. Stand 20. 01. 2007.

[42] Johnson, R. *et al.*: *Spring, Java/J2EE Application Framework, 2.0.2*. `http://static.springframework.org/spring/docs/2.0.x/spring-reference.pdf`, 2007.

[43] Justin, A.: *Spring and EJB 3.0 in Harmony*. http://java.sys-con.com/read/180386_1.htm, February 2006.

[44] Kanthak, S. and J. Schluchter: *Design and Implementation of a Secure Authorization Concept*. Stand 01. 10. 2006.

[45] Kunstein, F.: *Die elektronische Signatur als Baustein der elektronischen Verwaltung*. Juristische Reihe TENEA Bd. 88, 2005. http://download.jurawelt.com/download/dissertationen/tenea_juraweltbd88_kunstein.pdf.

[46] Maier, M.: *Das Spring Framework als Teil eines Paradigmenwechsels? Vergleich der leichtgewichtigen Alternative zur traditionellen J2EE Entwicklung*. Diplomarbeit, FH JOANNEUM Gesellschaft mbH, Juli 2005. http://www.martinmaier.name/wp-content/Diplomarbeit_Martin_Maier.pdf.

[47] Middendorf, S., R. Singer und J. Heid: *Programmierhandbuch und Referenz für die Java^{TM}-2-Plattform, Standard Edition*. dpunkt.Verlag, 3. Auflage, 2002. http://www.dpunkt.de/java/index.html.

[48] Myers, M. *et al.*: *Online Certificate Status Protocol - OCSP*. RFC 2560, Internet Engineering Task Force, June 1999.

[49] Ohst, D.: *Einsatz elektronischer Signaturen und Zeitstempel für die Sicherung digitaler Dokumente*. Diplomarbeit, Humboldt-Universität zu Berlin, 2004. http://edoc.hu-berlin.de/docviews/abstract.php?lang=ger&id=25521.

[50] Regulierungsbehörde für Telekommunikation und Post: *Maßnahmenkatalog für digitale Signaturen - auf Grundlage von SigG und SigV von 1997*. Entwurf. Version 1.0, Stand 18. 11. 1997.

[51] Roßnagel und Schmücker (Herausgeber): *Beweiskräftige elektronische Archivierung*. Economica, 2005.

[52] RSA Laboratories: *PKCS #7: Cryptographic Message Syntax Standard*, November 1993. http://www.rsasecurity.com.

[53] Rütten, C.: *Hash-Funktion SHA-1 in Bedrängnis*. http://www.heise.de/newsticker/meldung/77235, August 2006.

[54] Sriganesh, R. P., G. Brose, and M. Silverman: *Mastering Enterprise JavaBeans 3.0*. Wiley Publishing, July 2006.

[55] Sun Microsystems: *Java Management Extensions (JMX)*. http://java.sun.com/j2se/1.5.0/docs/guide/jmx. Stand 20. 01. 2007.

[56] Sun Microsystems: *Annotations*. http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html. Stand 14. 02. 2007.

[57] Sun Microsystems: *EJB Restrictions*. http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html. Stand 20. 01. 2007.

[58] Sun Microsystems: *Java TM 2 Platform Standard Edition 5.0 API Specification*. http://java.sun.com/j2se/1.5.0/docs/api/index.html. Stand 13. 02. 2007.

[59] Sun Microsystems: *The Java EE 5 Tutorial*. `http://java.sun.com/javaee/5/docs/tutorial/doc`, September 2006.

[60] T7 e. V. und TeleTrusT e. V.: *ISIS-MTT-Specification*. Version 1.1, March 2004. `http://www.isis-mtt.t7-isis.org/uploads/media/ISIS-MTT_Core_Specification_v1.1_03.pdf`.

[61] Westphal, F.: *Unit Tests mit JUnit*. `http://www.frankwestphal.de/UnitTestingmitJUnit.html`, Juni 2001.

[62] Zörkler, J.: *Einführung einer Public-Key-Infrastruktur in einer Universität*. Diplomarbeit, Technische Universität Darmstadt, Juni 2003. `http://www.cdc.informatik.tu-darmstadt.de/reports/reports/Zoerkler.diplom.pdf`.