Diplomarbeit

# Sicherheitsmechanismen für ein elektronisches Archiv digital signierter Dokumente

zur Erlangung des akademischen Grades
**Diplom-Informatiker (FH)**

vorgelegt dem
Fachbereich Mathematik, Naturwissenschaften und Informatik der
Fachhochschule Gießen-Friedberg

eingereicht von

Ingo Graser

im August 2004

Referent: Dr. phil. nat., Dipl.-Math. Burkhardt Renz

Koreferent: Dr. rer. nat., Dipl.-Math. Wolfgang Henrich

## Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

_____

Ort, Datum, Unterschrift

# Inhaltsverzeichnis

# Teil I

# Einleitung

# 1. Das elektronische Personenstandsbuch

## 1.1 Über das Projekt

Die gesamte öffentliche Verwaltung befindet sich zurzeit im Umbruch. Grund ist die Umstellung auf elektronische Verfahren. Erstes Ziel soll dabei die Vereinfachung der Verwaltungsarbeit sein. Gleichzeitig soll durch die Einführung einer Online-Verwaltung der Bürger und die Wirtschaft besser eingebunden werden.

Auch die Einführung von elektronischen Standesbüchern in den Standesämtern wurde seit einigen Jahren angedacht. Konkrete Ergebnisse gab es jedoch bislang keine. Das Projekt „elektronisches Personenstandsbuch“ (e.P.b.) entstand in Zusammenarbeit der Fachhochschule Gießen-Friedberg mit dem Verlag für Standesamtswesen in Frankfurt am Main.

Prof. Dr. Burkhardt Renz (FH Gießen-Friedberg) beschäftigte sich schon seit einigen Jahren mit der Thematik. Vor seiner Professur war er an der Entwicklung von Software für die Erstellung von Personenstandseinträgen beteiligt. Damit war er sehr gut mit der komplexen Standesamtsmaterie vertraut.

Der Verlag für Standesamtswesen (VfSt) startete 2004 mit der Fachhochschule Gießen-Friedberg, unter Leitung von Prof. Dr. Renz, das Projekt „elektronisches Personenstandsbuch“. Ziel ist die Konzeptionierung und der Entwurf eines solchen Systems. Im Sommersemester 2004 wurde dann ein wissenschaftlicher Mitarbeiter engagiert und diese Diplomarbeit vergeben.

Bei einem einführenden Treffen im Februar 2004 mit den Vertretern des VfSt Jörg Thiemer und Olaf Horn wurden die grundlegenden Ideen ausgetauscht und untereinander abgestimmt. Außerdem wurde ein Zeitplan für das Projekt aufgestellt.

Ein erstes Konzept für die Sicherheits- und Signaturlösungen wurde dann Ende März bei einem Treffen im VfSt mit Vertretern des hessischen Innenministeriums und der Entwicklungsabteilung des VfSt vorgestellt. Dabei konnte man sich auf die grundlegenden Konzepte eines e.P.b. verständigen.

Ausgehend von diesem Treffen konnte die Projektgruppe das Konzept weiter detaillieren. Im Mai wurde dann begonnen, eine softwaretechnische Lösung zur Umsetzung des Konzeptes zu entwerfen.

## 1.2 Zielsetzung für das Projekt

Das Ziel ist die Erarbeitung eines Konzeptes für die elektronische Archivierung von Personenstandseinträgen. Dieses elektronische Archiv soll die Einlagerung von Papierdokumenten zunächst ergänzen und auf lange Sicht ablösen. Die Hauptforderung an die elektronischen Einträge ist, dass sie auch nach Jahrzehnten noch genauso rechtskräftig wie ihre Papiervorgänger sind.

Damit das System nicht selbst durch die komplexen Anforderungen an Personenstandseinträge zu aufwendig wird, ist vorgesehen, ein reines Archivierungssystem zu entwerfen, welches lediglich die Endform der Einträge abspeichert und keine Informationen über deren Inhalt und Aufbau haben muss.

Neben der Einsparung von Papier und der bequemen Suche über die Einträge soll die Vernetzung der Standesämter weitere Vorteile bringen. Die Vision ist, dass der Bürger auf ein beliebiges Standesamt gehen kann. Angeforderte Einträge werden dort elektronisch bei den zuständigen Standesämtern angefordert. Die erforderlichen Urkunden oder beglaubigten Abschriften können dem Bürger sofort elektronisch oder als Ausdruck ausgehändigt werden. Dies ist zwar noch nicht vereinbar mit den bisherigen rechtlichen und organisatorischen Rahmenbedingungen in den Standesämtern, trotzdem soll ein Konzept vorgeschlagen werden, was eine solche Nutzung des Systems ermöglicht.

Zur Arbeitserleichterung für die Standesbeamten ist zudem vorgesehen, neben den signierten Abbildern der Einträge, die Eintragsdaten auch strukturiert zu speichern. Dadurch können diese Daten später ohne erneutes Eintippen weiterverwendet werden.

Die Architektur eines e.P.b. muss dabei so aufgebaut sein, dass die Einführung eines e.P.b. in einem Standesamt stufenweise erfolgen kann. Ein teilnehmendes Standesamt muss sich nicht automatisch an das e.P.b.-Netz ankoppeln. Diese Funktionen werden optional angeboten. Es wird an der jeweiligen Standesamtsleitung liegen, zu entscheiden, welche Funktionen des e.P.b. eingesetzt werden.

Alle nötigen Protokolle, Datenstrukturen und Architekturen, die dazu nötig sind, sollen entworfen werden. Am Ende des Projektes soll dann der Aufbau einer Infrastruktur stehen, an der die Entwürfe getestet und demonstriert werden.

## 1.3 Das Ziel dieser Arbeit (Exposé)

Der Inhalt dieser Arbeit ist zunächst die Erstellung eines Konzepts für die Signatur von Einträgen und für den Erhalt der Beweisfähigkeit dieser digitalen Unterschriften über Jahrzehnte.

Daneben soll auch ein Sicherheitskonzept ausgearbeitet werden, welches den sicheren Betrieb einer solchen Infrastruktur gewährleistet. Dieses Sicherheitskonzept geht dabei ausschließlich auf die Sicherheitsmerkmale ein, die durch Maßnahmen in der Software erreicht werden können. Die Einrichtung eines sicheren Netzwerkes in den Standesämtern ist nicht Teil dieser Arbeit.

Anschließend ist die Sicherheitskomponente zu entwerfen, die in der e.P.b.-Architektur alle Aufgaben zu Signatur und Signaturerhaltung übernehmen soll. Dazu ist der Entwurf aller benötigen Datenstrukturen, Programmstrukturen und APIs notwendig.

Abschließend ist experimentell zu erforschen, wie die geplante Funktionalität in Java umgesetzt werden kann. Insbesondere die verfügbaren Bibliotheken und APIs sind auf ihre Eignung für das e.P.b. zu testen.

## 1.4 Ergebnis

In dieser Diplomarbeit wurde zunächst ein Konzept zur Langzeiterhaltung der digitalen Signaturen, angelehnt an die Empfehlungen des Bundesamtes für Sicherheit in der Informationstechnik, entwickelt. Dabei versehen Zeitstempel die Signaturen mit einem verlässlichen Datum. So kann bewiesen werden, dass eine Signatur zu einer Zeit entstanden ist, zu der die Rahmenbedingungen (Zertifikate und Algorithmen) eine rechtsgültige Signatur ermöglichten.

Das erarbeitete Konzept sieht außerdem vor, die Kommunikation mittels dem SSL-Protokoll zu verschlüsseln. In Kombination mit dem zusätzlich entwickelten Login-Mechanismus, der als Challenge-Response-Protokoll auf Basis qualifizierter Zertifikate ausgelegt ist, soll damit von Seite des e.P.b.-Systems eine möglichst hohe Sicherheit vor Angriffen auf das System erreicht werden

Neben der Beschreibung dieses Konzeptes enthält die Diplomarbeit auch die erforderlichen Programm- und Datenstrukturen, die zu einer Implementierung des Konzeptes herangezogen werden können. Diese entsprechen dem aktuellen Stand des Projektes bei Abgabe der Diplomarbeit und können bei zukünftigen Weiterentwicklungen als Richtschnur dienen.

Der letzte Teil befasst sich mit der Implementierung der geplanten Funktionalität in Java. Dabei werden jeweils Teilaspekte herausgegriffen und ihre Unterstützung seitens Java mit Codebeispielen erläutert.

## 1.5 Danksagung

An dieser Stelle möchte ich mich bei Dr. Burkhardt Renz und Diplom-Informatiker Sebastian Süß für die produktive Zusammenarbeit bedanken. An diesem Projekt mitzuwirken hat mir persönlich viel Spaß gemacht. Über eine weitere Zusammenarbeit in der Zukunft würde ich mich freuen.

Mein Dank geht auch an Jörg Thiemer und Olaf Horn vom Verlag für Standesamtswesen, die meine Fragen stets schnell und kompetent beantworteten.

Abschließend ein großes Dankeschön an die folgenden Personen, die bewusst oder unbewusst, ihren Beitrag zu dieser Arbeit geleistet haben: Katrin, Carsten, Karlheinz, Angelika und Bastian.

*Gießen, den 13.08.2004*

# 2. Das Grundkonzept des e.P.b.

Bevor in dieser Arbeit auf die Entwicklung eines detaillierten Konzeptes eingegangen wird, folgt zunächst ein Überblick über die Hintergründe im Personenstandswesen. Danach wird das Grundkonzept, welches seit dem Start des Projektes verfolgt wird, präsentiert.

Das Grundkonzept lag schon vor Beginn dieser Arbeit fest. Das in dieser Arbeit entwickelte Signatur- und Sicherheitskonzept passt sich nahtlos in dieses Konzept ein.

## 2.1 Hintergründe des Personenstandswesens

Wenn in dieser Arbeit von Personenstandseinträgen (oder einfach: Einträgen) die Rede ist, so handelt es sich dabei um die einzelnen Urkunden, die in den Personenstandsbüchern archiviert werden. Die Standesämter führen folgende Personenstandsbücher: Das *Heiratsbuch* enthält die Einträge über Eheschließungen. Das *Geburtenbuch* enthält Einträge zur Beurkundung von Geburten und das *Sterbebuch* die Einträge zur Beurkundung der Sterbefälle. Die Einträge im *Lebenspartnerschaftsbuch* enthalten die Daten der gegründeten Lebenspartnerschaften. Die Form der einzelnen Einträge ist abhängig von dem Personenstandsbuch und gelegentlichen Anpassungen der Regularien unterworfen.

Mit seiner Unterschrift unter einem Eintrag beglaubigt der Standesbeamte die ausgestellte Urkunde. Jedoch muss dieser Eintrag nach der Unterschrift nicht zwingend unverändert bleiben. Durch Fortführungen können Tippfehler, welche den Inhalt in seinem Sinn verändern, korrigiert werden und Ergänzungen angebracht werden.

Diese Fortführungen werden am Rand des Eintrags angefügt und von dem fortführenden Standesbeamten unterzeichnet. Sie sind also eindeutig als solche Hinzufügungen zu erkennen.

Ergänzend zu den Büchern existiert in den Standesämtern noch eine Kartei (Namenverzeichnis), welche Kataloginformationen zu allen Einträgen enthält. Damit wird es erleichtert, einen alten Eintrag wieder aufzufinden.

Die Standesämter setzen bei der Erstellung der Personenstandseinträge schon jetzt auf die elektronische Datenverarbeitung. Über Eingabemasken werden die erforderlichen Daten eingegeben. Die Software erstellt dann den Eintrag nach den aktuellen Formvorschriften. Bisher wird dieser Eintrag dann ausgedruckt und vom Standesbeamten per Hand unterschrieben.

Die gesammelten Einträge eines Jahres werden gebunden und als Personenstandsbuch eingelagert. Dabei werden große Lagerräume benötigt, die die enormen Papiermengen, die sich über die Jahre anhäufen, aufnehmen können.

## 2.2 Vorteile und Möglichkeiten eines e.P.b.

Das Projekt „elektronisches Personenstandsbuch“ (e.P.b.) erarbeitet ein Konzept für die elektronische Archivierung dieser Personenstandseinträge. Statt Unmengen von Papierdokumenten einzulagern, soll ein elektronisches Archiv die Einträge aufnehmen. Die digitale Signatur ersetzt auf diesen Einträgen die Unterschrift per Hand. Die Herausforderung ist dabei, dass die elektronischen Dokumente auch nach Jahrzehnten noch genauso rechtskräftig wie ihre Papiervorgänger sein müssen.

Für das System ist eine lokale, dezentrale Lösung vorgesehen. In jedem teilnehmenden Standesamt wird ein e.P.b.-Archivserver aufgestellt. Dieser lokale Server beinhaltet alle Personenstandseinträge dieses Standesamtes[1].

Die e.P.b.-Archivserver mehrerer Standesämter können vernetzt werden. Damit ist es möglich, eine entfernte Suche nach Einträgen eines anderen Standesamtes durchzuführen. In Zukunft soll es damit für den Bürger möglich werden, auf ein beliebiges Standesamt zu gehen und dort alle gewünschten Urkunden aus dem gesamten Bundesgebiet zu erhalten.

Zurzeit ist eine solche Funktionalität noch nicht durchsetzbar, jedoch wird e.P.b. mehrere Möglichkeiten dazu anbieten, um für die Zukunft gerüstet zu sein. Die Aufsicht darüber, ob und welche dieser Möglichkeiten im jeweiligen Standesamt zum Einsatz kommt, obliegt dabei dem einzelnen Standesamt.

Die erste Möglichkeit sieht vor, bei der Suche nach einem fernen Eintrag lediglich den eindeutigen Schlüssel dieses Eintrags auszuliefern. Mit diesem kann der Eintrag beim zuständigen Standesamt über eine externe sichere Kommunikationsplattform, die e.P.b. nicht zur Verfügung stellt, angefragt werden.

Als Alternative dazu bietet e.P.b. die Möglichkeit, gezielt einzelnen Standesämtern oder Standesbeamten den Fernabruf von Einträgen, bei der jeweiligen Angabe eines Grundes für den Abruf, zu erlauben. So ist der Personenkreis, welcher Zugriff hat, bekannt. Außerdem lässt sich Missbrauch leicht aufdecken.

Einträge von Jahrgängen, bei denen die Einträge selbst noch nicht elektronisch abgespeichert wurden, können durch das Namensverzeichnis für die Suche erschlossen werden. Dabei werden lediglich die Suchdaten im e.P.b.-Archivserver abgelegt. Bei einer Suche kann damit das Standesamt ermittelt werden, welches den gewünschten Eintrag besitzt.

Die Architektur von e.P.b. wird dabei so aufgebaut sein, dass die Einführung von e.P.b. in einem Standesamt stufenweise erfolgen kann. Ein teilnehmendes Standesamt muss sich nicht automatisch an das e.P.b.-Netz ankoppeln. Diese Funktionen werden optional angeboten. Es liegt an der jeweiligen Standesamtsleitung zu entscheiden, welche Funktionen von e.P.b. eingesetzt werden.

## 2.3 Betriebsszenarien

Um einen Überblick über die späteren Möglichkeiten eines e.P.b. zu geben, hier eine Aufstellung der erwarteten Funktionalität und der Betriebsabläufe aus Sicht der Endanwender:

[1] Ein aktuelles Standesamt kann aus einem Zusammenschluss mehrerer früherer Standesämter hervorgegangen sein. Somit können auch die Einträge der früheren Standesämter auf diesem lokalen Server abgelegt sein.

**Erstmalige Erfassung eines Eintrags** Der Eintrag wird mithilfe der Software, die für die Bearbeitung von Personenstandsfällen im Standesamt verwendet wird, erstellt. Unterstützt diese Software das e.P.b., dann fordert sie bei der Ablage in das Archiv zunächst den Standesbeamten zur Signatur mit seiner persönlichen Chipkarte und PIN auf. Dann übergibt sie das Dokument an den e.P.b.-Archivserver, der alle weiteren Schritte selbst vornimmt.

**Scan eines vorhandenen Eintrags** Der Aufwand die bisherigen, schriftlichen Einträge automatisiert für das e.P.b. in strukturierter Form zu erfassen, wäre unverhältnismäßig groß. Deshalb wird vorgesehen die alten Einträge als Bild zu erfassen und in dieser Form, unstrukturiert, in das e.P.b. aufzunehmen.

Hier verläuft die Aufnahme genau so wie bei einem neuen Eintrag, mit dem Unterschied, dass der Bearbeiter mit seiner Signatur lediglich die korrekte Aufnahme belegt. Eine digitale Signatur des damals verantwortlichen Standesbeamten wird nicht benötigt.

**Nachbeurkundung** Das e.P.b. ermöglicht auch einen alten Eintrag in einem neuen Formular aufzunehmen. Dabei werden alle bisherigen Fortführungen mit eingearbeitet. Auf diese Weise können auch Eintragsdaten erfasst werden, die eine spätere Weiterverwendung erleichtern.

**Suche und Einsichtnahme** Eine Standesamtssoftware, die das e.P.b. unterstützt, kann nach Eingabe von Suchbegriffen eine Liste der gefundenen Einträge anfordern. Anhand eines eindeutigen Schlüssels kann der Standesbeamte dann aus dem Archiv einen früheren Eintrag laden.

Dieser wird in der gewohnten Oberfläche präsentiert. Der Archivserver liefert den Eintrag und garantiert dessen Integrität. Anhand der mitgelieferten Signaturen, Zertifikate und Zeitstempel kann der Abnehmer seinerseits die Unterschrift prüfen.

**Fortführen eines Eintrags** Bei einer Fortführung fordert die Standesamtssoftware, wie bei der Einsichtnahme, den Eintrag beim e.P.b. an. Der Unterschied hierbei ist, dass nun im Archiv vermerkt wird, dass ein Eintrag gerade in Bearbeitung ist. Dies sichert ab, dass nur ein Standesbeamter zur gleichen Zeit einen Eintrag fortführt.

Wird der Eintrag zurückgespeichert, bestätigt der Bearbeiter mit seiner Signatur die Gültigkeit seiner Fortführung und die Zugehörigkeit zu dem bestehenden Eintrag. Hierbei wird der alte Eintrag lediglich als veraltet gekennzeichnet (und nicht gelöscht). Schließlich wird der neue Eintrag angelegt, welcher einen Verweis auf den alten enthält[2]. Die Signatur des Bearbeiters der Fortführung bezieht sich auf das gesamte Paket.

[2] Dies ist eine notwendige Konsequenz des eingesetzten Signaturkonzeptes. Mehr dazu später in der genauen Beschreibung des Konzeptes zur Langzeiterhaltung der Signaturen.

# Teil II

# Der Weg zu einem Signatur- und Sicherheitskonzept

# 3. Technische Hintergründe

## 3.1 Digitale Signatur

Obwohl es die digitale Signatur schon seit einigen Jahren gibt und das deutsche Signaturgesetz deren Einsatz im elektronischen Geschäftsverkehr fördert, ist sie in der Regel noch nicht im Alltag angekommen. Auch die Standesbeamten besitzen noch keine Schlüsselpaare und Zertifikate. Der erste Schritt zu digital signierten Personenstandseinträgen muss also die Ausstattung der Beamten mit Zertifikaten und entsprechendem Equipment sein.

Um Vertrauen in die Sicherheit der digitalen Signatur zu bekommen, ist ein grundlegendes Verständnis dieser Technologie und der dahinter stehenden Infrastruktur von Nöten. Die folgenden Abschnitte geben eine Einführung in die Funktionsweise der digitalen Signatur und zeigen, welche Einrichtungen nötig sind, die Standesämter auf die digitale Signatur vorzubereiten.

### 3.1.1 Einleitung

Je mehr elektronische Kommunikation den bisherigen Briefverkehr ablöst, desto lauter werden die Rufe nach Möglichkeiten für rechtsverbindliche elektronische Transaktionen. Die bisherige Unterschrift per Hand ist ungeeignet für den elektronischen Geschäftsverkehr. Allzu leicht könnten eingescannte Unterschriften kopiert und missbräuchlich verwendet werden. Deshalb ist eine solche Unterschrift vor Gericht nicht beweisfähig.

Abhilfe schafft die digitale Signatur. Diese entsteht dadurch, dass dem Text eine verschlüsselte, redundante Information beigefügt wird. Diese weist Eigenschaften auf, die einer Unterschrift per Hand ähneln. Außerdem bietet sie weitere Eigenschaften, die die Sicherheit der elektronischen Kommunikation erhöhen:

- Man kann sie einem eindeutigen Urheber zuordnen,
- nur dieser kann sie erstellen,
- sie ist dann nur für dieses eine Dokument gültig und
- der unterschriebene Text kann später nicht mehr unbemerkt verändert werden.

Eine qualifizierte Signatur ist essenziell wichtig für Geschäfte über elektronische Plattformen. Der Absender einer signierten Nachricht kann später nicht mehr bestreiten, der Urheber gewesen zu sein. Wie diese Eigenschaften erreicht werden, ist in den folgenden Abschnitten beschrieben.

Diese Merkmale verfallen jedoch mit der Zeit. Im Gegensatz zu einer Unterschrift per Hand ist eine digitale Signatur nicht unendlich lange gültig. Dies liegt zum einen an organisatorischen Gegebenheiten des Signatursystems (siehe Abschnitt *„Die Public-Key-Infrastruktur“*). Zum anderen liegt es aber auch daran, dass neue Computer immer leistungsfähiger werden und sich somit die dem Signatursystem zugrunde liegenden Kryptoalgorithmen irgendwann „aushebeln“ lassen könnten.

Aus diesem Grund wird die aktuelle Entwicklung ständig überwacht. Es wird festgelegt, welche Kryptoalgorithmen noch als sicher anzusehen sind, und welche nicht mehr eingesetzt werden dürfen. Die Regulierungsbehörde für Telekommunikation und Post (RegTP) gibt Empfehlungen für Schlüssellängen und Algorithmen heraus. Im Bundesanzeiger werden dann die verbindlichen Schlüssellängen und zugelassenen Algorithmen veröffentlicht.

Ist nun eine vorliegende Signatur auf Basis eines Kryptoalgorithmus entstanden, der als nicht mehr sicher eingestuft wird, ist diese Signatur alleine automatisch nicht mehr beweisfähig.

Vor allem bei der Einlagerung von rechtsverbindlichen Urkunden oder Akten, die über einen längeren Zeitraum aufbewahrt werden, besteht diese Problematik. Dieser Umstand macht es nötig, geeignete Konzepte einzuführen, die die langfristige Gültigkeit von Signaturen bewahren. Im Laufe dieser Arbeit wird ein Konzept zur langfristigen Sicherung der Beweisfähigkeit digitaler Signaturen vorgestellt. Mehr dazu im Kapitel *Das Konzept für ein e.P.b.*.

Dafür, dass die digitale Signatur vor Gericht einer handschriftlichen Signatur ebenbürtig ist, sorgt das deutsche Signaturgesetz. Der ersten Fassung von 1997 folgte 2001 eine Überarbeitung, um den Gesetzestext an die europäische Signaturrichtline anzupassen. Das Signaturgesetz (*SigG*) gibt die Rahmenbedingungen vor, die ein digitales Signaturverfahren erfüllen muss, um rechtskräftige Signaturen zu erzeugen. Die Signaturverordnung (*SigV*) setzt die Vorgaben des SigG um und zeigt damit den Weg zu beweiskräftigen Signaturen. Die Auswirkungen des Gesetzes auf ein Signatursystem wird im Kapitel *„Die aktuelle Rechtslage für Signaturen“* beleuchtet.

Der Zusammenfassung der Verfahren in den folgenden Abschnitten wurde zu großen Teilen [22] zugrunde gelegt.

### 3.1.2 Signatur mittels Hashfunktion und RSA

Nach einer allgemeinen Beschreibung digitaler Signaturen, nun zu der technischen Seite. Das hier vorgestellte Verfahren mittels kryptografischer Hashfunktion und RSA-Verschlüsselung (Erfinder: **R**ivest, **S**hamir und **A**dleman) ist das verbreitetste. Andere Verfahren spielen heutzutage praktisch keine Rolle mehr.

Eine kryptografische Hashfunktion unterscheidet sich dabei nicht von einer allgemeinen Hashfunktion. Es wird bei diesem Algorithmus lediglich großer Wert darauf gelegt, wenige Kollisionen zu erzeugen. Das bedeutet, dass sich die berechneten Werte möglichst gleichmäßig über den gesamten Wertebereich verteilen sollen. Damit soll es schwerer werden zu einem gegebenen Text einen zweiten zu finden, der denselben Hashwert erzeugt.

Für heute gängige und zugelassene (siehe weiter unten) kryptografische Hashfunktionen beträgt die Hashwertlänge 160 Bit (RipeMD-160, SHA-1). Es gibt also etwa 1.46 x $10^{48}$ verschiedene Hashwerte, die einem Dokument zugeordnet werden können. Somit ist es hinreichend aufwendig, ein weiteres Dokument mit demselben Hashwert zu finden.

Das eigentliche Signaturverfahren ist aber die RSA-Verschlüsselung[9]. Dabei handelt es sich um ein asymmetrisches Verschlüsselungsverfahren. Das bedeutet, es werden zwei verschiedene Schlüssel benötigt, um ein Dokument zu verschlüsseln und später wieder zu entschlüsseln. Diese zwei Schlüssel werden genau entgegengesetzt verwendet. Wird ein Dokument mit dem ersten Schlüssel verschlüsselt, so lässt es sich lediglich mit dem zweiten Schlüssel dieses Paares wieder entschlüsseln. Dies funktioniert in beide Richtungen.

Man spricht hierbei auch von Public-Key-Verfahren, da einer der Schlüssel öffentlich bekannt gemacht (public key) und der andere vom Inhaber geheim gehalten (private key) wird. Dabei ist der Schlüsselinhaber selbst dafür verantwortlich, dass sein geheimer Schlüssel nicht öffentlich bekannt wird.

Auf diese Weise wird es möglich, einen Text mit dem geheimen (privaten) Schlüssel zu verschlüsseln, der dann von jedem anderen, dem der zugehörige öffentliche Schlüssel bekannt ist, entschlüsselt werden kann. Damit ist der Text signiert, denn er kann *nur* mit dem privaten Schlüssel verschlüsselt worden sein. Lediglich mit dem privaten Schlüssel verschlüsselte Nachrichten lassen sich später mit dem öffentlichen Schlüssel entschlüsseln. Da dieser geheime Schlüssel ausschließlich einer Person bekannt ist, kann die Urheberschaft bewiesen werden.

Schlüsselerzeugung beim RSA-Verfahren:

1. Wählen zweier möglichst großer Primzahlen *p* und *q*.
2. Berechnen von $n = p \cdot q$.
3. Beliebige Wahl einer dritten Zahl *e*, die teilerfremd zu $(p-1) \cdot (q-1)$ ist.
   (*e* und *n* sind nun gemeinsam der öffentliche Schlüssel)
4. $d = e^{-1}\ mod\ ((p-1) \cdot (q-1))$ (*d* ist der geheime Schlüssel)

Verwendung der Schlüssel:

1. Verschlüsseln der Nachricht *m* mit $c = m^e\ mod\ n$.
2. Entschlüsseln von *c* mit $m = c^d\ mod\ n$

Als Schlüssellänge (Größe von *n*) sind derzeit 1024 Bit als Minimalwert und 2048 Bit als empfohlener Wert von der Regulierungsbehörde für Telekommunikation und Post bis 2007 festgelegt worden[1]. Diese Empfehlung wurde aufgegriffen und im Bundesanzeiger veröffentlicht[2].

Bei dem hier vorgestellten Signaturverfahren wird nicht die gesamte Nachricht signiert, sondern zunächst der Nachrichtentext mittels einer kryptografischen Hashfunktion komprimiert und dann lediglich dieser Hashwert signiert. Das liegt daran, dass das RSA-Verfahren sehr rechenaufwendig ist und je nach Nachrichtenlänge zu unzumutbaren Wartezeiten bei Sender und Empfänger führen könnte.

[1] `http://www.regtp.de/imperia/md/content/tech_reg_t/digisign/163.pdf` [Stand vom 17.05.2004]

[2] Bundesanzeiger (Nr. 30, S. 2537-2538 vom 13.02.2004)

Der Nachteil dieser Vorgehensweise ist jedoch auf der anderen Seite, dass man sich nun auf die Sicherheit von zwei Algorithmen verlassen muss. Wenn ein Angreifer zu einer signierten Nachricht einen anderen Text finden kann, der denselben Hashwert besitzt, dann könnte er den Ursprungstext gegen den Neuen austauschen, ohne dass dies später erkennbar wäre. Im schlimmsten Fall wird dann der Urheber der ursprünglichen Signatur für den untergeschobenen Inhalt zur Verantwortung gezogen. Aus diesem Grund gibt die RegTP zeitlich befristete Empfehlungen für Verfahren und Schlüssellängen.

Abbildung 3.1 zeigt den Ablauf des kompletten Verfahrens:

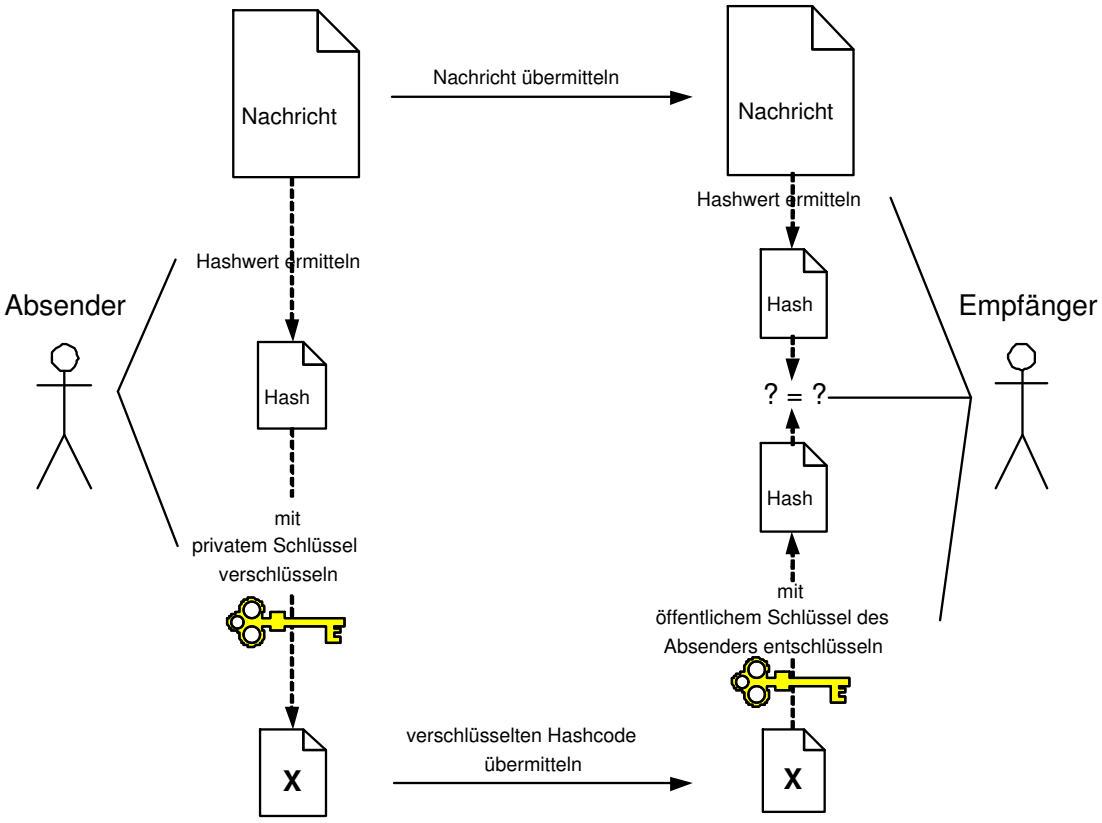


Abbildung 3.1: Das Verfahren für digitale Signaturen per Hashfunktion und RSA-Verschlüsselung.

1. Der Absender ermittelt zu seiner Nachricht den Hashwert.
2. Dieser wird mit dem privaten Schlüssel des Absenders verschlüsselt und
3. die Nachricht plus verschlüsselter Hashwert werden zum Empfänger gesendet.
4. Der Empfänger entschlüsselt den Hashwert mit dem öffentlichen Schlüssel des Absenders und
5. ermittelt aus dem empfangenen Dokument ebenfalls den Hashwert.
6. Stimmen selbst erzeugter Hashwert und verschlüsselt empfangener Hashwert überein, ist

- die Nachricht unverändert beim Empfänger angekommen und
- der Absender bestätigt mit seiner Signatur den Inhalt.

Die Signatur konnte also nur mit dem geheimen Schlüssel des Absenders verschlüsselt worden sein. Aber wer sagt, dass der verwendete öffentliche Schlüssel des Absenders auch der des *richtigen* Absenders ist? Dies kann nur durch eine vertrauenswürdige Infrastruktur garantiert werden.

### 3.1.3 Die Public-Key-Infrastruktur (PKI)

Damit Public-Key-Verschlüsselungsverfahren sinnvoll einsetzbar sind, wird eine zuverlässige Infrastruktur benötigt. Dabei muss die Identifikation von Schlüsselinhabern, die Geheimhaltung der privaten Schlüssel und die Verbreitung der öffentlichen Schlüssel organisiert werden.

Heutzutage finden verschiedene Arten von PKI Anwendung. Bei den einfacheren geht es hauptsächlich um die zuverlässige Verbreitung der öffentlichen Schlüssel. Die einfachste Form ist *Direct Trust*. Dabei vertraut man direkt dem Kommunikationspartner, der selbst seinen öffentlichen Schlüssel übermittelt (z.B. per Mail, Brief, etc.). Eine solche Organisation ist natürlich nur für den Kleinstbereich anwendbar.

Für kleinere Netze existiert ein Vertrauensmodell mit gegenseitigem Vertrauen aller Teilnehmer (*Web of Trust*). Zunächst funktioniert dies wie Direct Trust: Man vertraut gezielt seinen Kommunikationspartnern. Will man nun mit jemandem kommunizieren, dessen öffentlichen Schlüssel man noch nicht besitzt, so wird ein weiterer Weg geboten.

Ein Beispiel: A vertraut B, B vertraut C. Möchte nun A mit C kommunizieren, benötigt A den öffentlichen Schlüssel von C. Dies wird gelöst, indem B den öffentlichen Schlüssel von C signiert und damit für dessen Echtheit bürgt. Da A B vertraut, akzeptiert A somit den signierten öffentlichen Schlüssel von C als echt.

Web of Trust funktioniert dabei auch über mehrere Zwischenstationen. Für den Heimgebrauch ist diese Form der PKI durchaus ausreichend und wird auch angewandt (z.B. PGP). Für die professionelle Anwendung ist jedoch allein die hierarchische Organisation (*Hierarchical Trust*) im Einsatz. Das deutsche Signaturgesetz (siehe weiter unten) schreibt diese Organisationsform für rechtsgültige digitale Signaturen vor. Deshalb werde ich hier nur auf diese PKI näher eingehen.

Um neue Teilnehmer eindeutig zu identifizieren und deren öffentliche Schlüssel zu verbreiten, existiert in der hierarchischen PKI eine neutrale Stelle. Alle Teilnehmer des Systems müssen dieser Einrichtung vertrauen. Sie bürgt auch vor Gericht dafür, dass der Urheber einer Signatur zweifelsfrei ermittelt werden kann. Diese neutrale Stelle nennt sich *Trustcenter*.

Die Hauptkomponenten eines Trustcenters sind (Abbildung 3.2):

- Zertifizierungsinstanz oder englisch *Certification Authority* (CA)
- Registrierungsinstanz oder englisch *Registration Authority* (RA)
- Verzeichnisdienst oder englisch *Directory* (DIR)

Wie der Name schon andeutet, ist das Ausstellen von Zertifikaten die Hauptaufgabe der CA. Zertifikate sind elektronische Dokumente, mit denen der öffentliche Schlüssel eines Teilnehmers mit seinem Namen verbunden und verbreitet wird. Die Zertifikate sind von der ausstellenden Instanz (CA) signiert. Damit bürgt die CA für die korrekte Aufnahme der Daten und deren Unverfälschtheit[3].

Typische Zertifikate enthalten:

1. Den Namen, auf den das Zertifikat ausgestellt ist,
2. den Namen der CA,
3. den öffentlichen Schlüssel des Teilnehmers,
4. den öffentlichen Schlüssel der CA,
5. eine CA interne fortlaufende Zertifikatsseriennummer,
6. den Gültigkeitszeitraum des Zertifikats und
7. die digitale Signatur der CA, mit der sie die Angaben des Zertifikats bestätigt.

So genannte Attributzertifikate können neben diesen Daten auch weitere Angaben zur Person enthalten. Dies wird oft benötigt, da der Name nicht eindeutig sein muss. Nach dem deutschen Signaturgesetz werden noch weitere Inhalte für Zertifikate vorgeschrieben.

Um eine reibungslose Interoperabilität zu gewährleisten, gibt es einen Standard für digitale Zertifikate, der von allen Trustcentern verwendet wird: *X.509*. Die verbreiteten PKI-Standards *ISIS* und *PKIX* (siehe weiter unten) setzen auf X.509 als Standard für den Austausch von digitalen Zertifikaten.

Der Standard X.509v1 aus dem Jahr 1988 definierte die Felder eines Zertifikats. Mit X.509v2 wurden noch weitere Felder hinzugenommen. Jedoch waren beide Standards nicht optimal für die Anforderungen einer modernen PKI geeignet. Unter anderem, weil sie mit X.500 Namen zur Identifikation von Teilnehmer und CA arbeiteten. Diese Verzeichnisdienststrukturen waren nicht für die Verwendung im Web geeignet.

Der X.509v3 Standard brachte 1996 dann die nötigen Anpassungen für moderne PKI-Systeme mit. Es können dort beliebige neue Felder definiert werden. Damit konnten einige neue Standardfelder deklariert werden, die nun bei digitalen Signaturen üblich sind. Die erforderlichen Felder, für digitale Zertifikate nach dem Signaturgesetz, werden weiter unten, bei der Einführung in das Signaturgesetz, beschrieben.

Die ausgestellten Zertifikate können schon vor ihrem eigentlichen Ablaufdatum für ungültig erklärt werden. Dies passiert, wenn ein Schlüssel kompromittiert wurde (z.B. die Chipkarte des Teilnehmers verloren gegangen ist) oder ein Teilnehmer kündigt. Aus diesem Grund geben CAs regelmäßig Sperrlisten (*Certificate Revocation Lists, CRL*) heraus, die alle ungültigen Zertifikatsseriennummern enthalten.

[3] In diesem Dokument kann mit CA jedoch auch das komplette Trustcenter gemeint sein. Dies ist in der Literatur heute allgemein üblich.

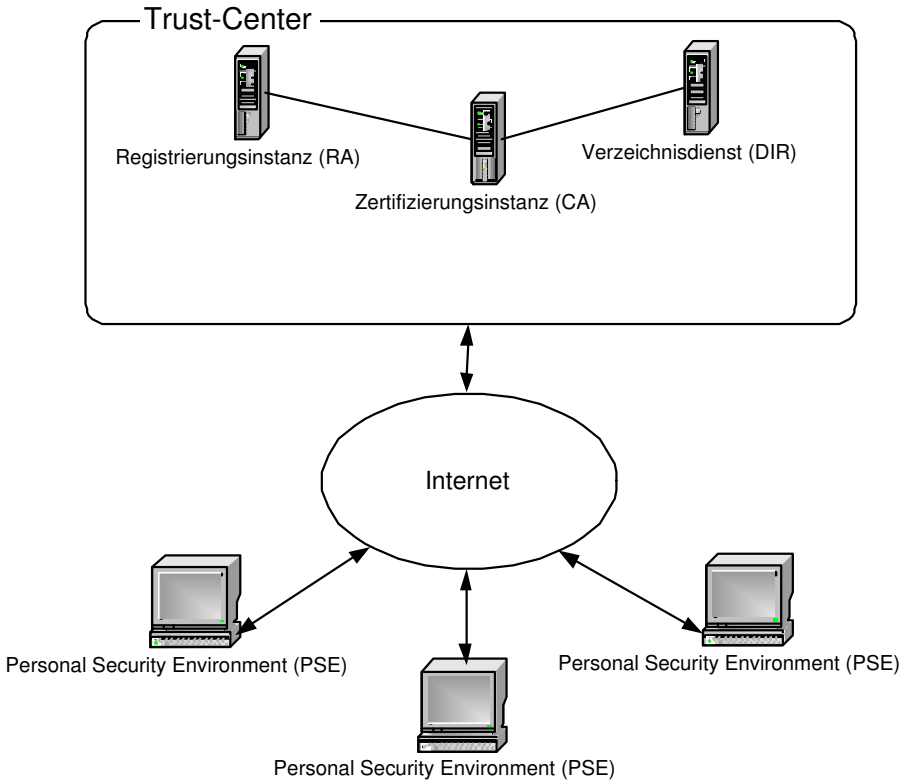


Abbildung 3.2: Die Bestandteile eines Trust-Centers

Die Registrierungsinstanz ist die Anmeldestelle, bei der sich alle Teilnehmer ausweisen müssen. Üblich ist hier persönlich zu erscheinen und sich mit dem Personalausweis zu identifizieren. Diese Daten werden an die CA übermittelt, die daraus ein Zertifikat erstellt. Der Verzeichnisdienst schließlich dient zur Verbreitung der Zertifikate. Alle Teilnehmer beziehen über ihn die benötigten Zertifikate und Sperrlisten.

Mit dieser PKI ist es für die Überprüfung von Signaturen lediglich nötig, sich das passende Zertifikat und die aktuelle Sperrliste über den Verzeichnisdienst zu besorgen. Um sicher zu gehen, dass das Zertifikat auch von dem Trustcenter stammt und nicht verändert wurde, ist es darüber hinaus auch erforderlich, den öffentlichen Schlüssel des Trustcenters zu erhalten. Dies kann natürlich etwas Aufwand erfordern und mit einem persönlichen Termin beim Trustcenter oder einem Telefonanruf verbunden sein, jedoch muss das nur einmal geschehen, um alle weiteren Zertifikate überprüfen zu können.

Diese zentrale Stellung eines Trustcenters hat einen großen Vorteil: Die Betriebsgrundsätze (*Policy*) werden zentral vereinbart. Somit arbeiten alle Teilnehmer auf dem gleichen Sicherheitslevel. Dazu gehört ein regelmäßiger Schlüsselwechsel, festgelegte Schlüssellängen und zentrale Sperrungen von Schlüsseln, falls diese kompromittiert wurden.

Allerdings ist es in der Realität häufig so, dass die Teilnehmer unterschiedliche Trustcenter verwenden. Das hierarchische Vertrauensmodell (Abbildung 3.3) sieht vor, dass sich die einzelnen Trustcenter, bei einem weiteren so genannten Wurzel-Trustcenter, zertifizieren lassen. Dort können die Teilnehmer dann die Zertifikate der einzelnen Trustcenter anfordern. Die Wurzelinstanz garantiert mit ihrer Signatur für die Echtheit der öffentlichen Schlüssel. Nur noch ihr öffentlicher Schlüssel muss den Teilnehmern bekannt sein. In Deutschland ist als oberstes Trustcenter die Regulierungsbehörde für Telekommunikation und Post (RegTP) zuständig.

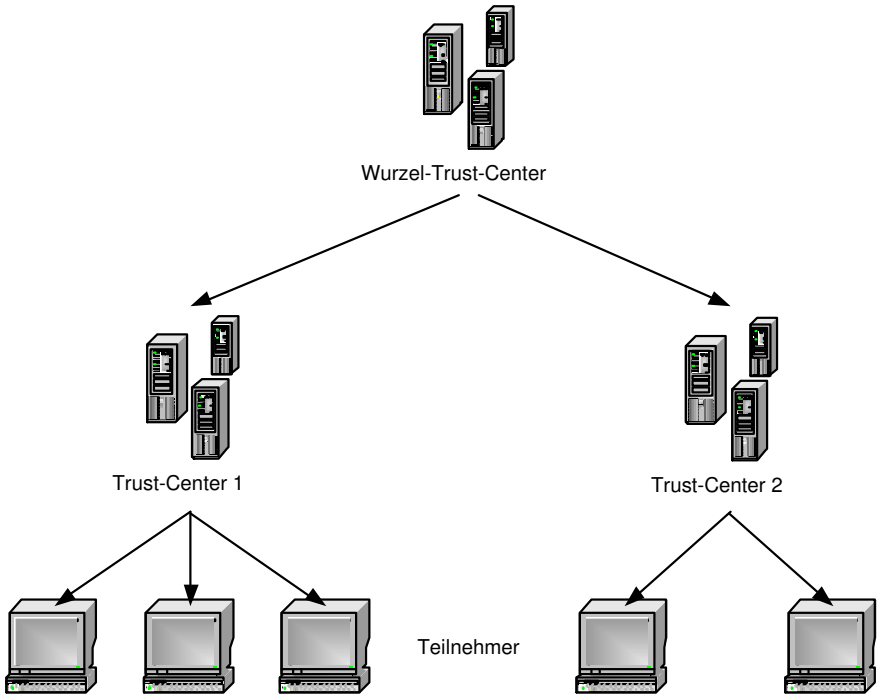


Abbildung 3.3: Der Aufbau einer hierarchischen Public-Key-Infrastruktur.

Neben diesen Hauptaufgaben einer PKI bieten die Trustcenter meist weitere Dienstleistungen an. Diese helfen das Signatursystem einfacher zu machen und erhöhen die Sicherheit weiter. Hervorzuheben sind dabei der *Zeitstempeldienst* und der *OCSP-Dienst* (Online Certificate Status Protocol). Sie werden in e.P.b. vor allem für die langfristige Beweiserhaltung der Signaturen eingesetzt.

### 3.1.4 Zeitstempel

Zu den Dienstleistungen eines Trustcenters gehört oft ein Zeitstempeldienst (englisch *Time-stamping Server*, TSS). Für Zeitstempel existiert ein Standard von PKIX, der als Request For Comments (RFC) veröffentlicht wurde [21].

Das deutsche Signaturgesetz definiert qualifizierte Zeitstempel als

> **§ 2 Abs. 14:** [...] elektronische Bescheinigungen des Zertifizierungsdiensteanbieters [hier: des Trustcenters, Anm. d. Autors] [...] darüber, dass ihm bestimmte elektronische Daten zu einem bestimmten Zeitpunkt vorgelegen haben.

Der Gesetzestext bringt den Zweck des Zeitstempels auf den Punkt. Der prinzipielle Ablauf:

1. Man sendet den Hashwert der betreffenden Daten an den Zeitstempeldienst.
2. Dieser fügt dem Hashwert eine Information über den aktuellen Zeitpunkt hinzu und signiert das Paket.
3. Der Zeistempeldienst übermittelt die gewonnene Signatur.

Für die Signatur verwendet der Zeitstempeldienst ein eigenes Zertifikat, welches von der zugehörigen CA ausgestellt wurde.

Ein Hauptzweck für den Einsatz von Zeitstempeln ist, signierte Dokumente mit einem zuverlässigen Zeitpunkt zu versehen und sie damit länger gültig zu halten. Läuft nämlich das Zertifikat, welches der Signatur zugrunde liegt, aus, so beweist der vorher angehängte Zeitstempel, dass die Signatur vor dem Ablauf stattfand. Damit ist nachvollziehbar, dass die Signatur zu einem Zeitpunkt entstanden ist, als das Zertifikat noch gültig war.

Das Dokument ist dabei aber leider nicht für alle Ewigkeit beweiskräftig. Das e.P.b.-Konzept für die weitere Beweiserhaltung wird in einem Kapitel weiter unten beschrieben.

### 3.1.5 OCSP

Ebenso wie der Zeitstempel ist auch diese weitere optionale Dienstleistung von Trustcentern für e.P.b. interessant: OCSP-Abfragen (*Online Certificate Status Protocoll*). OCSP ist ein Protokoll für die Sperrprüfung von Zertifikaten. Es wurde im Juni 1999 von der *IETF* (The Internet Engineering Task Force) und PKIX vorgestellt [20].

Der prinzipielle Ablauf:

1. Man sendet eine Nachricht mit den zu überprüfenden Zertifikatsseriennummern an den Zertifikateserver des Trustcenters.
2. Dieser antwortet mit einer signierten Nachricht über den aktuellen Status der Zertifikate. Bei Sperrungen von Zertifikaten wird normalerweise auch der Sperrgrund genannt.

Per OCSP kann online überprüft werden, ob ein Zertifikat im Moment gültig ist. In der Antwort ist die Gültigkeitsdauer der Auskunft vermerkt. Weil diese Antworten signiert sind, können sie zur Beweissicherung einer Signatur aufbewahrt werden.

In e.P.b. werden diese Antworten gespeichert, um nach Jahren den letzten Gültigkeitszeitpunkt der verwendeten Zertifikate nachvollziehen zu können. Dies wird nötig, da die Zertifizierungsdiensteanbieter die ausgegebenen Zertifikate nicht lange genug aufbewahren.

Die Alternative zu OCSP ist der regelmäßige Abruf von Zertifikatssperrlisten (CRLs). Das hat gegenüber OCSP zwei Nachteile: Die Sperrlisten sind groß und das regelmäßige Abrufen verursacht einigen Netzwerkverkehr. Außerdem ist eine zeitnahe Sicherstellung von Zertifikatsgültigkeiten nicht möglich. Die Sperrung eines Zertifikates wird erst beim nächsten Abruf der Sperrliste bemerkt.

### 3.1.6 Standards für PKI

Es existieren einige Standards im Umfeld von PKI. Diese Standards regeln die Kommunikation zwischen den einzelnen Bestandteilen einer PKI. Dabei geht es vor allem um Protokolle und Datenformate. Die detaillierte Beschreibung der Standards ist nicht Teil dieser Arbeit. Sie ist bei dem vorliegenden Grad der Abstraktion nicht von Bedeutung. Im hinteren Teil dieser Arbeit werden Java-API Methoden vorgestellt, die sich dieser Standards bedienen. An diesen Stellen wird das benötigte Hintergrundwissen zusammengefasst vermittelt. Die folgende Liste soll lediglich einen Überblick über die gebräuchlichsten Standardisierungen bieten.

Hier schlagwortartig die wichtigsten Standards für das PKI-Umfeld im Überblick:

- **X.509** - Standard für Zertifikate und Sperrlisten[4].
- **IETF -Public-Key Infrastructure (X.509) (PKIX)** - Standards für alle Bereiche einer PKI[5].
- **IETF - Simple Public Key Infrastructure (SPKI)** - Standards für PKI. Einfacher im Vergleich zu PKIX[6].
- **RSA-Security - PKCS** - zum größten Teil Datenformate für Public-Key-Kryptografie[7].
- **TeleTrusT, T7 - ISIS-MTT** - Standards für alle Bereiche einer PKI mit speziellen Ergänzungen für SigG-Konformität[8].

Allen voran ist der ISIS-MTT Standard in Deutschland wichtig, allein schon deshalb, weil er aus einem Zusammenschluss verschiedener deutscher Trustcenterbetreiber heraus entstanden ist. Diese Trustcenter unterstützen selbstverständlich den Standard. Doch auch über die Grenzen von Deutschland hinaus wird auf diesen Standard zurückgegriffen.

### 3.1.7 Digitale Signatur in der praktischen Anwendung

Bisher wurden die technischen Hintergründe von Signatursystemen vorgestellt. Doch wie präsentiert sich ein solches System für den Anwender?

[4] siehe PKIX oder ISIS-MTT
[5] `http://www.ietf.org/html.charters/pkix-charter.html` [Stand: 18.05.2004]
[6] `http://www.ietf.org/html.charters/spki-charter.html` [Stand: 18.05.2004]
[7] `http://www.rsasecurity.com` [Stand: 18.05.2004]
[8] `http://www.teletrust.de/anwend.asp?id=30410` [Stand: 18.05.2004]

Das Konzept für e.P.b. sieht vor, die Beamten mit Smartcards auszurüsten. Diese Prozessorchipkarten enthalten Funktionen zum Verschlüsseln und Signieren von Nachrichten. Dafür beherbergen sie den privaten Schlüssel und das Zertifikat des Inhabers. Sie sind also nicht übertragbar.

Im Gegensatz zu den alten Magnetstreifenkarten können Smartcards nicht kopiert werden. Dafür sorgen verschiedene Sicherheitsvorkehrungen. Der private Schlüssel verlässt bei Gebrauch nie die Karte. Alle Kryptofunktionen, die auf diesen Schlüssel zugreifen, laufen direkt auf der Karte.

Soll nun eine Nachricht signiert werden, so muss der Inhaber seine Smartcard in den Kartenleser einführen. Dann gibt er sein persönliches Passwort für diese Karte ein. Mit diesem Passwort schaltet der Inhaber die Karte für einen Signaturvorgang frei. Die Anwendung übergibt nun den Hashwert an die Karte. Diese verschlüsselt den Wert und gibt die Signatur zurück.

Der Beamte erhält seine persönliche Smartcard bei der Registrierungsinstanz des Trustcenters. Dort zeigt er seinen Personalausweis vor und bekommt ein Zertifikat ausgestellt. Das Zertifikat und der zugehörige private Schlüssel werden in eine Smartcard eingetragen. Verliert der Beamte seine Karte, so muss er dies sofort melden und die Karte sperren lassen.

## 3.2 Das SSL-Protokoll

Die folgenden Ausführungen legen die Beschreibungen in [22], die RFC 2246 [15] und die Referenz der Java SSL Implementierung JSSE [2] zugrunde.

Die Abkürzung SSL steht für *Secure Socket Layer*. Das Protokoll setzt auf dem TCP-(Transport)-Protokoll auf und bietet auf dieser Ebene Verschlüsselung, Authentifizierung und Integritätsschutz der Datenpakete. Die Anwendungsschicht spricht also nicht direkt das TCP-Protokoll an, sondern verwendet das SSL-Protokoll und fügt seiner Kommunikation damit nach außen die genannten Merkmale hinzu.

SSL wurde von der Firma *Netscape* 1994 entwickelt und hat sich heute zu einem übergreifenden Standard entwickelt. 1999 wurde SSL von der *IETF* standardisiert und in *TLS* (Transport Layer Security) umbenannt. TLS ist bis heute noch nicht weit verbreitet, weshalb hier weiterhin von SSL die Rede sein wird. Da TLS abwärtskompatibel zu den vorigen SSL-Versionen 1.0, 2.0 und 3.0 ist und nur geringe Weiterentwicklungen aufweist, kann alles, was hier über SSL gesagt wird, für TLS übernommen werden.

Der breiten Masse wurde SSL bekannt, als immer mehr Internetshops die sichere Übermittlung von Zahlungsdaten mit dieser Technik realisierten. Dies führte dazu, dass heute praktisch jeder Internetbrowser das Protokoll unterstützt. Bei der Absicherung von Webseiten ist SSL heute also erste Wahl.

Jedoch ist SSL nicht nur dazu in der Lage HTTP-Verbindungen abzusichern. Als Transportprotokoll kann SSL von praktisch jedem Anwendungsprotokoll genutzt werden, welches auf TCP aufsetzt. UDP wird von SSL nicht unterstützt. Auch Webservices lassen sich per SSL um die genannten Sicherheitsmerkmale erweitern.

Im e.P.b.-Sicherheitskonzept wird SSL zur Absicherung der Kommunikation zwischen den e.P.b.-Archivservern und auch zwischen lokalen Clients und zugehörigem e.P.b.-Archivserver eingesetzt. Mehr dazu weiter unten.

### 3.2.1 Die Technik

SSL ordnet sich im Protokollstapel als zusätzliche Schicht zwischen TCP-Schicht und Anwendungsschicht ein. Die Schnittstelle zwischen Anwendungsprogramm und TCP-Protokoll sind Sockets. SSL bietet nun den Anwendungen eigene Sockets, die von den Anwendungen wie TCP-Sockets verwendet werden. Somit können auch bestehende Anwendungen leicht um SSL-Funktionalität erweitert werden.

Ein weiterer Vorteil von SSL ist, dass es sehr nahe an der Anwendungsschicht liegt. So können die Anwendungen, die SSL einsetzen, auch in das Protokoll eingreifen. Dies ist nützlich, um SSL besser an die Anwendung anzupassen.

Die Pakete werden bei SSL über das TCP-Protokoll gesendet. Genau wie TCP handelt es sich bei SSL um ein verbindungsorientiertes Protokoll. Das verbindungslose UDP wird nicht unterstützt.

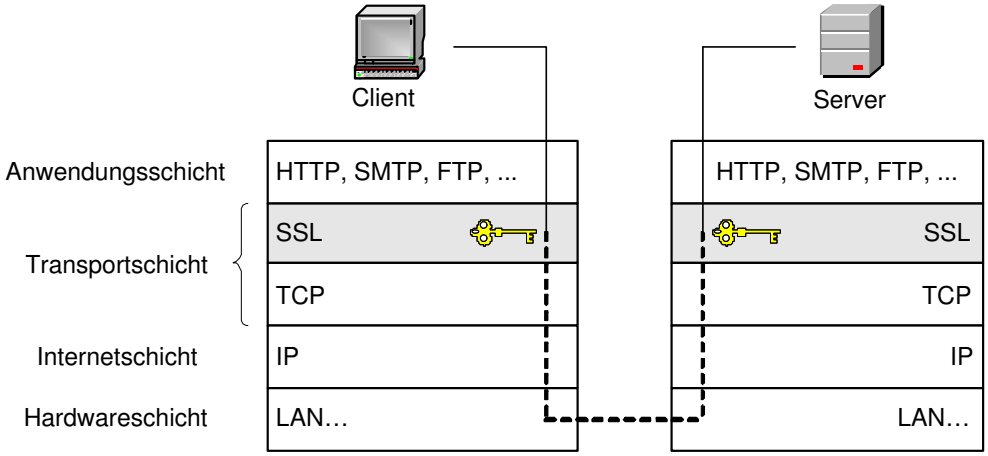


Abbildung 3.4: Im ISO-OSI-(Schichten)-Modell befindet sich SSL in der Transportschicht. (Nach Abbildung 24-1 aus [22].)

Beim Handshaking vor der eigentlichen, verschlüsselten Kommunikation verhandelt SSL die verwendeten Kryptoalgorithmen und Schlüssellängen zwischen den zwei Teilnehmern aus. Dabei werden die Algorithmen vereinbart, die von beiden Partnern unterstützt werden und den besten Kompromiss zur Sicherheit darstellen.

Es werden symmetrische Verschlüsselungsverfahren wie *DES*, *IDEA* und *RC4*, die kryptografischen Hashfunktionen *SHA-1* und *MD5* sowie *RSA* und *Diffie-Hellman* für den Schlüsselaustausch angeboten. Die digitale Signatur wird von diesem Protokoll lediglich zur Authentifizierung der Partner eingesetzt (*POP* - Proof of Posession). Zur Übermittlung öffentlicher Schlüssel werden digitale Zertifikate verwendet.

Sind nach dem Handshaking die Sitzungsschlüssel ausgetauscht, erfolgt die Kommunikation von nun an symmetrisch verschlüsselt. Alle über SSL ausgetauschten Daten werden als SSL-Record verpackt und versendet.

Insgesamt definiert SSL fünf Protokolle:

- Das *Handshake-Protokoll* regelt den Verbindungsaufbau.
- Das *ChangeCipherSpec-Protokoll* gibt nach einem erfolgreichen Handshaking die gewählten Parameter an die *Record*-Schicht weiter.
- Das *Alert-Protokoll* definiert Fehler- und Warnmeldungen.
- Das *ApplicationData-Protokoll* regelt den Datenaustausch zwischen der Anwendung (Socketschnittstelle) und dem Record-Protokoll.
- Das *Record-Protokoll* definiert den Aufbau der Datenpakete. Es sorgt dafür, dass ein Hashwert, zum Überprüfen der Integrität der Pakete, erzeugt wird. Vor der Übergabe an die TCP-Ebene werden die Pakete dann noch samt Hashwert mit den gewählten Parametern verschlüsselt.

### 3.2.2 Der Ablauf der Kommunikation

Vor der eigentlichen Kommunikation werden mit dem *Handshake-Protokoll* die Kommunikationspartner gegenseitig authentifiziert, die Kryptoalgorithmen ausgehandelt und die Sitzungsschlüssel ausgetauscht.

Der Ablauf des Handshake-Protokolls:

1. *Client hello* - Der Client sendet seinen Wunsch nach einer mit SSL gesicherten Kommunikation und übermittelt dabei die beste verfügbare SSL-Version und eine Liste der unterstützten Kryptoalgorithmen und Schlüssellängen an den Server.
2. *Server hello* - Der Server wählt die sichersten Parameter und sendet diese Informationen an den Client.
3. *Certificate* - Der Server sendet dem Client sein Zertifikat und (wenn nötig) die ganze Zertifikatskette bis zur Wurzelinstanz.
4. *Certificate request* - Der Server fordert das Zertifikat des Clients an, um diesen identifizieren zu können.
5. *Server key exchange* - Sollten die Daten des Zertifikats zum Schlüsselaustausch nicht genügen oder wird gar kein Zertifikat eingesetzt, so kann der öffentliche Schlüssel auch separat übermittelt werden.
6. *Server hello done* - Der Server meldet dem Client das Ende der Nachrichten zum Schlüsselaustausch.
7. *Certificate* - Der Client sendet dem Server sein Zertifikat und (wenn nötig) die komplette Zertifikatskette.
8. *Client key exchange* - Der Client erzeugt einen symmetrischen Schlüssel, mit dem die Kommunikation verschlüsselt werden soll. Er übermittelt diesen, verschlüsselt mit dem öffentlichen Schlüssel des Servers.

9. *Certificate verify* - Der Client sendet nun eine digital signierte Nachricht an den Server, um den Besitz des privaten Schlüssels, der zu dem vorher übermittelten Zertifikat gehört, zu beweisen. (POP - Proof of Posession)
10. *ChangeCipherSpec* - Der Client fordert nun den Server auf, die Kommunikation zu verschlüsseln.
11. *Finished* - Der Client meldet seine Bereitschaft zur sicheren Kommunikation.
12. *ChangeCipherSpec* - Der Server meldet, dass er nun die Kommunikation verschlüsselt.
13. *Finished* - Der Server meldet seine Bereitschaft zur sicheren Kommunikation.
14. *Encrypted data* - Ab nun kommunizieren Client und Server verschlüsselt mit dem ausgetauschten symmetrischen Schlüssel.

Nachdem das Handshake-Protokoll einmal durchgeführt wurde, laufen spätere Verbindungsaufnahmen schneller ab, da die gleichen Parameter wiederverwendet werden können.

Per *ChangeCipherSpec-Protokoll* wird nach erfolgreichem Handshaking dem *Record-Protokoll* mitgeteilt, auf welche Kryptoalgorithmen und Schlüssellängen sich die beiden Kommunikationspartner verständigt haben. Außerdem kommt das Protokoll immer dann zum Einsatz, wenn sich an diesen Parametern etwas ändert. Sobald das Protokoll abgearbeitet wurde, verwendet das *Record-Protokoll* die übermittelten Kryptoalgorithmen.

Mit dem *ApplicationData-Protokoll* werden die Daten zwischen Anwendungsschicht und SSL-Record-Protokoll übermittelt. Dieses Protokoll dient also als hauptsächliche Schnittstelle zur Anwendung.

Treten Fehler auf, werden Meldungen nach dem *Alert-Protokoll* versendet.

# 4. Die aktuelle Rechtslage für Signaturen

Die Ausführungen folgen zum Teil der Abhandlung in [22]. Grundlegend sind jedoch das deutsche Signaturgesetz [6] und die zugehörige Signaturverordnung [7]. Weitere wichtige Dokumente sind in diesem Zusammenhang die Begründung zur Verordnung [5] und der Maßnahmenkatalog von BSI und RegTP zur digitalen Signatur [14].

1997 war das deutsche Signaturgesetz (SigG) das erste seiner Art in der Welt. Ziel des Gesetzes war es, Rahmenbedingungen für elektronische Signaturen zu schaffen. Damit sollte sich die digitale Signatur offiziell als Alternative für die Unterschrift per Hand durchsetzen[1].

1999 trat die europäische Signaturrichtlinie in Kraft, welche eine Überarbeitung des deutschen Gesetzes erforderte. Die Richtlinie führte unterschiedliche Abstufungen von Signaturen ein: *fortgeschrittene Signaturen* mit entweder *einfachen* oder *qualifizierten Zertifikaten*. Die Anpassung des deutschen Signaturgesetzes trat 2001 in Kraft und besteht in dieser Form bis heute.

Die Bezeichnung aus der Signaturrichtlinie *„fortgeschrittene elektronische Signatur mit qualifiziertem Zertifikat“* wurde in die kürzere *„qualifizierte elektronische Signatur“* umgetauft. Daneben gibt es die *„fortgeschrittene elektronische Signatur“*. Eine fortgeschrittene Signatur gilt zwar als Beweismittel vor Gericht, sie ersetzt jedoch nicht die Unterschrift per Hand. Dazu wird eine qualifizierte Signatur benötigt. Welche Voraussetzungen ein qualifiziertes Zertifikat mitbringen muss, ist im nächsten Abschnitt beschrieben.

Das alte Gesetz machte noch eine Akkreditierung von Trustcentern, welche qualifizierte Zertifikate erzeugen wollten, zur Voraussetzung. Das bedeutet, sie mussten sich einer regelmäßigen technischen Überprüfung durch die RegTP unterziehen. Im Gesetz von 2001 können qualifizierte Zertifikate auch von nicht akkreditierten Trustcentern herausgegeben werden. Die Nutzung von akkreditierten Trustcentern wird jedoch immer noch bevorzugt. Die von ihnen herausgegebenen Zertifikate heißen *„qualifizierte Zertifikate mit Anbieterakkreditierung“*.

Das Gesetz selbst definiert die Fachbegriffe und Regularien für digitale Signaturen. Eine Festlegung auf bestimmte technische Standards wird hier nicht gemacht. Die Signaturverordnung (SigV) konkretisiert diesen Punkt: Sie beschreibt unter anderem das Akkreditierungsverfahren für Trustcenter mit Kosten und Sicherheitskonzepten. Technische Einzelheiten enthält außerdem der Maßnahmenkatalog des Bundesamtes für Sicherheit in der Informationstechnik (BSI).

In den folgenden Abschnitten wird nun auf Einzelheiten aus dem Signaturgesetz und der Signaturverordnung, die für ein e.P.b.-System wichtig sind, eingegangen.

[1] Digitale Signaturen wurden auch schon vor dem Gesetz teilweise als Beweismittel anerkannt (Prinzip der freien Beweiswürdigung). Das Gesetz gab jedoch erstmals für die Gerichte bindende Rahmenbedingungen vor.

## 4.1 Details aus dem deutschen Gesetz

Nach der Definition der Fachbegriffe geht der zweite Abschnitt des SigG auf die Anforderungen an die Zertifizierungsdiensteanbieter[2] ein. § 4 sieht vor, dass der Betrieb eines Trustcenters grundsätzlich genehmigungsfrei ist. Die Betreiber müssen jedoch die Rechtsvorschriften aus § 24 einhalten und ausreichende Fachkunde und finanzielle Deckung (250.000 Euro) mitbringen. Die Aufnahme des Betriebes muss außerdem bei der RegTP angemeldet werden.

Stellt das Trustcenter den Betrieb ein, so muss dies gemeldet werden. Alle ausgestellten qualifizierten Zertifikate müssen dann entweder gesperrt oder an ein anderes Trustcenter übergeben werden.

Jedes Trustcenter darf qualifizierte Zertifikate herausgeben. Voraussetzungen dafür sind:

- eine zuverlässige Prüfung der Identität des Antragsstellers
- Das Zertifikat muss bei Zustimmung des Inhabers jederzeit abrufbar gehalten werden.
- Weitere (über die SigG Forderungen hinausgehende) personenbezogene Daten dürfen nur mit Zustimmung des Betroffenen in das Zertifikat einbezogen werden.
- Anstatt des wirklichen Namens kann ein Pseudonym im Zertifikat verwendet werden.
- Der Trustcenterbetreiber muss die Sicherheit des Kryptosystems hinsichtlich der Geheimhaltung von privaten Schlüsseln und der Eignung von Algorithmen sicherstellen.
- Die eingesetzten Produkte und Verfahren müssen SigG-konform sein.
- Der Trustcenterbetreiber muss sich über die sichere private Kryptoausrüstung des Antragsstellers ein Bild verschafft haben.

§ 6 erläutert, welche Unterrichtspflicht der Trustcenterbetreiber gegenüber dem Antragssteller hat. Es ist vorgeschrieben, dass der Trustcenterbetreiber seine Kunden über die Rahmenbedingungen für einen sicheren Einsatz der Signatur informieren muss.

Dazu gehört auch, dass der Kunde erfahren muss, welche Maßnahmen nötig sind, um eine langfristige Gültigkeit von Signaturen zu erreichen. Dem Kunden muss ebenso erklärt werden, dass seine elektronischen Signaturen die gleiche Bewertung im Rechtsverkehr haben, wie seine Unterschrift per Hand. Den Erhalt dieser Informationen bestätigt der Kunde mit seiner Unterschrift. Erst dann darf dem Kunden sein qualifiziertes Zertifikat überreicht werden.

Nach SigG müssen *qualifizierte Zertifikate* folgenden Inhalt haben:

- der Name des Schlüsselinhabers oder ein Pseudonym[3]
  (Der richtige Name muss jedoch beim Trustcenter hinterlegt und bei berechtigten Ansprüchen Dritter herausgegeben werden.)

[2] Wortlaut SigG: „Zertifizierungsdienstebetreiber - natürliche oder juristische Personen, die qualifizierte Zertifikate oder qualifizierte Zeitstempel ausstellen“

[3] Beides soll jeweils unverwechselbar sein. Besteht bei einem Namen die Gefahr einer Verwechslung, ist ein Zusatz beizufügen.

- der zugehörige öffentliche Schlüssel
- die Algorithmen, für die das Schlüsselpaar verwendet werden darf
- eine interne fortlaufende Nummer des Trustcenters
- der Gültigkeitszeitraum
- der Name des Trustcenters und des Staates mit dem Firmensitz
- mögliche Beschränkungen für den Einsatz des Schlüssels (*policy*)
- die Angabe, dass es sich um ein qualifiziertes Zertifikat handelt
- optional weitere Attribute des Inhabers
- die qualifizierte elektronische Signatur des Trustcenters.

Die optionalen, weiteren Attribute können auch in einem zusätzlichen Zertifikat ausgelagert werden, welches eindeutigen Bezug auf das qualifizierte Zertifikat nimmt. Dieses zusätzliche Zertifikat muss ebenfalls vom Trustcenter signiert werden. Beide Zertifikate müssen jederzeit zu sperren sein. Ausgegebene Zertifikate und Sperrlisten müssen aufbewahrt werden und jederzeit nachprüfbar sein. Diese Informationen dürfen nachträglich nicht mehr unbemerkt geändert werden können.

Dem Inhaber des Signaturschlüssels muss Einblick in die über ihn gespeicherten Daten gestattet werden. Bei Einstellung des Betriebes geht die Dokumentation entweder an das Trustcenter über, welches die Zertifikate übernimmt, oder wird bei der RegTP hinterlegt.

§ 11 legt die Haftung für Trustcenter fest. Bei Versagen der Sicherheitsinfrastruktur eines Trustcenters, muss dieses für die entstandenen Schäden durch Missbrauch aufkommen, wenn der Trustcenterbetreiber schuldhaft gehandelt hat.

Der dritte Abschnitt behandelt das Akkreditierungsverfahren. Dort ist beschrieben, welche Pflichten und Vorteile eine Akkreditierung mit sich bringt. Dazu gehört, dass die RegTP bei Einstellung des Betriebes die Weiterführung der bestehenden Verträge mit Signaturschlüsselinhabern weiterführen oder weitervermitteln muss. Ein akkreditiertes Trustcenter erhält die für seinen Betrieb nötigen qualifizierten Zertifikate direkt von der RegTP.

Im vierten Abschnitt wird auf die technische Sicht näher eingegangen. Dabei lassen die Formulierungen viel Raum und legen sich nicht auf bestimmte Produkte oder Standards fest.

Die Software für die Erzeugung digitaler Signaturen muss vor der Signatur dem Benutzer klar anzeigen, dass nun eine Signatur erzeugt wird und auf welche Daten sich die Signatur bezieht. Software zum Überprüfen von Signaturen müssen anzeigen, auf welche Daten sich die Signatur bezieht und Komponenten bereitstellen die überprüfen, ob diese unverändert sind, wer die Signatur erzeugt hat, welche Inhalte das verwendete Zertifikat hat und ob das Zertifikat noch gültig ist oder nicht.

Die Aufsicht über die korrekte Einhaltung des Gesetzes liegt in der Verantwortung der RegTP. Sie kann bei Verstößen die Trustcenter sperren. Dabei bleibt die Gültigkeit der ausgestellten Zertifikate jedoch erhalten.

Innerhalb der EU sind qualifizierte Signaturen gleichgestellt, wenn der Anbieter die Anforderungen der EU-Richtlinie erfüllt. Für ausländische Trustcenter sind die Signaturen je nach der dort gültigen Signaturrichtlinie ebenfalls gleichgestellt.

## 4.2 Wichtige Punkte der Verordnung

Die Signaturverordnung geht zunächst genauer darauf ein, welche Nachweise für die Anmeldung eines Trustcenters zu erbringen sind und wie die Aufnahme der personenbezogenen Daten zu erfolgen hat. Weitere Regelungen betreffen die Erzeugung, die Übergabe und weitere Aufbewahrung der privaten Schlüssel. Diese müssen unter strengsten Sicherheitsmaßnahmen geschehen. Akkreditierte Trustcenter müssen alle drei Jahre einer Sicherheitsinspektion unterzogen werden. Im Folgenden werden nur die aus der Sicht eines e.P.b. interessanten Aspekte der Verordnung aufgeführt.

Aus Sicht des e.P.b. ist § 4 interessant. Dieser schreibt vor, wie lange ausgegebene Zertifikate aufzubewahren sind. Dies sind mindestens 5 Jahre ab dem Ende des Jahres, an dem das Zertifikat ausläuft. Akkreditierte Trustcenter müssen die Zertifikate mindestens 30 Jahre aufbewahren.[4]

Für Sperrungen der Zertifikate müssen Trustcenter eine Rufnummer bekannt geben, über die der Kunde schnellstmöglich seine qualifizierten Zertifikate sperren lassen kann. Vor der Sperrung muss sich das Trustcenter von der Identität des Kunden überzeugen.

Die Kryptoalgorithmen, die nach dem Signaturgesetz zulässig sind, werden regelmäßig, nach Empfehlungen der RegTP, im Bundesanzeiger veröffentlicht. Diese Parameter sind verbindlich für qualifizierte Signaturen. Dabei geht es um Signatur-, Verschlüsselungs- und Hashalgorithmen und die dabei verwendeten Schlüssellängen. Daneben werden aber auch Anforderungen an die eingesetzte Technik gestellt. Hierbei zählt z.B. der Zufallszahlengenerator, der im Trustcenter zur Schlüsselgenerierung eingesetzt wird.

§ 17 schreibt explizit vor, dass Dokumente, die längere Zeit in signierter Form benötigt werden, vor dem Ende der Eignung von Algorithmen oder Parametern (z.B. Schlüssellängen) mit geeigneten Verfahren erneut zu signieren sind[5].

Im Anhang der Verordnung finden sich die Anforderungen an akkreditierte Signaturprodukte. Die Vorgaben richten sich nach den Sicherheitsprüfstufen der *„Common Criteria for Information Technology Security Evaluation“* (Bezeichnung „EAL“) oder der *„Information Technology Security Evaluation Criteria“* (Bezeichnung „E“). Die Hardware für die Signaturerstellung[6] muss mindestens der Prüfstufe „EAL4“ oder „E3“ entsprechen. Signaturanwendungskomponenten müssen mindestens die Prüfstufe „EAL3“ oder „E2“ erfüllen.

Die Akkreditierung erfordert deshalb die Überprüfung aller Phasen im Betrieb eines Trustcenters. Angefangen in der Planungsphase und wiederholte Prüfungen im laufenden Betrieb. Dies stellt hohe finanzielle Anforderungen an die Trust-Center-Betreiber. So können nur große, finanzstarke Unternehmen die begehrte Auszeichnung erhalten.

[4] Diese Aufbewahrungsfrist ist eindeutig zu kurz für e.P.b.. Das in dieser Arbeit entwickelte Konzept sieht deshalb vor, diese Daten selbst aufzubewahren.

[5] Nach allgemeiner Auffassung genügt auch ein erneuter qualifizierter Zeitstempel, der über die alte Signatur und das Dokument erstellt wird. Mehr dazu im später vorgestellten Konzept.

[6] z.B. die Smartcard

# 5. Blick über den Tellerrand

Neben den technischen und rechtlichen Hintergründen ist es bei der Entwicklung eines Signatur- und Sicherheitssystems für das e.P.b. nötig, einen Überblick über vergleichbare Entwicklungen, aktuelle Technologien und die Literatur in diesem Bereich zu haben. Dieser Rundblick ist notwendig, um nicht entgegen aktuellen Entwicklungen und Standards im Bereich des eGovernments zu agieren.

Wegen der schieren Masse solcher Quellen kann dieses Kapitel lediglich einen groben Überblick vermitteln. Neben der Vorstellung der einzelnen Themen wird deren Eignung und Relevanz bezüglich einer Anwendung im e.P.b. untersucht.

## 5.1 Das elektronische Grundbuch

Das elektronische Grundbuch war ein Vorreiter bei der Umstellung der Verwaltung auf elektronische Systeme. Dabei wurden die Grundbuchblätter eingescant und, nach einer manuellen Sichtkontrolle, in ein zentrales Archiv gestellt. Die hessischen Grundbücher sollen mit diesem Verfahren bis Ende 2004 vollständig aufgenommen worden sein.

Die erzeugten Abbilder werden auf *WORM-Medien* (write once - read many) abgelegt, sodass sie später nicht mehr verändert werden können. Weiterführungen werden nicht in die Bilddaten aufgenommen, sondern zusammen mit ihnen gespeichert. Sie werden beim Abruf in die Bilddaten eingeblendet (verschiedene Schichten). Eintragungen werden digital vom zuständigen Beamten unterschrieben.

Das System ist als Terminalserver ausgelegt. Die Anwendungen zur Einsichtnahme in das Grundbuch und zur Weiterführung der Grundbuchblätter laufen also nicht auf den Clients, sondern auf einem zentralen Server.

Zur Einführung dieses Systems wurden die Grundbuchordnung [1] und die Grundbuchverfügung (GBV) [3] angepasst, um den rechtlichen Rahmen zu bieten. Dort wurde die grundsätzliche Zulässigkeit der elektronischen Verarbeitung von Grundbuchdaten festgestellt. Außerdem wurden die Funktionsweise und die Art der Nutzung eines solchen Systems beschrieben und die anfallenden Sicherheitsbedürfnisse spezifiziert.

Die *Grundbuchverfügung* [3] stellt in *§ 64, Absatz 2* folgende Anforderungen an das System:

1. Die Nutzung darf ausschließlich durch authentifizierte und identifizierte Benutzer erfolgen.
2. Die Benutzerrechte müssen im System verwaltet werden.
3. Bei der Authentifizierung eines Benutzers müssen seine Benutzerrechte geprüft werden.
4. Alle Veränderungen und Ergänzungen müssen protokolliert werden.
5. Eine Wiederherstellung ausgefallener Systeme muss jederzeit möglich sein.

6. Verfälschungen der Daten durch Fehlfunktionen müssen erkennbar sein.
7. Fehlfunktionen des Systems müssen rechtzeitig gemeldet werden.
8. Die Übertragung von Daten zwischen Clients und Archiv, sowie zwischen verschiedenen Teilen des Archivs, muss gesichert erfolgen.

Die Erfüllung dieser Anforderungen entspricht dabei in Teilen nicht den aktuellen Sicherheitsstandards, gewährt aber dennoch eine angemessene Sicherheit. Die Kommunikation mit dem elektronischen Grundbuch (EGB) erfolgt über ISDN-Verbindungen. Zugangsgenehmigungen werden nur an Benutzer vergeben, die ein berechtigtes Interesse darlegen können. Die ISDN-Kennungen dieser Teilnehmer werden zum Zugriff autorisiert und die Teilnehmer erhalten ein Passwort.

Auch die Einsicht eines Grundbuchamtes in das Grundbuch eines anderen Amtes und die Einsicht Dritter (Behörden, Gerichte, Notare und Vermessungsingenieure) ist möglich. Das einsichtgewährende Amt entscheidet dabei jedoch über die Vergabe von Berechtigungen. Außerdem bedarf es der Genehmigung seitens der Landesjustizbehörden.

Durch Protokollierung aller Zugriffe kann die Zulässigkeit der Abrufe später nachvollzogen werden. Geprüft werden die Protokolle nur, wenn es einen konkreten Anlass dazu gibt oder zu routinemäßigen Stichproben.

Backups zur Wiederherstellung der Daten im Schadensfall werden täglich erneuert. Die Backups müssen so aufbewahrt werden, dass eine gleichzeitige Beschädigung des Archivs und der Backups ausgeschlossen ist.

In der GBV wird auch geregelt, wer feststellt, ob ein System die erwarteten Anforderungen erfüllt. Dies kann über ein inländisches oder ausländisches Prüfzeugnis belegt werden (ähnlich zu einer Akkreditierung) oder durch die zuständige Landesjustizverwaltung geprüft und festgestellt werden.

**Maßnahmen für ein e.P.b.:** Die acht Anforderungen, die für das elektronische Grundbuch gelten, können für ein e.P.b. übernommen werden. Das in dieser Arbeit entwickelte Konzept muss Maßnahmen vorgeben, die diese Anforderungen erfüllen. Ein weiteres Vorbild kann die Freigabe der Software durch eine Behörde darstellen. Damit könnte eine teure Akkreditierung der e.P.b.-Software nach dem Signaturgesetz vermieden werden. Dazu müssten jedoch zuerst für die Standesämtern die rechtlichen Rahmenbedingungen, ähnlich wie beim EGB, geschaffen werden.

## 5.2 OSCI

OSCI (Online Services Computer Interface) ist der Bremer Vorschlag auf den 1998 von der Bundesregierung ausgerufenen Städtewettbewerb MEDIA@komm, der Konzepte hervorbringen soll, die neuen Medien erfolgreich für eine bürgernahe öffentliche Verwaltung zu nutzen. Dabei stand das Thema „elektronische Signatur“ im Vordergrund. OSCI steht jedoch auch für eine Marke der *Bremen Online Services*, die ihre Produkte mit diesem Kürzel vermarkten.

Die Informationen über OSCI in diesem Abschnitt entstammen zum Teil [19]. Die Informationen zu Governikus entstammen einer Präsentation, die nicht publiziert ist, uns jedoch freundlicherweise von Herrn Klüber vom HZD zur Verfügung gestellt wurde.

*OSCI-Transport* ist ein Nachrichtenformat, das Online-Behördengänge, sowie behördeninterne Kommunikation vollständig und sicher ermöglicht. In Version 2.0 stellt es einen Standard für die komplette Sicherheits- und Signaturumgebung von den Rechnern der Beamten, über zentrale Vermittlungsstellen bis zur Software für den Bürger dar.

Der Bund beauftragte Bremen 1999, den OSCI-Standard zu entwickeln und in der Bremer Verwaltung prototypisch umzusetzen. Dies soll ein Signal an alle öffentlichen Verwaltungen sein, diese Lösungen zu inspizieren und ggf. OSCI als Standard zu übernehmen.

Weiterentwickelt wird der OSCI-Standard von dem *KoopA-ADV*. Diese OSCI-Leitstelle bemüht sich um weitgehende Hersteller- und Produktunabhängigkeit. Die Ergebnisse stehen der öffentlichen Verwaltung zur Verfügung, um weitestgehende Transparenz und Spielraum für die Entwickler kommunaler Software zu erreichen.

Ziel ist es, OSCI als übergreifenden Standard im eGovernment zur elektronischen Kommunikation zwischen Bürgern und öffentlicher Hand deutschlandweit durchzusetzen.

Das OSCI-Transportprotokoll ist ein sicheres Austauschformat für Nachrichten zwischen Behörden und Bürgern über unsichere Netze. Alle Beteiligten (insbesondere auch die teilnehmenden Bürger) verfügen über ein Zertifikat, mit zugehörigem privaten Schlüssel, auf einer Chipkarte und den dazu passenden Kartenleser.

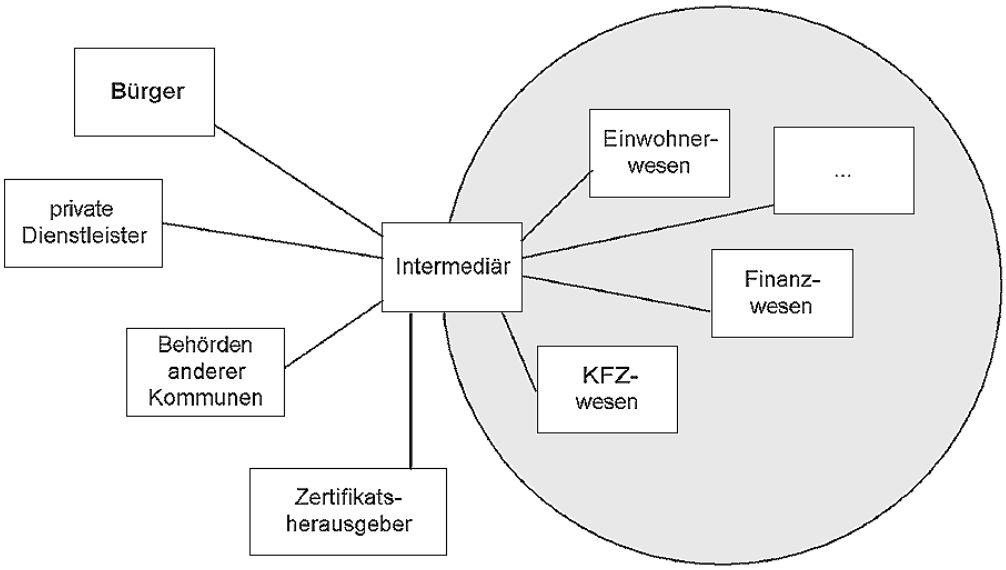


Abbildung 5.1: Der Intermediär steht bei OSCI im Mittelpunkt der Kommunikation. (Entnommen aus [19])

Die digitale Signatur aller ausgetauschten Nachrichten erfüllt alle Anforderungen an eine rechtssichere Kommunikation. Die Nachrichten sind klar einem Absender zuzuordnen und das Abschicken einer Nachricht lässt sich später nicht abstreiten.

Mittelpunkt und wichtigster Bestandteil von OSCI ist der so genannte Intermediär. Er hält Postkörbe für alle beteiligten Nutzer bereit und jegliche Kommunikation erfolgt durch ihn und unter seiner Kontrolle.

Aus technischer Sicht besteht der Intermediär als Webservice. Er hat Zugriff auf einen Backendserver, der Speicher für die Nachrichten und zusätzliche Funktionalität bereitstellt. Direkten Zugriff auf diesen Server von anderen, an OSCI-Kommunikation Beteiligten, ist ausgeschlossen.

Alle versendeten Nachrichten werden mit dem öffentlichen Schlüssel des Empfängers verschlüsselt, sodass nur dieser den Inhalt lesen kann. Auch der Intermediär hat keinen Zugriff auf die unverschlüsselten Inhalte. Damit dieser jedoch die Nachricht verarbeiten und weiterleiten kann, sieht das Nachrichtenformat von OSCI drei Schichten vor:

Die versendeten Dokumente werden als Inhaltsdaten bezeichnet. Diese werden signiert und verschlüsselt versendet. Nur der Empfänger kann diese lesen. Die zugehörigen Zertifikate sind in der Auftragsebene der Nachricht abgelegt. Durch diese Verschlüsselung hat auch der Intermediär keinen Zugriff auf diese Daten und kann die Signatur des Dokumentes nicht prüfen. Somit muss der Client selbst diese Überprüfung vornehmen.

Nachrichten werden von einem Knoten in der OSCI-Infrastruktur zum nächsten gesendet. Dabei wird die Auftragsebene vom jeweils weitersendenden Knoten signiert und für den nächsten Empfänger der Nachricht verschlüsselt. Sie enthält Nutzdaten, die benötigt werden, die Nachricht an den Empfänger weiterzuleiten. Dazu verfügt auch der Intermediär über ein Schlüsselpaar.

Man spricht hierbei auch von einem *„Doppelten Umschlag Prinzip“* (siehe Abb. 5.2).

Bei der Verschlüsselung der Inhalts- und Nutzdaten wird ein Hybridkonzept eingesetzt. Verwendet wird ein effizientes, symmetrisches 2-Key Tripple DES Verfahren. Dabei wird lediglich der Sitzungsschlüssel per RSA verschlüsselt, der Inhalt mit dem symmetrischen Verfahren. Der für die Signatur benötigte Hashwert wird mit dem SHA-1 Algorithmus ermittelt und ist 20 Byte lang.

Die äußerste Schicht des Nachrichtenformats schließlich ist unverschlüsselt. Sie enthält das Chiffrierzertifikat des Empfängers und einen Verschlüsselungskopf, der alle für den Empfänger nötigen Informationen enthält, die Nachricht zu entschlüsseln und die Signatur zu überprüfen.

Die Verarbeitung der Nachrichten durch den Intermediär wird durch einen Laufzettel gesteuert. Der Client erstellt diesen und füllt ihn mit Informationen über Sender und Empfänger, Betreff und weiteren Informationen. Außerdem sind Regeln enthalten, wie die Verteilung der Nachricht durch den Intermediär zu erfolgen hat. Bei der Bearbeitung der Nachricht ergänzt der Intermediär den Laufzettel. Auch der Laufzettel wird signiert, um seine Integrität zu gewährleisten.

Für Sender und Empfänger werden Quittungen ausgestellt, um den erfolgreichen Versand und die abgeschlossene Übertragung an den Empfänger zu bestätigen.

OSCI unterstützt auch die kontinuierliche Kommunikation, bei der mehr als eine Nachricht zwischen Bürger und Verwaltung ausgetauscht wird. Eine Dialog-ID ordnet Nachrichten dieser Kommunikationsreihe zu.

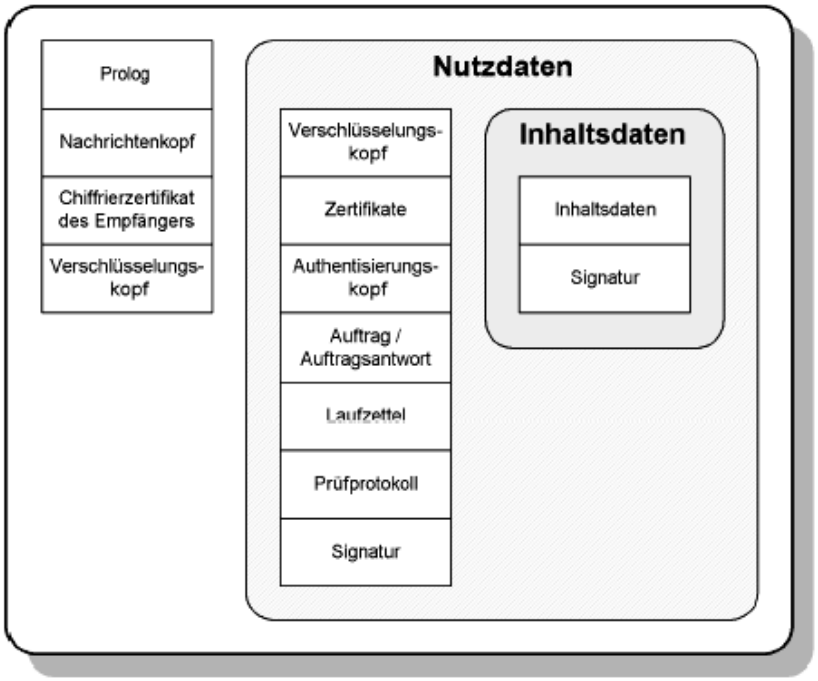


Abbildung 5.2: Der Aufbau einer OSCI Nachricht. (Entnommen aus [19])

Als Format der ausgetauschten Nachrichteninhalte ist XML vorgesehen, um eine möglichst medienbruchfreie Kommunikation zuzulassen. Beliebige Anhänge können (base64 encoded) angefügt werden.

Um auch die Verteilung der Software unter den Bürgern abzusichern, ist vorgesehen, die benötigten Komponenten als signierte Java Anwendungen zu übertragen. Dabei ist ausgeschlossen, dass den Bürgern fremde Software untergeschoben wird, ohne dass diese den Betrug merken.

Für einen sicheren Betrieb eines OSCI Nachrichtensystems sieht OSCI auch ein Betriebskonzept vor, dass Richtlinien bezüglich der Intranets in den Verwaltungen, Firewalls usw. vorsieht.

**Maßnahmen für ein e.P.b.:** Für den direkten Einsatz in einem e.P.b. kommt OSCI-Transport nicht in Frage, denn Kommunikation mit OSCI-Transport geschieht asynchron. Aus diesem Grund ist ein Einsatz von OSCI-Transport zwischen den verteilten e.P.b.-Archivservern nicht sinnvoll möglich. Ein Standesbeamter, der eine Suchanfrage an einen fernen e.P.b.-Archivserver sendet, erwartet sofort eine Antwort und nicht erst nach Minuten.

Jedoch ist der Einsatz von OSCI zum sicheren Mailaustausch innerhalb des Amtes und Ämter übergreifend sinnvoll. Das konzeptionierte e.P.b. wird in diesem Zusammenhang so konfiguriert werden können, dass bei Suchanfragen von fernen Standesämtern lediglich der eindeutige Schlüssel eines Eintrags geliefert wird. Über eine sichere Plattform, wie OSCI, kann dann der Eintrag direkt beim zuständigen Standesbeamten angefordert werden.

Der Einsatz der OSCI-Bibliothek im e.P.b.-System ist darüber hinaus nicht möglich. Die Bibliothek bietet Methoden zum Erzeugen, Versenden und Empfangen von OSCI-Transport-Nachrichten. Die dazu nötigen Kryptomethoden werden jedoch von externen Java Cryptographic Architecture (JCA) Providern zur Verfügung gestellt.

Der OSCI-Client *Governikus* setzt auf den Kryptoprovider des *Institute for Applied Information Processing and Communication* (IAIK[1]) der Technischen Universität Graz auf. Dieser kommerzielle Kryptoprovider stellt die Funktionalität für kryptografische Methoden und PKI in Java zur Verfügung und kommen somit, neben weiteren Produkten (siehe weiter unten), auch für den Einsatz im e.P.b. in Frage.

## 5.3 SAGA und der KoopA-ADV Handlungsleitfaden

Im Umfeld des eGovernment existieren viele Dokumente, die versuchen interoperable Systeme, durch den Einsatz verschiedener Standards in der Verwaltung, zu erreichen. Zwei der wichtigsten werden hier kurz vorgestellt.

SAGA - *(Standards und Architekturen für eGovernment Anwendungen)*[11] wird herausgegeben vom KBSt[2] im Bundesministerium des Inneren und dem BSI. In diesem Dokument werden verbreitete eGovernment Standards vorgestellt und ihre Einsatzfähigkeit in eGovernment-Projekten bewertet. Für alle Projekte im Bereich der eGovernment-Dienstleistungen des Bundes sind die von SAGA festgelegten Standards verbindlich.

Eingeordnet werden die Standards in die Kategorien *obligatorisch*, *empfohlen* und *unter Beobachtung.* Nur in begründeten Ausnahmefällen darf ein niedrig eingestufter Standard eingesetzt werden, wenn ein höher eingestufter existiert.

Darüber hinaus werden auf der SAGA-Homepage die Standards auch im zeitlichen Verlauf eingeordnet, da in neueren Versionen des SAGA-Dokuments bisher empfohlene Standards herausfallen können. Diese werden in die *Blacklist*, die *Whitelist* oder die *Greylist* eingeordnet. Die Blacklist enthält alle bisher abgelehnten Standards, die Whitelist Standards, die noch zu beurteilen sind und die Greylist die Standards, die Bestandsschutz genießen und in früheren SAGA-Dokumenten empfohlen wurden.

Für den Entwurf von eGovernment-Software wird auch ein gemeinsames Entwicklungsmodell vorgesehen. Nach *RM-ODP* (Referenzmodell für offene, verteilte Datenverarbeitung) werden *Viewpoints* definiert, anhand derer die Entwicklung eines eGovernment-System ausgerichtet sein soll. Noch einen Schritt weiter geht SAGA mit der Definition einer Referenzarchitektur für eGovernment-Systeme. Hier wird ein Vierschichtmodell, implementiert in Java (oder J2EE), als obligatorisch vorgesehen.

Ein weiteres Kapitel im SAGA-Dokument befasst sich mit einer vorgeschriebenen Sicherheitsinfrastruktur. Diese reicht von Gebäudesicherung über Zugangskontrollen bis zu Empfehlungen für Firewall und VPN.

Der *KoopA-ADV Handlungsleitfaden für die Einführung der elektronischen Signatur und Verschlüsselung in der Verwaltung[8]* soll interoperable Lösungen für Signatur- und Verschlüsselungssysteme bei Verwaltung, Wirtschaft und Bürgern fördern.

[1] `http://jce.iaik.tugraz.at/` [Stand:19.04.2004]

[2] Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung

Dem Kooperationsausschuss ADV gehören der Bund, die Länder und die kommunalen Spitzenverbände an. Er stellt das Gremium zur Besprechung von IT-Lösungen in der öffentlichen Verwaltung dar.

Dazu gibt er eine Anforderungsanalyse für die kryptografische Absicherung von eGovernment-Lösungen und nennt bereits implementierte Beispiele und/oder Spezifikationen, die für bestimmte Problemstellungen herangezogen werden können.

Besonderes Augenmerk wird auf die gesicherte eMail- und Dokumentenübertragung gelegt. Sind benötigte Szenarien nicht in diesem Handlungsleitfaden aufgelistet, soll der Bedarf dem BSI mitgeteilt werden.

Leider wird nicht auf Langzeitarchivierungen eingegangen. Dazu heißt es, existieren noch keine Pilotprojekte (Stand Dezember 2002).

Empfohlen wird eine verbreitete Kryptofunktionsbibliothek einzusetzen, um die Austauschbarkeit von Dokumenten und die Validierbarkeit der Signaturen durch Dritte zu erleichtern. Außerdem müssen die Algorithmen konfigurierbar sein. Dabei darf die Signaturkomponente applikationsspezifisch, die Verschlüsselungskomponente muss jedoch applikationsunabhängig implementiert sein. Clientprogramme sollten zur besseren Wartbarkeit fernadministrierbar sein.

**Maßnahmen für ein e.P.b.:** Die in *SAGA* vorgeschrieben Standards sind für ein e.P.b. nicht obligatorisch, da dieses kein Bundesprojekt ist. Jedoch ist die Anwendung der genannten Standards sinnvoll und erhöht die Sicherheit des Systems.

Dort wird unter anderem die Anwendung des IT-Grundschutzhandbuchs des BSI vorgeschrieben. Die Einrichtung einer sicheren Netzinfrastruktur ist zwar nicht Teil dieser Arbeit, jedoch ist es empfehlenswert die Konfiguration des Netzwerkes nach den im IT-Grundschutzhandbuch festgelegten Sicherheitskonzepten zu richten. Neben den Sicherheitsmaßnahmen der e.P.b.-Software ist eine sichere Netzwerkumgebung ausschlaggebend für die Gesamtsicherheit.

Zugriffe auf das interne Netz von außerhalb müssen wirksam unterbunden werden. Hier können die Standards in SAGA zur Datensicherheit (*Kapitel 9*) als Maßstab herangezogen werden.

Der *KoopA-Handlungsleitfaden* enthält kein passendes Szenario für ein Langzeitarchiv signierter Dokumente. Deshalb kann hier keine Lösung entnommen werden.

## 5.4 PKI-1-Verwaltung

Die PKI-1-Verwaltung wurde ins Leben gerufen, um eine sichere Kommunikationsplattform auf Basis von Verschlüsselungs- und Signaturverfahren zu ermöglichen. Dieser Service wird Bundes- und Landesbehörden, Kommunen sowie öffentlichen Institutionen angeboten. Ziel ist die Erreichung des IT-Grundschutzniveaus[3] bei elektronischem Rechts- und Geschäftsverkehr nach einem Beschluss der Bundesregierung. Das Bundesamt für Sicherheit in der Informationstechnik stellt dabei die Wurzelinstanz.

[3] Das IT-Grundschutzhandbuch wird vom BSI herausgegeben und gibt Empfehlungen für den Aufbau einer sicheren Netzinfrastruktur.

Es kommen dabei jedoch lediglich Zertifikate zum Einsatz, die sich für fortgeschrittene Signaturen eignen (siehe [13]). Diese können eine Unterschrift per Hand nicht ersetzen.

**Maßnahmen für ein e.P.b.:** Wegen der ausschließlichen Eignung für fortgeschrittene Signaturen kommt der Einsatz der PKI-1-Verwaltung als Trustcenter für ein e.P.b. derzeit nicht in Frage. Jedoch sollte die Entwicklung dieser PKI im Auge behalten werden, da sie eines Tages eine Kosten sparende Alternative zu den kommerziellen PKI sein kann. Das e.P.b. ist nicht auf eine bestimmte PKI festgelegt und könnte, wenn qualifizierte Signaturen und Zeitstempel mit der PKI-1-Verwaltung möglich sind, auch damit betrieben werden.

## 5.5 Schutzbedarfsanalyse

Bei einer Schutzbedarfsanalyse wird für alle Teile eines Systems eine Prognose abgegeben, welcher mögliche Schaden bei einer Verletzung der Integrität, Vertraulichkeit oder Verfügbarkeit auftreten könnte. Dabei sollte der schlimmst mögliche Fall (worst case) betrachtet werden, um das Risiko zu bestimmen.

Die möglichen Schäden können nach [8] und [4] in drei Kategorien eingeordnet werden:

**niedrig/mittel:** Es ist lediglich ein geringfügiger, überschaubarer Schaden zu erwarten. Ein möglicher Ansehensverlust des Amts kann als gering angesehen werden.

**hoch:** Es kann zu einem erheblichen Schaden kommen. Ein beträchtlicher Ansehensverlust des Amts ist zu erwarten.

**sehr hoch:** Ein katastrophaler, existenzbedrohender Schaden muss befürchtet werden. Neben einem beträchtlichen Ansehensverlust des Amts kann der Schaden die gesellschaftliche Stellung oder die wirtschaftlichen Verhältnisse der Betroffenen beeinflussen und diese ruinieren.

Sind die verwalteten Daten als **Verschlusssache** (VS) eingestuft, gelten die VS-IT-Richtlinien des Bundes oder der Länder. Hier ist zu überprüfen, ob diese Daten für eine elektronische Verwaltung geeignet sind, bzw. es müssen geeignete Sicherheitsmaßnahmen für das System ergriffen werden.

Je nach dem festgestellten Risiko muss nun im Einzelnen geklärt werden, ob die Kosten für Absicherung gegen das Risiko in einem angemessenen Verhältnis zum erwarteten Schaden stehen. Insbesondere müssen Kryptomechanismen zum Einsatz kommen, die eine ausreichende Absicherung bieten.

Die Schutzbedarfsanalyse sollte sich jedoch nicht nur über das System an sich ausstrecken, sondern auch die Infrastruktur mit einschließen. Hierzu wird die Anwendung des IT-Grundschutzhandbuchs empfohlen.

**Maßnahmen für ein e.P.b.:** Die Sensibilität von Personenstandseinträgen kann als *hoch* bzw. *sehr hoch* angesehen werden. Im geplanten e.P.b. wird dieser Tatsache dadurch Rechnung getragen, dass eine Übertragung vom Server zum Client, sowie zwischen verschiedenen Servern, verschlüsselt abläuft. Außerdem findet eine Benutzerauthentifizierung mit qualifizierten Zertifikaten statt, die eine hohe Sicherheit bieten.

Diese Maßnahmen alleine genügen jedoch nicht, um die Daten ausreichend vor Missbrauch zu schützen. Die Netzwerkinfrastruktur muss zusätzlich so abgeschottet sein, dass sich keine Hintertüren für Angreifer bieten. Dies zu erreichen liegt außerhalb dieses Konzeptes für ein e.P.b.. Hier müssen die Standesämter geeignete Vorkehrungen treffen und eigene Schutzbedarfsanalysen erstellen.

Eine Möglichkeit, große Sicherheit für die Übertragung herzustellen, sei hier aber dennoch genannt. Der Bund betreibt ein sicheres Netz für die Kommunikation von Behörden (*IVBB*, *TESTA*). Diese Netze bieten ständige Verschlüsselung der Kommunikation. Ein e.P.b. könnte bei einer entsprechenden Entscheidung auch an diese Netze angeschlossen werden.

## 5.6 Die Landes-Policy von Hessen

Bei einem Termin in der *Hessischen Zentrale für Datenverarbeitung* (HZD)[4] in Hünfeld wies der Projektleiter Justizverfahren, Herr Klüber, auf einen weiteren Punkt hin, der bei der Entwicklung eines e.P.b. zu beachten ist: Es existiert eine Landes-Policy für die IT-Systeme in Hessen, die unbedingt für öffentliche Einrichtungen einzuhalten ist.

In Bezug auf das geplante e.P.b.-System ist zu beachten, dass es untersagt ist, interne Rechner über das Internet direkt zu kontaktieren. Herr Klüber gab an, dass deshalb interne Rechner im HZD, auf die der Zugriff von außen möglich sein soll, nur über Protokollumsetzer erreichbar sind.

**Maßnahmen für ein e.P.b.:** Für das e.P.b. ergeben sich verschiedene Möglichkeiten mit dieser Situation umzugehen. Betroffen ist dabei lediglich die Kommunikation zwischen e.P.b.-Archivservern an verschiedenen Standorten.

Die Lösung mit Protokollumsetzern ist auch bei dem e.P.b. vorstellbar. Diese Geräte wandeln ein Protokoll in ein anderes um. In einem solchen Aufbau kommunizieren die e.P.b.-Archivserver mit den Protokollumsetzern. Diese leiten die Kommunikation an den entfernten Protokollumsetzer weiter, der mit dem dortigen e.P.b.-Archivserver kommuniziert. Dabei werden unterschiedliche Protokolle zwischen e.P.b.-Archivserver und Protokollumsetzer, sowie zwischen den Protokollumsetzern, verwendet.

Eine weitere Möglichkeit ist, lediglich die Suchinformationen, die von außen zugreifbar sein sollen, in einer demilitarisierten Zone abzulegen. Eine demilitarisierte Zone ist ein geschützter Teil des Netzwerks, in dem von außen zugreifbare Dienste untergebracht sind. Nun können entfernte e.P.b.-Archivserver auf diese Suchinformationen zugreifen. Hierbei ist zu prüfen, ob diese Daten auf einem Server liegen dürfen, der über das Internet zugreifbar ist.

[4] Die HZD ist für den Betrieb des elektronischen Grundbuchs verantwortlich.

Direkte Auswirkungen auf die in dieser Arbeit entworfene Sicherheitskomponente wird eine solche Entscheidung nicht haben. Beide vorgestellten Möglichkeiten werden damit zu realisieren sein.

# 6. Das Konzept für ein e.P.b.

Die bis jetzt vorgestellten Fakten werden nun in einem Konzept zusammengebracht, welches eine Signatur- und Sicherheitslösung für das spätere e.P.b. darstellen soll. Dabei geht es lediglich um Fragestellungen, die durch Maßnahmen in der Software eines e.P.b. zu erreichen sind. Für den sicheren Betrieb eines e.P.b. werden darüber hinaus auch Fragen der Netzwerksicherheit zu klären sein. Das ist jedoch nicht Teil dieser Arbeit. Trotzdem wird zum Ende dieses Kapitels auf mögliche Backupstrategien eingegangen, die sich für das geplante e.P.b. besonders gut eignen.

Für die zu entwickelnde Sicherheits- und Signaturkomponente sind folgende Themen zu bearbeiten:

- Es wird ein Konzept benötigt, welches die Rechtskräftigkeit der signierten Dokumente über Jahrzehnte erhält.
- Um Missbrauch auszuschließen, muss ein Loginkonzept entworfen werden, um die Benutzer gegenüber dem e.P.b.-Archivserver zu authentifizieren.
- Die Kommunikation zwischen den e.P.b.-Archivservern und den Clients muss in einer sicheren Art und Weise erfolgen. Das Konzept muss sicherstellen, dass die Kommunikation nicht abgehört oder verfälscht werden kann.

Die folgenden Abschnitte beschreiben zunächst die Konzepte, die die genannten Anforderungen für das e.P.b. umsetzen. Eine exakte Beschreibung der Verfahren und Ansätze deren Implementierung befindet sich in einem späteren Kapitel.

## 6.1 Langfristige Signatur

Die signierten Einträge, die in einem e.P.b. abgelegt werden, müssen ihre Rechtsgültigkeit über viele Jahrzehnte bewahren. Dies stellt verbreitete Signaturverfahren vor ein Problem. Das liegt daran, dass jeder Benutzer ein Schlüsselpaar und ein Zertifikat vom Trustcenter erhält, in dem das Trustcenter die Identität des Benutzers bestätigt. Diese Zertifikate sind in der SigV aus Sicherheitsgründen auf einen Gültigkeitszeitraum von *maximal* fünf Jahren beschränkt.

Diese Einschränkung soll zum einen eine regelmäßige Erneuerung der personenbezogenen Daten im Zertifikat sicherstellen und zum Anderen einen regelmäßigen Wechsel der Schlüssel erzwingen. Dabei wird davon ausgegangen, dass selbst mit den schnellsten Computern der Schlüssel nicht innerhalb von fünf Jahren geknackt werden kann. Regelmäßig werden neue Schlüssellängen und neue Hash- und Signaturverfahren zum Standard erhoben, die jeweils wieder eine minimale Sicherheit von sechs Jahren bieten müssen.

Wird nun ein Zertifikat ungültig, entweder durch den Ablauf des Gültigkeitszeitraums oder durch eine Sperrung, werden alle damit erstellten Signaturen ungültig. Die gebräuchlichste Möglichkeit einer Signatur zu einer längeren Lebensdauer zu verhelfen ist, die Signatur zusätzlich mit einem Zeitstempel des Trustcenters zu versehen.

Dabei wird die Signatur an das Trustcenter gesendet. Dieses signiert den Hashwert mitsamt dem aktuellen Datum und der Uhrzeit. Hängt man diese zusätzlichen Daten der signierten Nachricht an, beweisen sie, dass die Originalsignatur *vor* der Anforderung dieses Zeitstempels erfolgte. Wird nun später ein Dokument angefordert, kann anhand des Zeitstempels überprüft werden, ob das zugrunde liegende Zertifikat zum Zeitpunkt der Signatur noch gültig war.

Nach dem Ablauf werden die Zertifikate bis zu 30 Jahren bei akkreditierten Trustcentern vorgehalten. Zur Überprüfung einer Signatur kann das Zertifikat, in dieser Zeit, bei dem Trustcenter angefordert werden. Diese Vorgehensweise genügt jedoch nicht für ein Langzeitarchiv eines e.P.b. aus zwei Gründen:

Das Zertifikat des Trustcenters läuft ebenfalls irgendwann aus. Schöpft man die maximale Laufzeit von fünf Jahren für das eigene Zertifikat aus und hängt dann erst an eine Signatur den Zeitstempel des Trustcenters, erreicht man eine maximale Gültigkeit von zehn Jahren (dann läuft spätestens das für den Zeitstempel verwendete Trustcenterzertifikat aus). Das ist zu wenig für Personenstandseinträge.

Die zweite Bedrohung für eine langfristige Signatur liegt darin, dass die Hashfunktion unsicher wird. Gelingt es einem Angreifer ein Dokument zu finden, welche den gleichen Hashwert, wie das signierte Dokument aufweist, dann kann er der Signatur das geänderte Dokument unterschieben. Angriffsmethoden auf Hashfunktionen sind bekannt. Die geläufigste ist der *Geburtstagsangriff*[1].

Es müssen Vorkehrungen getroffen werden, die diese Hindernisse für eine langfristige Gültigkeit der Signaturen überwinden.

### 6.1.1 Bestehende Lösungen

In dem Bereich der Langzeiterhaltung von Signaturen gibt es bisher nur wenige Konzepte. Zum Zeitpunkt der Recherche für diese Arbeit waren drei Konzepte verbreitet, die alle durch den Einsatz von Zeitstempeln eine längere Gültigkeit der Signaturen erreichen. Die drei Konzepte werden hier in aller Kürze vorgestellt.

[1] Die Geburtstagsangriffsmethode hat ihren Namen von einem alten statistischen Trick. Jemand wettet, dass sich in einer (relativ) kleinen Gruppe Menschen zwei finden lassen, die am gleichen Tag Geburtstag haben.

Man könnte nun annehmen, dass es bei 365 Tagen schon einer größeren Menschenmenge benötigt, um die Wette zu gewinnen. Dem ist jedoch nicht so. Schon bei 23 Menschen ist die Wahrscheinlichkeit 50%, dass sich zwei mit demselben Geburtstag finden lassen. (ohne Beweis)

Bei einem Angriff auf eine Hashfunktion lässt sich diese statistische Eigenschaft ausnutzen:
Der Angreifer besitzt ein signiertes Dokument, dessen Inhalt er nachträglich ändern möchte. Dazu ändert er das Dokument nach seinen Wünschen ab und definiert Platzhalter in diesem Dokument, die mit verschiedenen Wörtern (oder auch nur Leerzeichen) gefüllt werden können, ohne das Dokument in seinem gewünschten Sinn zu verändern.

Nun lässt er ein Programm darüber laufen, welches alle Permutationen des Dokuments ausprobiert, die mit den verschiedenen Inhalten für die Platzhalter möglich sind. Dadurch lassen sich leicht mehrere Millionen und Milliarden Variationen des Dokuments erzeugen. Jedes Dokument erzeugt einen anderen Hashwert.

Durch die schiere Masse ist es damit möglich ein Dokument zu finden, welches den gleichen Hashwert, wie das Originaldokument aufweist. Der Angreifer kann dann dieses Dokument der Signatur unterschieben.

Die im Moment empfohlenen Hashfunktionen, wie SHA-1, erzeugen 160 Bit lange Hashwerte. Dadurch sind so viele Permutationen nötig, dass sie in überschaubarer Zeit nicht durchprobiert werden können. Deshalb gelten diese Hashfunktionen noch als sicher.

**ETSI** (*European Telecommunicatons Standards Institute*) definiert einheitliche technische Standards für einen EU-übergreifenden Binnenmarkt. In dem Dokument „Electronic Signature Formats“[16] wird neben Datenformaten für Signaturen auch ein Konzept für die Langzeiterhaltung der Signaturen präsentiert.

Dort ist vorgesehen, dass möglichst kurz nach der ersten Signatur eines Dokuments, ein Hashwert über die Signatur gebildet und von diesem ein Zeitstempel angefordert wird. Die Informationen zur Verifizierung der Signatur (Zertifikate, CRL oder OCSP-Antworten) müssen zusätzlich aufbewahrt werden.

Der Schwachpunkt dieses Konzeptes für den Einsatz in einem e.P.b. ist, dass der Zeitstempel lediglich über die Signatur gebildet wird. Dies schützt nicht davor, dass die eingesetzte Hashfunktion unsicher werden könnte. Wird die Hashfunktion unsicher, dann kann ein Angreifer weitere Dokumente erzeugen, welche der Signatur entsprechen. Der Hashwert muss dabei nur dem Hashwert in der Signatur entsprechen, wenn die angefügten Zeitstempel lediglich über die Signatur gebildet wurden.

**SigI** enthält die *Signatur-Interoperabilitäts-Spezifikationen* des BSI[12], nach den Vorgaben von SigG/SigV, an eine PKI. Diese Spezifikationen sollen dafür sorgen, dass die verschiedenen Implementierungen von PKI-Systemen über kompatible Schnittstellen verfügen, was den Erfolg der PKI vorantreiben wird.

Abschnitt B5 *„erneute digitale Signatur“* zeigt ein Konzept, welches die Bedrohungen für eine langfristige Signatur anspricht und eine Lösung bietet. Laut SigI ist allein die erneute Signatur dazu fähig, den Wert digitaler Unterschriften über den Ablauf der Eignung von Hash- und Signaturverfahren hinaus zu sichern. Das Dokument definiert das Anfügen von Zeitstempeln als erneute Signatur, da diese mit einer Signatur versehen sind.

Vorgesehen wird, das signierte Dokument, möglichst bald nach der Einspeicherung, mit einem Zeitstempel des Trustcenters zu versehen. Im Unterschied zu dem Konzept des ETSI wird dabei das gesamte Dokument inklusive der Originalsignatur in den Zeitstempel mit einbezogen. Dies schützt das Dokument vor dem Ende der Eignung des Hashverfahrens der Originalsignatur.

Versucht später ein Angreifer der Originalsignatur ein neues Dokument unterzuschieben, muss dessen Hashwert nicht nur zu dem der Originalsignatur, sondern auch zum Hashwert für den Zeitstempel passen. Somit besteht ein doppelter Schutz vor Missbrauch der Signatur.

Um eine langfristige Gültigkeit der Signatur zu erhalten, ist es ab dann nötig, das Dokument, mitsamt aller vorigen Signaturen und Zeitstempel, regelmäßig mit einem weiteren Zeitstempel des Trustcenters zu versehen. Damit wird jeweils festgehalten, dass der vorige Zeitstempel *vor* dem Datum im neuen Zeitstempel erstellt wurde. Auf diese Weise lässt sich später beweisen, dass die Zeitstempel zu einer Zeit erstellt wurden, als das damalige Zertifikat des Trustcenters noch gültig und die eingesetzte Hashfunktion noch sicher war. Es wird natürlich vorausgesetzt, dass für die Erstellung der Zeitstempel jeweils aktuelle Hashfunktionen eingesetzt werden.

Um später den Beweis führen zu können, dass die Zertifikate der Standesbeamten gültig waren, muss man deren Gültigkeit zum Zeitpunkt der Unterschrift belegen können. Dies wird ermöglicht durch die regelmäßige Archivierung aktueller Zertifikate und Sperrlisten oder OCSP-Abfragen. Diese Validierungsdaten müssen selbstverständlich ebenso wie die Dokumente regelmäßig mit einem neuen Zeitstempel des Trustcenters versehen werden.

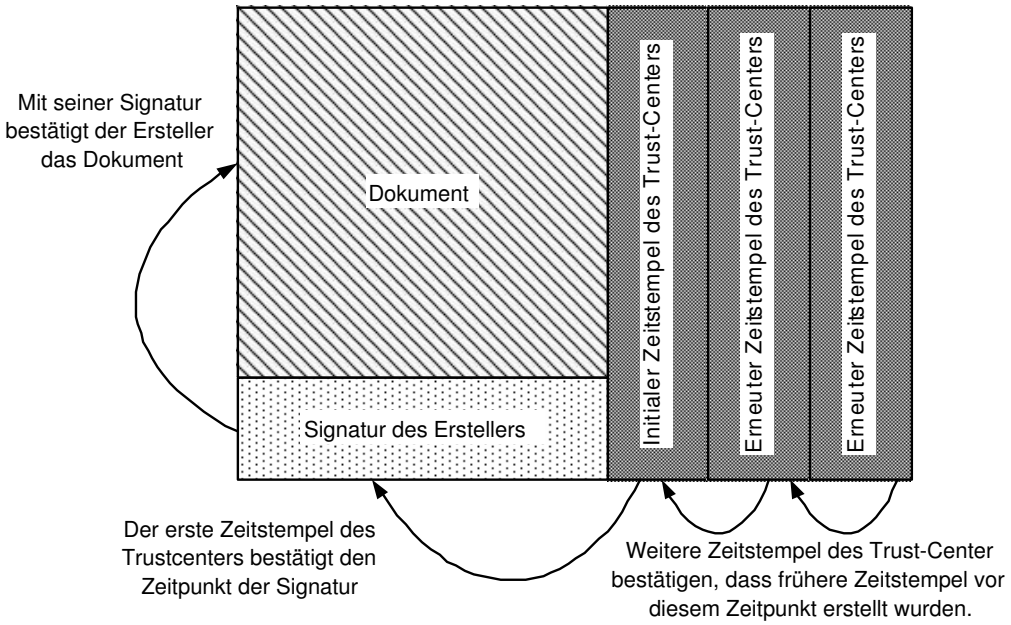


Abbildung 6.1: Zur Beweiserhaltung der Signatur sehen die SigI eine Kette von aufeinander aufbauenden Zeitstempeln vor.

Die Belege zu erbringen, dass die Zertifikate des Trustcenters zu dem betreffenden Zeitpunkt gültig waren, gehört zu den Aufgaben des Trustcenters.

Um die Anzahl der benötigten Zeitstempel für die erneute Signatur zu verringern, sehen die SigI-Spezifikationen auch das Signieren eines ganzen Archivs mit einem gemeinsamen Zeitstempel vor. Dabei können Dokumente dieses Archivs nachträglich nicht mehr gelöscht werden. Fehlt auch nur ein einzelnes, dann kann der Hashwerte über die Dokumente später nicht mehr berechnet werden und der Zeitstempel wird ungültig.

**ArchiSig** Das Konzept von ArchiSig setzt auf das Konzept von SigI auf und bietet eine Lösung, die es erlaubt, einzelne Dokumente nachträglich aus einem Archiv zu löschen, ohne die Gesamtintegrität des Archivs zu zerstören. Dieser Abschnitt referiert die Beschreibung des ArchiSig-Konzeptes in [10].

Das Projekt *ArchiSig - Beweiskräftige und sichere Langzeitarchivierung digital signierter Dokumente* wurde vom Bundesministerium für Wirtschaft und Technologie im Rahmen des Wettbewerbs *VERNET* gefördert. Ziel des Projektes war es ein Konzept für erneute Signaturen zu entwerfen und in der Praxis zu testen. Damit sollten Unternehmen zum Einsatz solcher Konzepte ermutigt werden. Bisher waren viele nicht zu einer Langzeitarchivierung digital signierter Dokumente bereit, da es auf diesem Gebiet bisher kaum funktionierende Lösungen gab.

Das Ergebnis wurde im Herbst 2003 in einem Praxistest validiert. Im Universitätsklinikum Heidelberg implementierte man das Konzept für die elektronische Patientenakte. Abschließend wurde anhand von simulierten Gerichtsverfahren die Beweisfähigkeit der darin abgelegten Akten geprüft. Das Konzept hat sich bewährt und das Projekt gilt nun als abgeschlossen.

Der ArchiSig-Ansatz baut auf der Datenstruktur *hashtree* auf. Direkt nach der Einspeicherung wird ein Zeitstempel für das Dokument erzeugt, um den Umgang mit ihm zu vereinfachen. Dann werden, in einem zentralen Archiv, gemeinsame Hashwerte für alle Zeitstempel der Dokumente mit der hashtree-Struktur gebildet. Für den Hashwert an der Wurzel wird ein Archivzeitstempel erstellt, der alle Dokumente gemeinsam signiert.

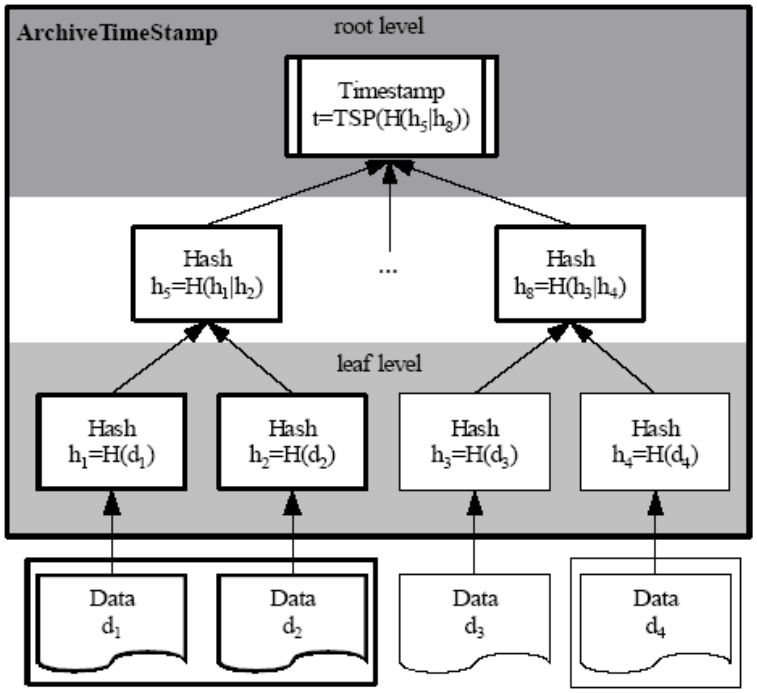


Abbildung 6.2: Der Archiv-Zeitstempel des ArchiSig-Konzeptes basiert auf einer hashtree-Struktur. (Entnommen aus [10])

Die *hashtree-Struktur* wird folgendermaßen gebildet (siehe Abb. 6.2):

- Die Blätter des Baumes bilden die Hashwerte der einzelnen Dokumente.
- Auf den höheren Ebenen wird jeweils ein Hashwert über zwei Hashwerte der darunter liegenden Ebene erstellt.
- Der Wurzel-Hashwert kann auch aus mehr als zwei Hashwerten der darunter liegenden Ebene gebildet werden.

Wird ein Dokument angefordert, so werden zusätzlich zum Originaldokument, alle für die Zurückverfolgung des hashtree-Weges benötigten Hashwerte mitgeliefert, sodass die Überprüfung der Signatur möglich ist.

Wird ein *erneuter Zeitstempel* für das Archiv fällig, so sind zwei Möglichkeiten vorgesehen:

1. Ein einfacher neuer Archivzeitstempel.

- Dies ist geht nur dann, wenn die Hashfunktion, die im Baum verwendet wurde, nicht unsicher wird. Denn nur der Hashwert wird neu signiert.

2. Den gesamten hashtree neu aufbauen.

   - Dabei wird auch das Originaldokument in die Bildung der Hashwerte mit einbezogen. Diese Vorgehensweise wird vor allem benötigt um bestimmte, in [10] nicht näher spezifizierte, Eigenschaften des Baumes zu erhalten.

Zertifikate und Sperrinformationen werden zusätzlich aufbewahrt.

Der Vorteil dieser Konstruktion ist, dass man einzelne Dokumente nachträglich entfernen kann. Dabei wird nur deren Hashwert aufbewahrt. Durch Abschreiten der Baumstruktur kann dann der Hashwert, der für die Berechnung des Archivzeitstempels verwendet wurde, weiterhin bestimmt werden.

### 6.1.2 Langzeit-Signaturerhaltung für ein e.P.b.

Das im e.P.b. eingesetzte Verfahren setzt auf den SigI-Spezifikationen auf. Der Nachteil, dass sich nachträglich keine Dokumente mehr entfernen lassen, gilt im Personenstandswesen nicht. Dort müssen einmal abgelegte Einträge nicht aus Datenschutzgründen nachträglich entfernt werden können. Damit wird die aufwendige ArchiSig hashtree-Struktur nicht benötigt. Das ETSI-Konzept kommt nicht in Frage, da dies keinen ausreichenden Schutz davor bietet, dass die Hashfunktion unsicher werden könnte.

Soll ein signierter Eintrag in das e.P.b. abgelegt werden, prüft das System zunächst die Gültigkeit der Signatur. Dabei muss nach der mathematischen Prüfung auch die Gültigkeit des Zertifikats ermittelt werden. Dazu wird der Zertifizierungspfad bis zu einem bekannten, vertrauenswürdigen Trustcenter zurückverfolgt. Dann wird der Status aller Zertifikate des Pfades per OCSP-Abfragen überprüft. Sind alle Zertifikate gültig, so akzeptiert das e.P.b. die Signatur. Alle diese Zertifikate werden aufbewahrt.

Sobald, als möglich werden die Einträge dann mit einem initialen Zeitstempel des Trustcenters versehen. Grundlage für den Zeitstempel ist ein Hashwert, der über den Eintrag und seine Signatur gebildet wird. Somit ist die Signatur auch vor späteren Angriffen auf die eingesetzte Hashfunktion geschützt. Der Zeistempel beweist, dass sowohl Eintrag, als auch Signatur zur gleichen Zeit erstellt wurden. Er verbindet also die Signatur mit dem Eintrag und macht ein nachträgliches Unterschieben veränderter Einträge unmöglich.

Ist ein Eintrag einmal mit einem Zeitstempel versehen, dann überprüft das e.P.b. beim späteren Umgang mit diesem Eintrag nur noch den Zeitstempel. Innerhalb des Systems wird die Gültigkeit der Originalsignatur dann als bewiesen angesehen.

Direkt nach dem initialen Zeitstempel werden OCSP-Abfragen für alle Zertifikate des Pfades gestellt. Damit kann später die Gültigkeit des verwendeten Zertifikats belegt werden. Da diese OCSP-Antworten nach dem initialen Zeitstempel angefordert wurden, kann später bewiesen werden, dass die Signatur vor dem Zeitpunkt der OCSP-Abfrage erfolgt ist. Der darin vermerkte Status der Zertifikate galt damit bewiesenermaßen bei Erstellung der Signatur.

Um die Beweisfähigkeit zu erhalten, werden in Intervallen neue Zeitstempel für alle Einträge eingeholt. Bevor ein neuer Zeitstempel hinzugefügt wird, prüft das e.P.b.-System zuerst, ob das Dokument seit dem letzten Zeitstempel unverändert ist. Dazu wird der letzte Zeitstempel mathematisch geprüft und die Gültigkeit des zugrunde liegenden Trustcenterzertifikats sichergestellt.

Durch diese Prüfung wird sichergestellt, dass sich keine Lücken bei der Kette der Zeitstempel ergeben, die den Beweis der Gültigkeit einer Signatur unmöglich machen würden. Die angehängten Zeitstempel schließen dabei den signierten Eintrag, die OCSP-Antworten und alle vorigen Zeitstempel mit ein.

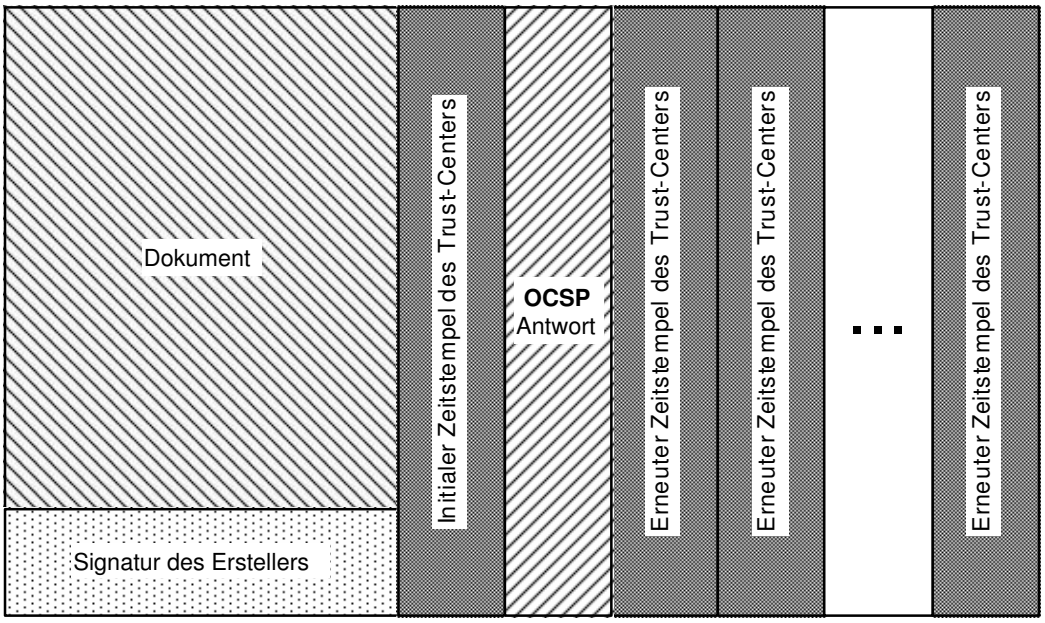


Abbildung 6.3: Das e.P.b.-Konzept zur Langzeiterhaltung der Beweisfähigkeit von Einträgen.

Ein kostenintensiver Punkt in einem solchen Signatursystem sind, bei den derzeitigen Preismodellen der Trustcenter, die Zeitstempel. Neben einem festen Betrag für das Ausstellen und Betreuen von Zertifikaten, verlangen die Trustcenterbetreiber einen Betrag pro ausgestelltem Zeitstempel. Durch die beweiserhaltenden Maßnahmen fallen bei einem e.P.b. eine große Anzahl an Zeitstempeln an.

Dies führte zu einer ersten Variante, welche die Anzahl der Zeitstempel reduzieren sollte, um den Betrieb möglichst Kosten sparend zu ermöglichen. Im Laufe der Arbeit stellte sich diese erste Variante jedoch als nicht praktikabel heraus und wurde durch eine neue Variante ersetzt. Mehr zu den Gründen nach der Vorstellung beider Varianten:

Um zu verhindern, dass die Anzahl der Zeitstempel im Laufe der Zeit überhand nimmt, wurde in der *ersten Variante* vorgesehen, mehrere Einträge zu einem Archiv zusammenzufassen. Bei den folgenden Zeitstempeln wird dann nur noch einer pro Archiv benötigt. Im Bereich des Standesamtes lässt sich dieses Konzept dazu benutzen, z.B. zum Jahresabschluss, alle neuen Einträge zu einem Archiv zusammenzufassen. Bei größeren Ämtern ist hier auch ein kürzerer Zeitraum (1/2 oder 1/4 Jahr) wählbar.

Ein solches Archiv kann auf Kosten sparende WORM-Medien (Write Once - Read Many) ausgelagert werden. Jeder neue Archivzeitstempel wird dann nicht mit auf dem Backupmedium abgelegt, sondern in einem gesonderten Bereich im e.P.b.-Archivserver. Das hat den Vorteil, dass die erstellten Backupmedien nicht mehr verändert werden müssen.

Zusätzlich zu den Zeitstempeln und Validierungsdaten fallen weitere Daten an, die im e.P.b.-Archiv abgelegt werden müssen. Dabei geht es um Verwaltungsdaten zu den einzelnen Einträgen.

Verwaltungsdaten fallen bei der Fortführung eines Eintrags an. Der alte Eintrag kann dabei nicht gelöscht werden. Seine Signatur muss weiterhin überprüfbar sein. Jedoch muss im Archiv vermerkt werden, dass eine neue Version existiert. Dazu wird in den Verwaltungsinformationen des alten Eintrags dieser Umstand vermerkt. In den Verwaltungsinformationen des neuen Eintrags wird ein Verweis auf den alten abgelegt.

Befindet sich ein Eintrag in einem Archiv, dann werden seine Verwaltungsinformationen zum Archiv abgelegt. Bei späteren Zeitstempeln werden sämtliche vorigen Verwaltungsinformationen mit einbezogen, um auch ihre Integrität zu schützen.

Der fortgeführte Eintrag wird neu im Archiv abgelegt. Die Clientsoftware legt dabei die Fortführung als neue Schicht über den bisherigen Eintrag und legt das komplette neue Dokument im e.P.b. ab. Die alte Version bleibt erhalten.

Der Beamte, der die Fortführung erstellt, signiert die Fortführung und die Zugehörigkeit zum alten Eintrag. Dazu unterschreibt er das komplette PDF-Dokument, nicht nur seine Fortführung.

Bei Anforderung eines Eintrags, prüft der e.P.b.-Archivserver zunächst die Integrität anhand des letzten Zeitstempels. Ist diese gegeben, so kann angenommen werden, dass die Signatur überprüft wurde und gültig ist. Der Eintrag wird zunächst nur mit Signatur und zugehörigem Zertifikat ausgeliefert.

Soll eine Signatur auf ihre Gültigkeit überprüft werden, so können alle benötigten Validierungsinformationen beim e.P.b.-Archivserver angefordert werden. Dazu gehören die Originalsignatur mit dem zugrunde liegenden Zertifikat und allen Zertifikaten des Zertifikatpfades. Außerdem die OCSP-Antworten und alle Zeitstempel, die bis zu diesem Zeitpunkt hinzugefügt wurden.

Wie bereits erwähnt, erwies sich diese erste Variante als nicht praktisch umsetzbar. Der Grund erschließt sich bei näherer Untersuchung der Datenmengen, die durch die Struktur in Jahresarchiven anfallen. Vor allem bei der Prüfung der Integrität eines Eintrags wird die Problematik ersichtlich. Hierzu ein Beispiel:

In einer großen Stadt fallen im Jahr etwa 20.000 Geburtenbucheinträge an (etwa 1% der Bevölkerung). Diese werden am Jahresende zu einem Archiv zusammengestellt und mit einem einzigen Zeitstempel versehen.

Soll nun die Integrität eines der dort eingespeicherten Einträge überprüft werden, so wird der letzte Zeitstempel (hier also der Zeitstempel des Jahresarchivs) überprüft werden.

Die Überprüfung des Zeitstempels erfolgt dadurch, dass ein Hashwert über alle Einträge dieses Archivs gebildet wird und dieser mit dem signierten Hashwert im Zeitstempel verglichen wird.

Geht man von einer durchschnittlichen Größe von 100 KByte PDF-Datei pro Eintrag aus, so ergibt sich folgende Datenmenge: $20.000 * 100$ KByte $= 2.000.000$ KByte = ca. 1,9 GByte. Allein zur Überprüfung eines Eintrags müssen demnach annähernd zwei Gigabyte Daten gelesen werden.

Geht man von alten Einträgen aus, die als Bild eingespeichert wurden, so kann man (optimistisch geschätzt) mit 1 MByte großen PDF-Dateien rechnen. Dabei ergibt sich eine Datenmenge von ca. 19,5 GByte.

Die Vorteile der ersten Variante wiegen unmöglich den dazu erforderlichen Verwaltungsaufwand auf. Eine tabellarische Zusammenstellung der Vor- und Nachteile der zwei Varianten befindet sich am Ende dieses Abschnitts. Im Kapitel über die Datenstrukturen für das e.P.b. wird auch eine Datenstruktur für die erste Variante vorgestellt, an der sich die Vor- und Nachteile nachvollziehen lassen.

Für die *zweite Variante* werden höhere Kosten für Zeitstempel in Kauf genommen, zugunsten eines effizienten Zugriffs auf die Einträge. Wichtigster Unterschied ist hier, dass es keine (Jahres-) Archive gibt. Die Einträge werden einzeln in der Datenbank abgelegt. Sollen die Einträge mit einem neuen Zeitstempel versehen werden, so wird je ein Zeitstempel pro Eintrag benötigt.

Durch eine veränderte Datenstruktur wird es auch hier ermöglicht, die Einträge auf WORM-Medien auszulagern. Im praktischen Einsatz eines e.P.b. wird das aber bei den heute gängigen Festplattengrößen nicht nötig werden und stellt somit eine Ausnahme dar. Mehr zu dieser Datenstruktur in einem der folgenden Kapitel. Bis auf die bisher genannten Punkte entspricht die zweite Variante der ersten.

Die folgenden Beispiele zeigen die Mehrkosten für die zweite Variante:

**Kosten bei Variante 1:** Ein Jahrgang besteht aus 20.000 Einträgen. Diese werden einem Archiv zugeordnet und alle vier Jahre mit einem neuen Zeitstempel versehen. Die Einträge werden über 100 Jahre aufbewahrt. Dabei wird jeweils immer nur ein neuer Zeitstempel für das gesamte Archiv benötigt.

Es werden also 25 Zeitstempel über den gesamten Zeitraum hinweg benötigt. Geht man von einem Cent pro Zeitstempel aus, sind das 25 Cent. Pro Eintrag fallen also $25$ Cent / $20.000 = 0,00125$ Cent pro Eintrag.

**Kosten bei Variante 2:** Hier muss für jeden Eintrag je ein Zeitstempel erstellt werden. Demnach fallen pro Eintrag 25 Cent über den Zeitraum an.

| | Variante 1 (mit Jahresarchiv) | Variante 2 (ohne Jahresarchiv) |
|---|---|---|
| **Vorteile** | +Kosten sparend durch wenige Zeitstempel<br>+Auslagerung der Einträge durch das automatische Zusammenfassen als Jahrgänge problemlos möglich | +begrenzte Datenmengen müssen bewegt werden<br>+einfachere Datenstruktur |
| **Nachteile** | -Hoher Verwaltungsaufwand durch die komplexe Struktur<br>-praxisferne Datenmengen müssen bewegt werden | -mehr teure Zeitstempel erforderlich |

Aufgrund der überwiegenden Nachteile der ersten Variante wird lediglich die zweite Variante in dieser Arbeit vertieft.

## 6.2 Benutzerauthentifizierung und Login

Der Authentifizierungsmechanismus in einem e.P.b. muss sicherstellen, dass lediglich befugte Personen Zugang zu den Personenstandseinträgen haben. Da es sich hierbei um äußerst sensible Daten handelt, muss das Sicherheitssystem höchsten Anforderungen entsprechen. Das Mittel der Wahl ist dabei auch in diesem Bereich die Verwendung eines qualifizierten Zertifikats. Die Standesbeamten müssen ohnehin mit einem qualifizierten Zertifikat ausgestattet werden, um die Einträge zu signieren. Somit ist die erforderliche Ausstattung vorhanden.

Da e.P.b. und die Standesamtssoftware eng miteinander verzahnt sind, kann auch die Authentifikation gegenüber der Standesamtssoftware und gegenüber dem e.P.b.-System gleichzeitig vollzogen werden. Für den Standesbeamten stellt es sich so dar, dass er seine Smartcard in den Kartenleser einführt und seine PIN eingibt. Damit ist dieser Benutzer für die gesamte Sitzung angemeldet.

### 6.2.1 Authentifikation und Sitzungsticket

Zur Authentifizierung wird ein POP[2] durch ein Challenge Response Verfahren durchgeführt. Dabei verschlüsselt der Server eine Nachricht mit dem öffentlichen Schlüssel des Benutzers aus dessen Zertifikat.

Nur der richtige Benutzer kann diese Nachricht entschlüsseln, da er über den zugehörigen privaten Schlüssel verfügt. Erhält der Server die entschlüsselte Nachricht zurück, so hat der Benutzer den Besitz des privaten Schlüssels bewiesen und gilt, für die gesamte Sitzung, als authentifiziert.

[2] Proof of Possession

Realisiert wird dieser Mechanismus in e.P.b durch ein Ticketsystem. Der e.P.b.-Archivserver authentifiziert den Benutzer und stellt ein Ticket aus. Bei späteren Anfragen des Clients an den Server wird immer dieses Sitzungsticket mitgeliefert. Anhand des Tickets kann der Server die eingehenden Nachrichten den Benutzern zuordnen. Es verfällt, sobald sich ein Benutzer beim System abmeldet oder nach einem Arbeitstag.

Damit ein Sitzungsticket nicht ausgespäht und missbräuchlich verwendet werden kann, sind weitere Sicherheitsmaßnahmen vorgesehen. Diese werden weiter unten im Abschnitt über die *Kommunikationssicherheit* vorgestellt.

Das zur Benutzeranmeldung vorgesehene *Challenge Response Verfahren* mit Ticketaustausch verläuft folgendermaßen:

1. Die Standesamtssoftware meldet dem e.P.b.-Archivserver, dass sich ein Benutzer anmelden will. Dazu sendet sie das zugehörige qualifizierte Benutzerzertifikat an den Server.
2. Nach der Prüfung des Zertifikats per OCSP, generiert der e.P.b.-Archivserver ein geheimes Sitzungsticket. Dieses sendet er, verschlüsselt mit dem öffentlichen Schlüssel aus dem Zertifikat, an die Standesamtssoftware zurück (*Challenge*).
3. Der Benutzer entschlüsselt das Sitzungsticket mit seinem privaten Schlüssel. Dabei erhält die Standesamtssoftware das Ticket für diese Sitzung des Benutzers.
4. Die Standesamtssoftware bestätigt nun den Erhalt, indem sie eine Nachricht, welche das Ticket enthält, an den e.P.b.-Archivserver zurücksendet.
5. Erhält der e.P.b.-Archivserver das erwartete Ergebnis (*Response*), so sendet er seine Bestätigung an die Standesamtssoftware zurück.
6. Damit ist der Benutzer bei e.P.b.-Archivserver und Standesamtssoftware angemeldet. Jeder weitere Zugriff auf den Server wird mit dem ausgetauschten Ticket versehen.

### 6.2.2 Die Rechteverwaltung

Welche Rechte der Benutzer hat, wird lokal auf dem e.P.b.-Server verwaltet. Die Rechte sind dabei an das Benutzerzertifikat gekoppelt.

Das Rechtesystem in e.P.b. kennt folgende Arten von Berechtigungen:

- *Einsichtnahme* - Der Benutzer darf *lokal* gespeicherte Einträge suchen und abrufen.
- *Fernsuche* - Der Benutzer darf nach Einträgen auf allen angeschlossenen e.P.b.-Archivservern suchen[3].
- *Erzeugen und Ändern* - Der Benutzer darf neue Einträge im Archiv ablegen und vorhandene lokale Einträge fortführen.

[3] Bisher ist ein verteiltes e.P.b. nicht erlaubt. Dieses Konzept sieht jedoch ein verteiltes e.P.b. als mögliche Erweiterung vor.

- *Monitoring* - Der Benutzer hat Zugriff auf die Zugriffsstatistiken der e.P.b.-Archivserver. Diese erlauben eine Analyse der Benutzerzugriffe.
- *Rechtezuweisung* - Der Benutzer darf die Rechte anderer Benutzer ändern und neue Benutzer im System anlegen.
- *Administrierung* - Der Benutzer hat Zugriff auf das Konfigurationsinterface des e.P.b.-Archivservers.

Die Rechte für *Einsichtnahme, Fernsuche* und *Erzeugen und Ändern* können pro Personenstandsbuch vergeben werden. So ist eine genaue Steuerung der Zuständigkeiten möglich.

Bei einem Zugriff von einem fernen e.P.b.-Archivserver wird auch das Zertifikat des Benutzers übermittelt, der eine Anfrage stellt (siehe Abschnitt *Kommunikationssicherheit*). Für diese Benutzer können dieselben Berechtigungen vergeben werden, wie für lokale Benutzer. Jedoch ist momentan vorgesehen, dass für ferne Benutzer lediglich Rechte für die Einsichtnahme in lokale Einträge vergeben werden können. Die Freigabe der Benutzer, die dazu berechtigt sind, obliegt damit dem jeweiligen Standesamt, welches die gesuchten Einträge besitzt.

## 6.3 Kommunikationssicherheit

Die Kommunikation zwischen den Komponenten eines e.P.b. darf weder abgehört noch verfälscht werden. Die e.P.b.-Software muss geeignete Verfahren unterstützen, um diesen Gefahren entgegenzuwirken.

Dieses Konzept sieht dazu den Einsatz von SSL vor, das die Kommunikation auf unterster Ebene[4] absichert. Die Anwendungsebene hat bei dieser Verwendung des Protokolls idealerweise keine Berührungspunkte mit SSL. In diesem Konzept für ein e.P.b. gibt es dabei jedoch eine Ausnahme: Serverseitig werden die ausgestellten *Sitzungstickets* (siehe oben) an das für SSL verwendete Clientzertifikat gebunden. Dadurch kann ein einmal ausgestelltes Sitzungsticket nur für Nachrichten von dieser Arbeitsstation aus verwendet werden.

Die Kommunikation wird durch den Einsatz dieses Protokolls verschlüsselt und kann deshalb schwer abgehört werden. Um ein Abhören quasi unmöglich zu machen, sind starke Kryptoalgorithmen mit großen Schlüssellängen nötig. Da der Einsatz dieser starken Kryptomechanismen kein Hindernis darstellt[5], bietet eine korrekt konfigurierte SSL Implementierung praktisch beliebig skalierbaren Schutz.

Neben der Verschlüsselung ist mit SSL auch die Client- und Serverauthentifizierung möglich. Dieser Mechanismus soll im e.P.b. verwendet werden, um den Zugriff auf den e.P.b.-Archivserver auf ausgewählte Arbeitsstationen und/oder Benutzer zu beschränken. Dazu wird jeder dieser Arbeitsstationen und dem lokalen e.P.b.-Archivserver ein Schlüsselpaar und ein Zertifikat ausgestellt. Es handelt sich dabei nicht um qualifizierte Zertifikate einer Zertifizierungsstelle, sondern lediglich um einfache Zertifikate, die die Identität eines Rechners ausweisen.

[4] aus Sicht der Software

[5] Einziges Hindernis sind die U.S. Transportbeschränkungen für Kryptoprodukte. Es existieren jedoch europäische Produkte, die keinerlei Beschränkung unterliegen.

Zu jedem Zertifikat gehört ein privater Schlüssel. Dieser wird auf dem betreffenden Client oder Server in einem Keystore (mehr dazu in einem späteren Kapitel) verschlüsselt hinterlegt. Bei der Anmeldung an der Arbeitsstation muss deshalb ein Passwort eingegeben werden, damit der Zugriff auf den privaten Schlüssel hergestellt werden kann.

Alle zugangsberechtigten Arbeitsstationen müssen dem e.P.b.-Archivserver durch das Hinzufügen des Zertifikats zu seinem Keystore bekannt gemacht werden. Genauso muss das Serverzertifikat auf den Arbeitsstationen abgelegt werden. Das SSL-Protokoll wird so konfiguriert, dass vor der Kommunikation diese Zertifikate geprüft werden. Ist eines der beiden Zertifikate nicht gültig, wird die Verbindung verweigert.

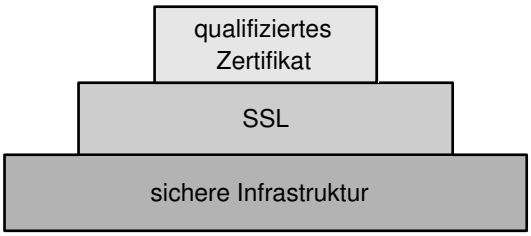


Abbildung 6.4: Der Schutz vor Missbrauch des e.P.b. basiert auf drei aufeinander aufbauenden Maßnahmen.

Dieses Verfahren bietet eine zusätzliche Hürde für interne Angreifer. Da die privaten Schlüssel aber lokal auf den Arbeitsstationen abgelegt werden, wird eine sichere Netzwerk Infrastruktur benötigt, die einen Zugriff anderer auf den privaten Schlüssel wirksam unterbindet. Das Sicherheitskonzept des e.P.b. mit den aufeinander aufbauenden Maßnahmen ist in Abbildung 6.4 dargestellt.

Eine Alternative zu solchen Client-Zertifikaten wäre es, das Zertifikat auf der Smartcard des Standesbeamten für die Kommunikation per SSL einzusetzen. Damit erreicht man eine höhere Sicherheit, da der private Schlüssel zu diesem Zertifikat nicht ausgelesen werden kann, da dieser niemals die Karte verlässt.

Nachteilig ist dabei jedoch, dass jedes Mal bei einem Zugriff auf den privaten Schlüssel die PIN der Karte eingegeben werden muss. Während der Arbeit an einem Server per SSL kann dies des Öfteren der Fall sein. Für den Standesbeamten könnte das schnell lästig werden. Deshalb wird nur die erste Variante für das e.P.b. vorgesehen.

Zwischen den verteilten e.P.b.-Archivservern wird das gleiche Verfahren eingesetzt. Jedes Serverzertifikat muss dabei jedem einzelnen e.P.b.-Archivserver bekannt gemacht werden. Erst dann kann der entfernte Zugriff erfolgen.

Der Aufbau der Infrastruktur des geplanten e.P.b. sieht folgendermaßen aus:

Die Clients eines Standesamtes arbeiten ausschließlich auf dem lokalen e.P.b.-Archivserver über eine, per SSL, gesicherte Verbindung. Der Zugriff auf entfernte Dokumente erfolgt dabei über den lokalen Archivserver, der die Anfrage weiterleitet. Dazu können die lokalen Archivserver per SSL gesicherte Verbindungen zu den entfernten Archivservern herstellen.

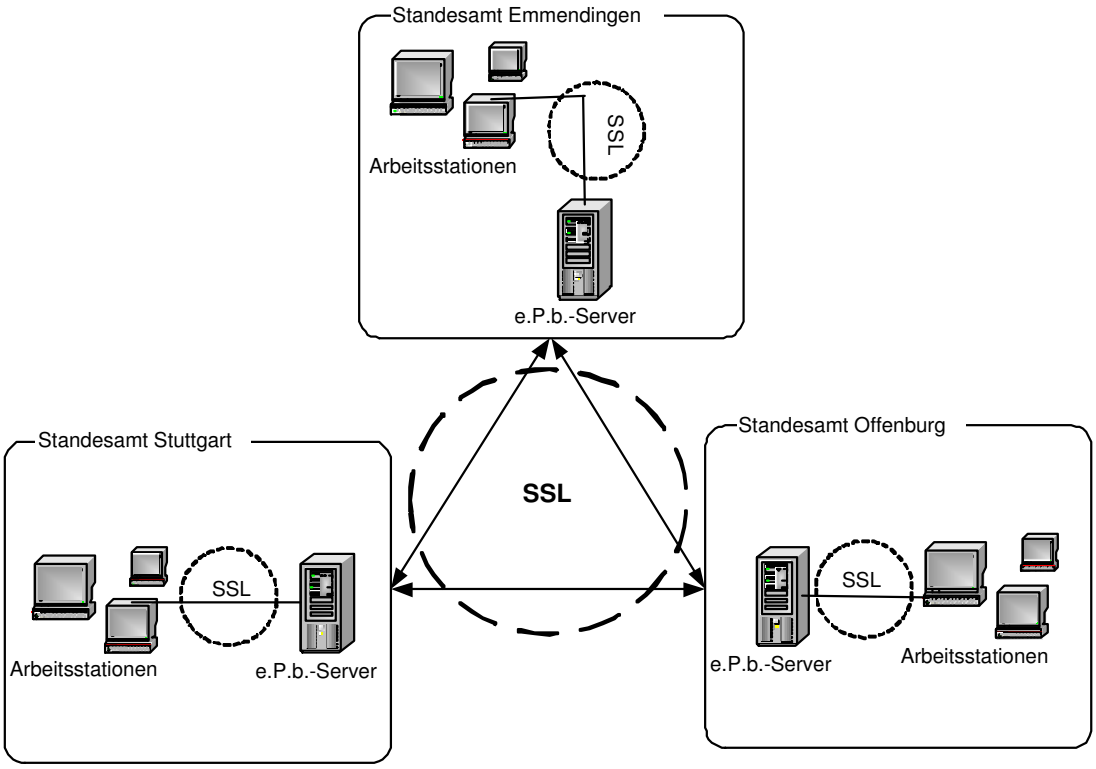


Abbildung 6.5: Die allgemeine Netzwerk-Struktur eines e.P.b.. Clients haben nur Zugriff auf den lokalen Archivserver. Für jede Fernabfrage wird eine SSL-Verbindung aufgebaut.

Dabei wird in der Standardkonfiguration lediglich eine Information, ob der gesuchte Eintrag in diesem Standesamt zur Verfügung steht, übermittelt. Existiert der gesuchte Eintrag, so wird gleichzeitig ein eindeutiger Schlüssel mitgeliefert. Anhand dieses Schlüssels kann dieses Dokument dann bei dem Standesamt nachgefragt werden. Dazu sind externe, sichere Kommunikationssysteme notwendig, die im e.P.b. nicht zur Verfügung gestellt werden[6].

Das Konzept sieht jedoch wahlweise eine Alternative vor: Ein Standesamt kann gezielt anderen Standesämtern oder Standesbeamten den Fernabruf von Einträgen gestatten (siehe oben, *Rechteverwaltung*). Bei jedem Abruf muss der Standesbeamte dabei den Grund des Abrufs angeben. Durch die Monitoringfunktionalität des e.P.b. lässt sich ein Missbrauch schnell aufdecken. Einmal freigeschaltet, macht es für die Clients keinen Unterschied, ob sie lokale oder entfernte Dokumente anfordern.

So könnte z.B. das Standesamt Stuttgart einem Mitarbeiter des Standesamtes Hamburg den Zugriff bei Angabe eines Grundes erlauben. Dieser Mitarbeiter kann dann beliebige Dokumente aus Stuttgart abrufen, wenn er jeweils einen Grund angibt.

[6] Hervorzuheben ist an dieser Stelle OSCI als sichere Transportplattform. Vom aktuellen Stand der Dinge ist davon auszugehen, dass die gesamte öffentliche Verwaltung in absehbarer Zeit mit der entsprechenden Infrastruktur ausgerüstet wird.

## 6.4 Hinweise zu möglichen Backupstrategien

Die digital unterschriebenen Einträge sind stark anfällig gegenüber Verfälschungen. Das ergibt sich aus den Eigenschaften der digitalen Signatur, die genau das verhindern soll. Deshalb muss dabei auch den eingesetzten Backupstrategien große Aufmerksamkeit gewidmet werden. Ein einzelnes falsches Bit in einem Eintrag genügt schon und die Signatur wird als ungültig erkannt.

Die eingesetzte Backupstrategie muss also sehr zuverlässig arbeiten und alle Daten bitgenau wieder herstellen können. Die Datenstruktur eignet sich dazu die großen Rohdaten (PDF-Dateien) gesondert auszulagern. Auf jeden Fall sind mehrere Sicherheitskopien, die an verschiedenen Standorten gelagert werden, sinnvoll.

Da sich diese gesicherten Daten nicht mehr ändern dürfen, ist die Sicherung auf Kosten sparende WORM-Speichermedien möglich. Spätere Zeitstempel werden gesondert gespeichert und können somit auf eigene Backupmedien gesichert werden.

Sinnvoll können auch zusätzliche Fehlerkorrekturmechanismen in den Einträgen selbst sein. Diese können vom System genutzt werden, um im Fall der Fälle die beschädigten Einträge automatisch korrigieren zu können.

Die Wiederherstellung von Einträgen kann dabei sehr zuverlässig arbeiten, denn die Software kann anhand der Signatur feststellen, ob die Daten korrekt wieder hergestellt wurden. Eine solche Fehlerkorrektur kann beim e.P.b. in die Persistenzkomponente integriert werden.

Die endgültige Strategie zur Sicherung des Datenbestands ist jedoch nicht Bestandteil dieser Arbeit.

# Teil III

# Entwurf der Komponenten

# 7. Skizzierung der Strukturen

## 7.1 Die Architektur

Das e.P.b. soll als Webservice implementiert werden. Dieser Standard definiert den Datenaustausch zwischen Softwarekomponenten sowie den Aufruf von entfernten Funktionen. Als Protokoll wird *HTTP* eingesetzt, über das per *SOAP*-Nachrichten kommuniziert wird.

Die im Folgenden vorgestellte Architektur beschreibt den Aufbau eines lokalen e.P.b.-Systems. Die Anbindung an entfernte e.P.b.-Archivserver wird Thema eines späteren Kapitels sein. Diese Beschreibung wird nicht ins Detail der Architektur eingehen. Das ist nicht Bestandteil dieser Arbeit. Hier soll lediglich ein Überblick gegeben werden, in welchem Umfeld die konzipierten Sicherheitskomponenten eingebettet werden. Die resultierenden Anforderungen an die Sicherheitskomponenten werden im nächsten Abschnitt vorgestellt.

Abbildung 7.1 zeigt den Aufbau des Systems. Aus logischer Sicht kann man vier hauptsächliche Komponenten ausmachen. Der komplexe Client ist die Software zur Bearbeitung von Personenstandseinträgen. Das eigentliche e.P.b. besteht aus zwei Komponenten. Das Webservice-Framework (WSF) steht im Dialog mit dem Client. Er vermittelt die Aufrufe an die eigentliche Anwendungskomponente (APP) weiter. Die Anwendungskomponente hat Zugriff auf eine Datenbank, in der die Einträge abgelegt werden.

Der Client sendet SOAP-Nachrichten an den Frontcontroller. Dieser verfolgt den aktuellen Zustand der Kommunikation. Festgelegte Konversationen regeln den Ablauf der Kommunikation. Das WSF ruft die entsprechenden Funktionen der Anwendung auf und reicht die Ergebnisse an den Client weiter. Sendet der Client Nachrichten in unerwarteter Art und Reihenfolge, so werden die Aufrufe nicht an die Anwendung weitergegeben.

Für die Ergebnisse stellt das WSF einen Container bereit. Die Anwendung beschreibt den Container und meldet dem WSF lediglich das Ende der Bearbeitung. Um das Format der Daten im Container hat das WSF kein Wissen. Die Daten werden so, wie sie dort stehen, verpackt in einer Soap-Nachricht, an den Client übermittelt.

Die Anwendungskomponente besitzt ebenfalls einen Controller (Applicationcontroller), der die Kommandos vom WSF entgegennimmt. Die Durchführung erfordert meist eine ganze Reihe von Methodenaufrufen. Der Controller startet Transaktionen, um die komplexen Anforderungen abzuarbeiten. Diese Transaktionen enthalten die Anwendungslogik des e.P.b..

Die unterste Ebene der Anwendungsschicht bilden die Persistenzkomponente und die Sicherheitskomponente. Die Persistenzkomponente kapselt Anfragen an die Datenbank auf der alle Personenstandseinträge abgelegt werden. Somit ist das e.P.b. unabhängig von der eingesetzten Datenbank. Die Persistenzkomponente stellt Schnittstellen zur Verfügung, die einen effizienten Zugriff auf Einträge ermöglichen.

Die Sicherheitskomponente stellt alle Funktionen zur Verfügung, die Fragen der Signatur und der Signaturerhaltung betreffen. Ihre Funktionen werden im Unterschied zur Persistenzkomponente nicht automatisch beim Laden und Speichern von Einträgen durchgeführt, sondern müssen von den entsprechenden Transaktionen angestoßen werden.

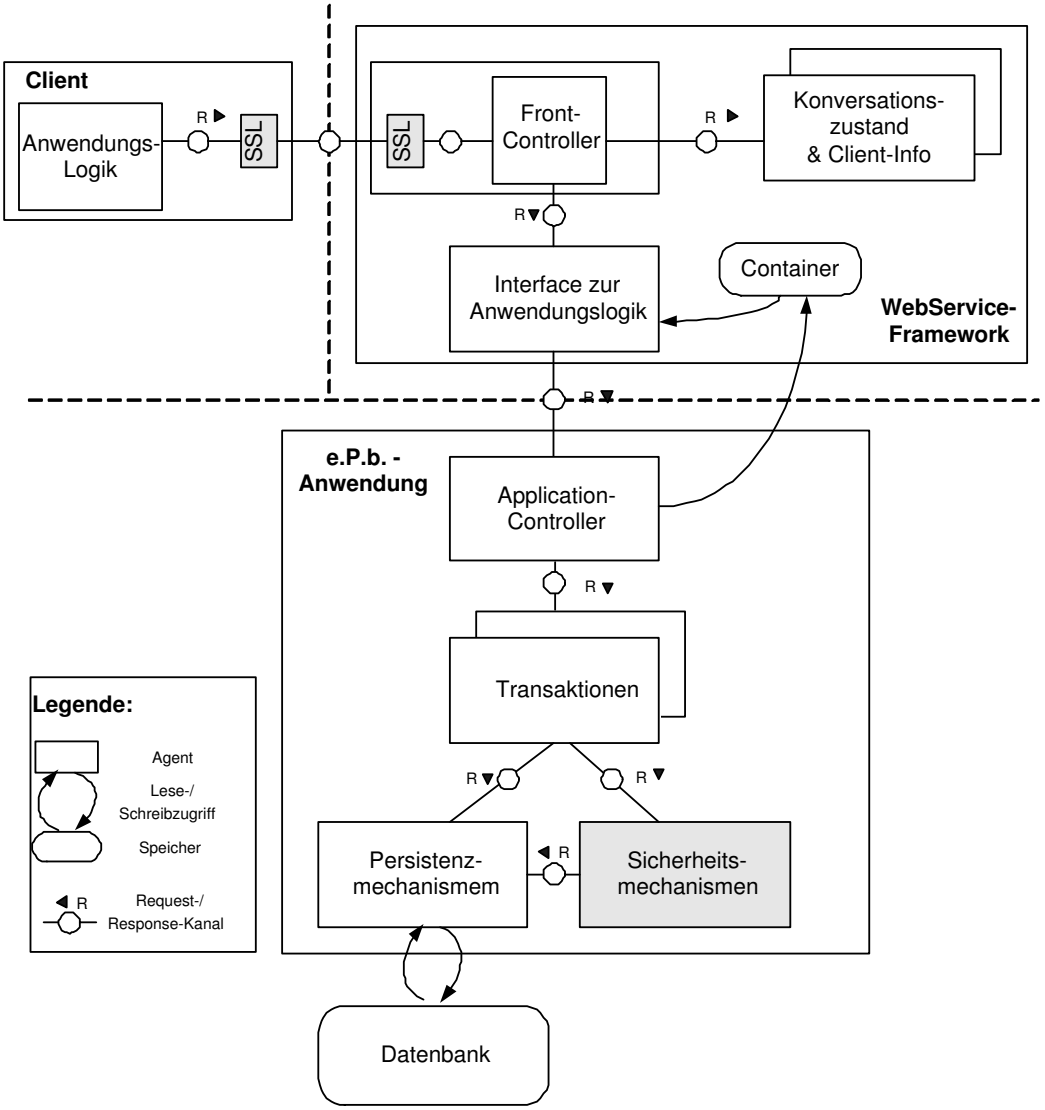


Abbildung 7.1: Die Architektur eines lokalen e.P.b.-Systems

Die Sicherheitskomponente kapselt das Format und die Art der signaturerhaltenden Archivierung. Zukünftige Ansätze, welche ein anderes Konzept der Beweiserhaltung zugrunde legen, können durch die Anpassung der Sicherheitskomponente implementiert werden.

Zusätzlich zu der Signaturfunktionalität stellt die Sicherheitskomponente kryptografische Hilfsmethoden zur Authentifikation von Benutzern zur Verfügung. Dazu gehören die Prüfung der Identität (per POP) und das Ausstellen von Sitzungstickets. Dazu hat die Sicherheitskomponente Zugriff auf die Zertifikatsdatenbank der bekannten Benutzer. Benutzerrechte werden hier jedoch nicht verwaltet. Das ist Aufgabe des Applicationcontrollers.

Das WSF kennt also lediglich die Konversationen und die zugehörigen Kommandos, die an die Anwendung weitergegeben werden. Damit kann das Framework sehr generisch gestaltet werden.

Auch die Benutzerrechte werden nicht im WSF verwaltet. Will sich ein Benutzer am System anmelden, so erfragt das WSF von der Anwendung die Benutzerrechte und lässt sich ein Ticket ausstellen, über das die Nachrichten später diesem Benutzer zugeordnet werden können. Dies dient jedoch nur dazu Aktionen zu verhindern, zu denen der Benutzer offensichtlich keine Rechte hat. Die Anwendung prüft trotzdem bei jeder Aktion, ob der Benutzer dazu berechtigt ist.

Diese Architektur erlaubt, dass die Anwendungskomponente als klassische Client/Server Applikation implementiert werden kann. Aus Sicht der Anwendung eröffnet ein Benutzer eine Sitzung und führt nacheinander Aktionen durch. Das WSF setzt dazu, mithilfe des Tickets, die Kommunikation mit Client über das zustandslose HTTP-Protokoll in einen kontinuierlichen Fluss von Aktionen um.

### 7.1.1 Resultierende Anforderungen an die Sicherheitskomponente

In Abbildung 7.1 sind die Komponenten grau eingefärbt, die Bestandteil dieser Arbeit sind. Der größte Teil ist die Sicherheitskomponente, welche Signatur und Benutzerauthentifikation regelt.

Die Absicherung der Kommunikation zwischen Client und Server erfolgt auf unterster Ebene durch das SSL-Protokoll. Hierbei wird lediglich eine Authentifikation der beteiligten Rechner durchgeführt, damit nur berechtigte Workstations Zugriff haben. Das SSL-Protokoll arbeitet voll transparent zum darüber liegenden System.

Die Einrichtung einer solchen SSL-Verbindung erfordert die korrekte Konfiguration des Servers und die Erstellung und Einrichtung der Rechnerzertifikate. Diese Themen werden in eigenen Kapiteln bearbeitet.

Die Sicherheitskomponente muss eine API definieren, über die den Transaktionen alle Funktionen bereitgestellt werden, die das Signatursystem betreffen. Außerdem muss die Konfiguration und die Verwaltung von Betriebsparametern für das Signatursystem ermöglicht werden. Zur Unterstützung der Benutzerauthentifikation werden darüber hinaus noch Methoden für den POP und für das Erstellen von Sitzungstickets benötigt.

## 7.2 Die Datenstruktur für Personenstandseinträge

In diesem Abschnitt wird die Datenstruktur für das e.P.b. vorgestellt. Aktuell wird lediglich die Variante *ohne* Jahresarchive verfolgt. Die Gründe hierzu wurden im vorigen Kapitel schon besprochen. Zur Komplettierung und zur Veranschaulichung der Probleme, die mit Jahresarchiven einhergehen, wird hier auch auf diese Variante kurz eingegangen.

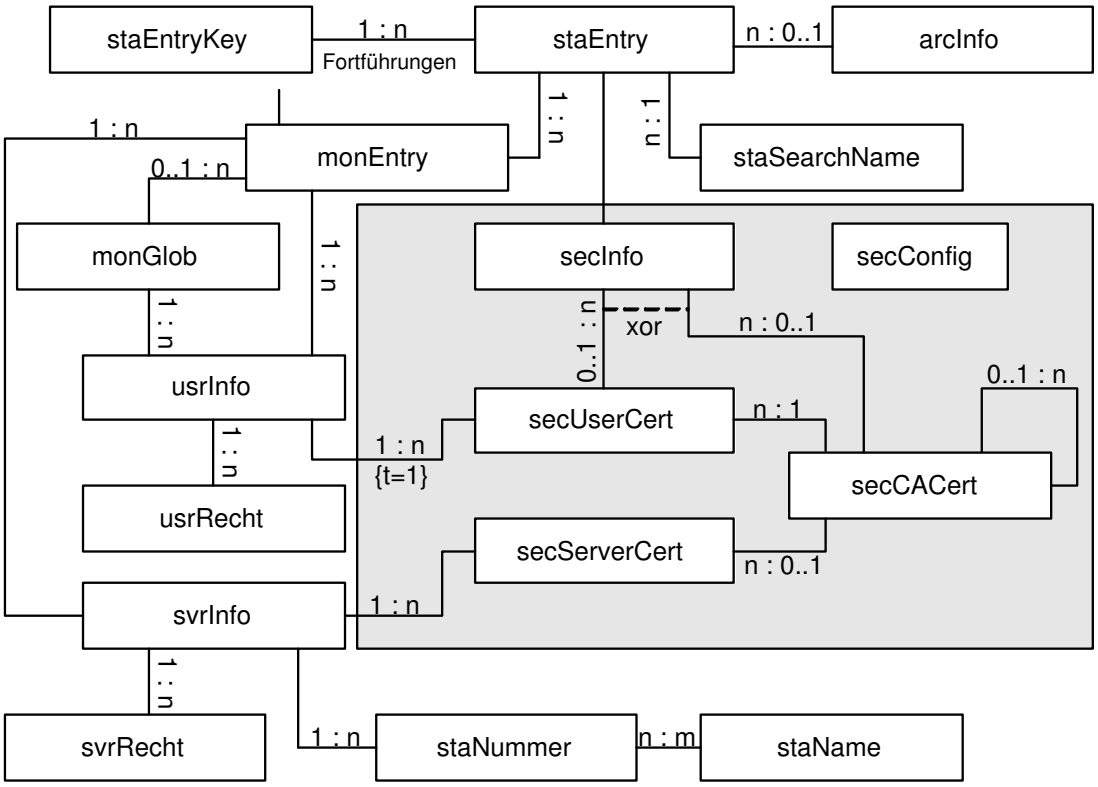


Abbildung 7.2: Die Datenstruktur für e.P.b.

### 7.2.1 Variante ohne Jahresarchive

Abbildung 7.2 zeigt die Datenbankstruktur für das e.P.b.. Die Tabellen, die mit einem grauen Kasten hinterlegt wurden, betreffen die Sicherheitskomponente und werden im Folgenden näher vorgestellt.

Einem Personenstandseintrag können im Laufe der Zeit beliebig viele Einträge in der Tabelle *secInfo* (Sicherheitsinformationen) zugeordnet werden. Dabei kann es sich um die Originalsignatur, eine OCSP-Antwort oder um einen Zeitstempel zur Beweiserhaltung handeln. Zeitstempel weisen dabei die Besonderheit auf, dass sie sich auf den Personenstandseintrag und alle bisherigen Sicherheitsinformationen beziehen.

Um die Gültigkeit der Originalsignatur zu belegen, müssen alle zugehörigen Einträge in dieser Tabelle ausgeliefert werden. Zusammen ermöglichen sie eine lückenlose Beweisführung.

Eine Sicherheitsinformation enthält Angaben über das verwendete Speicherformat und den Algorithmus. Dadurch lässt sich die abgespeicherte Signatur, die OCSP-Antwort oder der Zeitstempel später auslesen und interpretieren.

Daneben ist schließlich ein Verweis auf das zugrunde liegende Zertifikat enthalten. Bei der Originalsignatur ist dies ein Userzertifikat (*secUserCert*), bei OCSP-Antworten und Zeitstempeln ist dies das Zertifikat des Trustcenters (*secCACert*). Diese Zertifikate werden in eigene Tabellen ausgelagert, um unnötige Redundanzen zu vermeiden. Ansonsten müsste die gesamte Zertifikatskette zu jedem Eintrag abgelegt werden.

Im vorliegenden Datenmodell wird die Zertifikatshierarchie dadurch dargestellt, dass ein Userzertifikat den Verweis auf das Zertifikat des Trustcenters enthält, welches das Userzertifikat signiert hat. Trustcenterzertifikate können einen Verweis auf das Wurzeltrustcenter enthalten, dass ihr Zertifikat herausgegeben hat.

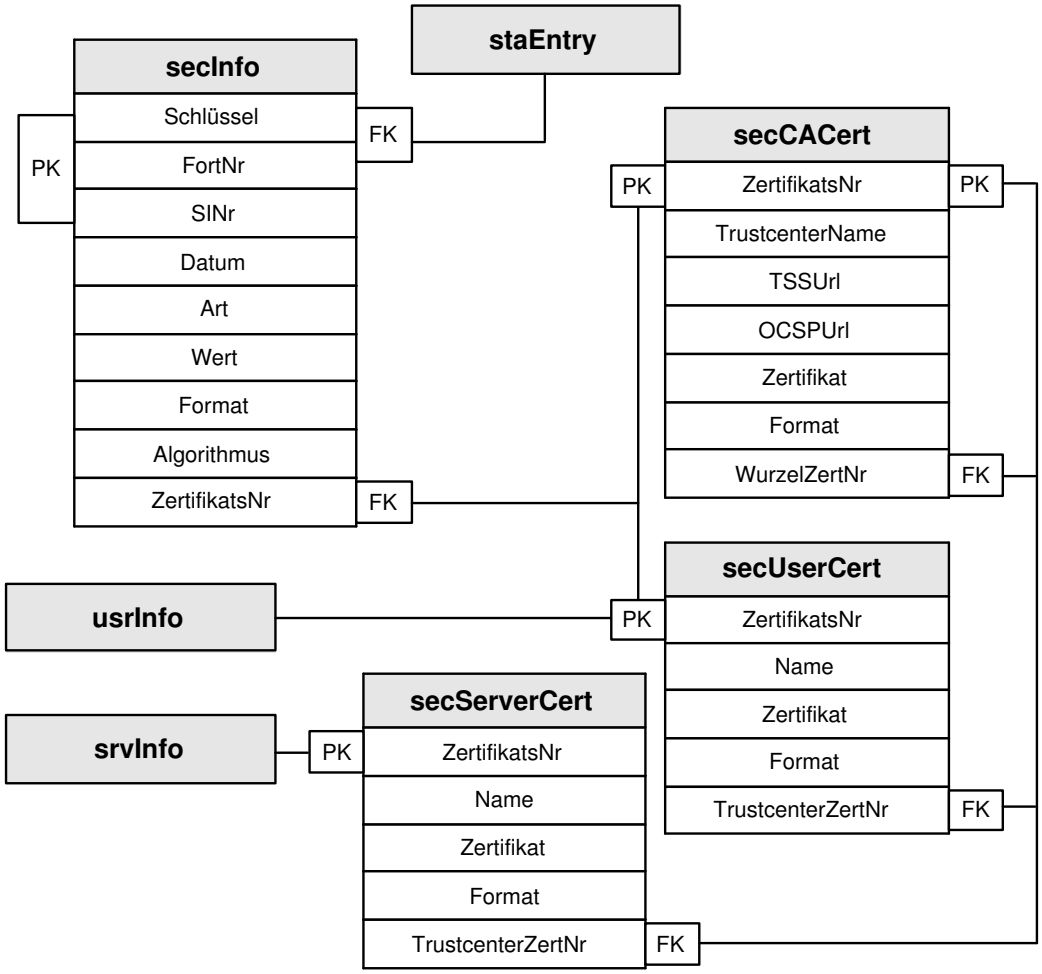


Abbildung 7.3: Die Tabellen für die Signaturkomponente (1).

Die Userzertifikate werden auch von einer Tabelle außerhalb des Zugriffsbereichs der Sicherheitskomponente referiert. In der Userzertifikat-Tabelle sind die Zertifikate aller zugriffsberechtigten Benutzer enthalten. Die Tabelle *secUser*, zur Verwaltung der Benutzerrechte, verwendet Verweise in diese Tabelle, um die Benutzer eindeutig zu identifizieren.

Neben den User- und Trustcenterzertifikaten ist eine Tabelle (*secServerCert*) für die Zertifikate der angeschlossenen, externen e.P.b.-Server vorgesehen. Diese dienen zur Authentifikation unterhalb der Server. Die Tabelle srvInfo, außerhalb der Sicherheitskomponente, referiert diese Zertifikate.

Außerhalb der Tabellen für das Signaturkonzept steht die Tabelle *secConfig*. Sie wird zur Konfiguration der Sicherheitskomponente verwendet. Zur Nomenklatur: Jede Tabelle, deren Namen mit der Vorsilbe *sec* beginnt, kann von der Sicherheitskomponente selbst (über die Persistenzkomponente) zugegriffen werden. Andere benötigte Daten müssen ihr von den aufrufenden Transaktionen übergeben werden.

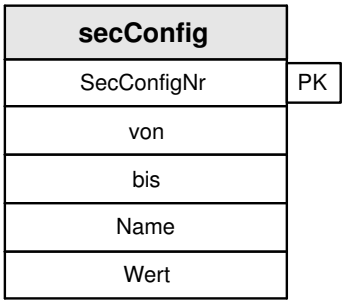


Abbildung 7.4: Die Tabelle zur Konfiguration der Signaturkomponente.

Die Abbildungen 7.3 und 7.4 zeigen die Datenfelder der Tabellen zur Realisierung der erwarteten Funktionalität.

Über den *Schlüssel* und die *Fortführungsnummer* lässt sich eine Sicherheitsinformation genau einem Personenstandseintrag (jede Fortsetzung zählt dabei als eigenständiger Eintrag) zuordnen.

In der vorliegenden Abbildung handelt es sich bei dem Schlüssel um eine Vereinfachung. Tatsächlich setzt er sich aus den Feldern Standesamtsnummer, Buch, Jahr und Eintragnummer zusammen. Da diese Informationen für die Sicherheitskomponente nicht von Belang sind, wurden sie in der Abbildung weggelassen.

Die Sicherheitsinformationen zu einem Eintrag werden dann jeweils mit einer *SINr* versehen, die für jeden Eintrag bei Null beginnt und hochgezählt wird. Die SINr und der Fremdschlüssel zum Verweis auf einen Eintrag bilden den Primary Key in dieser Tabelle.

Das *Datumsfeld* enthält die Angabe über die genaue Zeit und das Datum, an dem der Wert erstellt wurde. Das Feld *Art* gibt Aufschluss, um was für eine Sicherheitsinformation es sich handelt. Die möglichen Werte sind dabei `signature`, `ocsp` oder `timestamp`.

Im Feld *Wert* sind die Rohdaten der Sicherheitsinformation enthalten. Über die Informationen in *Art*, *Format* und *Algorithmus* können diese Daten interpretiert werden. Die Signaturen und Zeitstempel beziehen sich dabei auf die Rohdaten des Eintrags und aller vorigen Sicherheitsinformationen, sortiert nach aufsteigender SINr.

Die Formate und Algorithmen werden als Bezeichner gespeichert. Das Format für eine Signatur könnte z.B. `SHA1withRSA` lauten. Dadurch wird eine Signatur mit SHA-1 als Hashfunktion und RSA zur Verschlüsselung definiert.

Die verfügbaren Speicherformate und Algorithmen werden vom eingesetzten Java Securityprovider bereitgestellt. Um von diesen Bezeichnungen unabhängig zu sein, werden später Mapping-Tabellen die korrekte Interpretation sicherstellen. Damit werden diese Angaben unabhängig von einem konkreten Securityprovider.

Mehr zu den JCE-Providern und den e.P.b.-Bezeichnern für Algorithmen und Speicherformate im Abschnitt *Java Cryptography Architecture*.

Wie schon erwähnt, existieren drei Tabellen für Zertifikate. In diesen Tabellen sind die Rohdaten eines Zertifikats und das Format enthalten, um ein Zertifikat zu lesen und zu interpretieren. Daneben sind jedoch auch einige Daten aus dem Zertifikat herausgezogen, um einen effizienteren Zugriff auf diese Informationen zu ermöglichen.

Bei den Benutzerzertifikaten und Serverzertifikaten ist das der Name. Bei den Trustcenterzertifikaten sind es der Name des Trustcenters und die Adressen, über die OCSP- und Zeitstempeldienste erreicht werden können.

Die Primärschlüssel der beiden Tabellen *secCACert* und *secUserCert* müssen bei diesem Datenmodell Tabellen übergreifend eindeutig sein. Nur dadurch ist garantiert, dass das richtige Zertifikat zu einer Sicherheitsinformation gefunden werden kann.

Die letzte Tabelle *secConfig* zur Konfiguration der Sicherheitskomponente enthält Name - Wert Paare. Ein Eintrag könnte z.b. `Signaturalgorithmus - SHA1withRSA` sein. Zusätzlich geben die Felder *von* und *bis* den Gültigkeitszeitraum dieses Konfigurationseintrags an.

#### 7.2.1.1 Die Datenstruktur am Beispiel

Zur Verdeutlichung der Datenstruktur nun ein Beispiel. Ein neuer Eintrag wurde in das e.P.b. eingespeichert. Dabei wurde die Originalsignatur abgelegt, ein Zeitstempel erstellt und eine OCSP-Antwort eingeholt. Abbildung 7.5 zeigt den Inhalt der Tabellen.

Der neue Eintrag hat den Schlüssel XXX und die Fortführungsnummer 1. Damit handelt es sich hierbei um die erste Fortführung eines Eintrags. Der Ersteintrag erhält die Fortführungsnummer 0. Die Fortführungen sind aufsteigend nummeriert und beginnen ab Fortführungsnummer 1.

In der Tabelle Sicherheitsinformationen sind zu diesem Eintrag die Originalsignatur, der initiale Zeitstempel und eine OCSP-Antwort abgelegt. Diese sind von 0 bis 2 aufsteigend nummeriert (SINr).

Zeitstempel berücksichtigen die SINr. Sie erstrecken sich über die Rohdaten des Eintrags und über alle Rohdaten der Sicherheitsinformationen in Reihenfolge der aufsteigenden SINr.

Zu den Daten ist jeweils das Speicherformat und der verwendete Algorithmus angegeben, um später auf die Daten zugreifen zu können. Die ZertNr verweist auf das verwendete Zertifikat. In diesem Beispiel hat Müller den Eintrag signiert (ZertNr 0).

*Achtung!* Nur Sicherheitsinformationen von der Art `signature` verweisen in die Userzertifikate-Tabelle. Bei `timestamp` und `ocsp` wird in die Trustcenterzertifikate-Tabelle verwiesen. Der Zeitstempel und die OCSP-Antwort wurden also von SignTrust signiert.

Das Zertifikat von Müller ist von SignTrust ausgestellt. An dieses Trustcenter wird also eine OCSP-Abfrage nach dem Status dieses Zertifikats gerichtet. Das Zertifikat von SignTrust ist wiederum vom Trustcenter der RegTP unterschrieben.

**staEntry**

| secInfo | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Schlüssel** | **FortNr** | **SINr** | **Datum** | **Art** | **Wert** | **Format** | **Algorithmus** | **ZertNr** |
| XXX | 1 | 0 | 10.07.2004 14:53 | signature | [SIGNATUR] | CMS | SHA1withRSA | 0 |
| XXX | 1 | 1 | 10.07.2004 14:54 | timestamp | [ZEITSTEMPEL] | TSS1 | SHA1withRSA | 0 |
| XXX | 1 | 2 | 10.07.2004 14:54 | ocsp | [OCSP] | OCSP1 | SHA1withRSA | 0 |

| secUserCert | | | | |
|---|---|---|---|---|
| **ZertNr** | **Name** | **Zertifikat** | **Format** | **TrustcenterZertNr** |
| 0 | Müller | [ZERTIFIKAT] | ASN1-X.509 | 0 |
| 1 | Schneider | [ZERTIFIKAT] | ASN1-X.509 | 0 |

| secCACert | | | | | | |
|---|---|---|---|---|---|---|
| **ZertNr** | **TrustcenterName** | **TSSUrl** | **OCSPUrl** | **Zertifikat** | **Format** | **WurzelZertNr** |
| 0 | SignTrust | 131.175.204.3 | 131.175.204.3 | [ZERTIFIKAT] | ASN1-X.509 | 1 |
| 1 | RegTP | 131.161.106.2 | 131.161.106.3 | [ZERTIFIKAT] | ASN1-X.509 | null |

Abbildung 7.5: Beispiel: Alle Sicherheitsinformationen für einen Eintrag nach der Einspeicherung.

Das zweite Beispiel zeigt Einträge in die secConfig-Tabelle. Diese sind jeweils mit einem Gültigkeitszeitraum versehen. Im Beispiel ist das Feld *bis* jeweils null. Das bedeutet, dass diese Einstellungen gerade verwendet werden und die aktuellsten sind.

Die Einstellungen werden jeweils als Name - Wert Paar gespeichert. Im Beispiel ist als akzeptierter Signaturalgorithmus für neue Einträge SHA-1 mit RSA vorgesehen. Die minimale Schlüssellänge muss dabei 1024 Bit sein.

| secConfig | | | | |
|---|---|---|---|---|
| **secConfigNr** | **von** | **bis** | **Name** | **Wert** |
| 0 | 12.05.2004 | null | SignatureAlg | SHA1withRSA |
| 1 | 12.05.2004 | null | minKeyLenght | 1024 |

Abbildung 7.6: Beispiel: Einträge in der Konfigurationstabelle.

Bisher sind lediglich die zwei, im Beispiel gezeigten, Einstellungen vorgesehen. Weitere Optionen können jederzeit hinzugefügt werden.

### 7.2.2 Variante mit Jahresarchiven

Wie angekündigt folgt hier nun die Beschreibung des alten Konzepts (mit Jahresarchiven). Diese Variante entspricht *nicht* dem aktuellen Stand der Entwicklung von e.P.b. Außerdem handelt es sich dabei, im Gegensatz zum gerade dargestellten Datenmodell, um ein nicht normalisiertes Datenschema, das so nicht in einer Datenbank dargestellt werden kann.

Das eingesetzte Datenformat, in dem die Einträge in der Datenbank abgelegt werden, muss die erwarteten Eigenschaften darstellen können. Deshalb enthält ein Eintrag außer seinen Daten ein Feld mit allen Signaturinformationen, die die Sicherheitskomponente verwaltet.

Hier werden diese Daten ebenfalls *Sicherheitsinformationen* genannt. Sie enthalten die Erstsignatur des Erstellers. Diese bezieht sich ausschließlich auf die Rohdaten des Eintrags. Direkt darauf folgt ein Feld für den initialen Zeitstempel.

Die Antwort auf die OCSP-Abfrage nach dem Status des Erstellerzertifikats und allen Zertifikaten des Zertifikatspfads schließt die Informationen ab, die zunächst einem neuen Eintrag angehängt werden.

Beweiserhaltende Folgezeitstempel werden in einer Liste ebenfalls in den Sicherheitsinformationen untergebracht. Diese Liste ist jedoch normalerweise leer, da vorgesehen ist, alle Einträge eines Jahres zum Jahresabschluss einem Archiveintrag zuzuordnen. Zeitstempel werden dann dort gespeichert.

Ist ein Eintrag einem Archiv zugeordnet, so wird ein Verweis auf den Archiveintrag in seinen Sicherheitsinformationen hinzugefügt. Das ist notwendig, da ein Zusammenstellen der benötigten Signaturinformationen nur über den Archiveintrag möglich wird.

Das impliziert auch, dass die prüfende Instanz Zugriff auf alle Einträge eines Archivs haben muss. Einem externen Prüfer müssen also alle Einträge dieses Archiveintrags mit allen Validierungsinformationen vorgelegt werden.

Archiveinträge fassen, wie schon gesagt, viele Einträge zusammen (z.B. als Jahresarchiv) und enthalten alle weiteren Signaturinformationen, die das gesamte Archiv betreffen. Sie verweisen deshalb auf alle enthaltene Einträge.

Hashwerte, die für neue Zeitstempel benötigt werden, berücksichtigen die Struktur der Daten nicht. Die Hashfunktionen erfassen die einzelnen Daten, als stünden sie nahtlos nacheinander. Damit ist gesichert, dass bei einer Umstellung in der Datenablage keine Signaturen ungültig werden können.

Neben den Signaturen in der Datenstruktur müssen auch die Parameter gespeichert werden, mit denen sie erzeugt wurden. Dabei geht es um die verwendeten Algorithmen, Zertifikate und Speicherformate.

Bei der Erstellersignatur muss der Zertifikatspfad gespeichert werden auf dem die Signatur beruht. Von dem Zertifikat des Erstellers bis zum ausstellenden Root-Trustcenter. Außerdem muss der verwendete Signaturalgorithmus angegeben werden. Soll später überprüft werden, ob eine Signatur rechtsgültig war, so kann die Vertrauenskette bis zur Root-Instanz nachvollzogen werden. Die prüfende Instanz muss dann entscheiden, ob diese Root-Instanz zu dem damaligen Zeitpunkt vertrauenswürdig war.

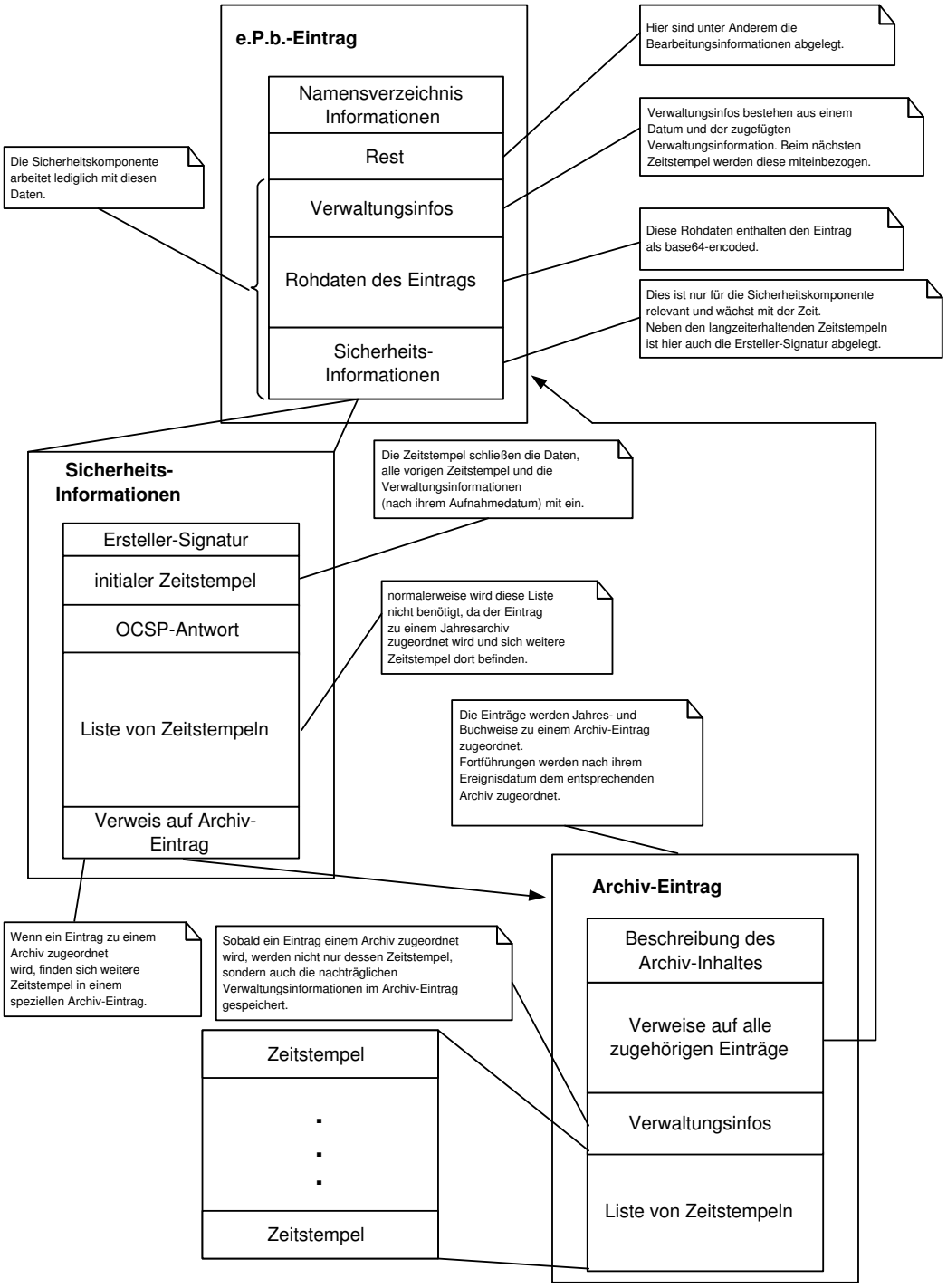


Abbildung 7.7: Das Datenformat für Personenstandseinträge im e.P.b.

Bei einer OCSP-Abfrage und den Zeitstempeln gilt Ähnliches. Zusätzlich muss noch das jeweilige Speicherformat abgelegt werden, um sie später richtig interpretieren zu können. Das Feld für die OCSP-Antwort enthält die OCSP-Antworten für alle Zertifikate des Zertifikatspfades.

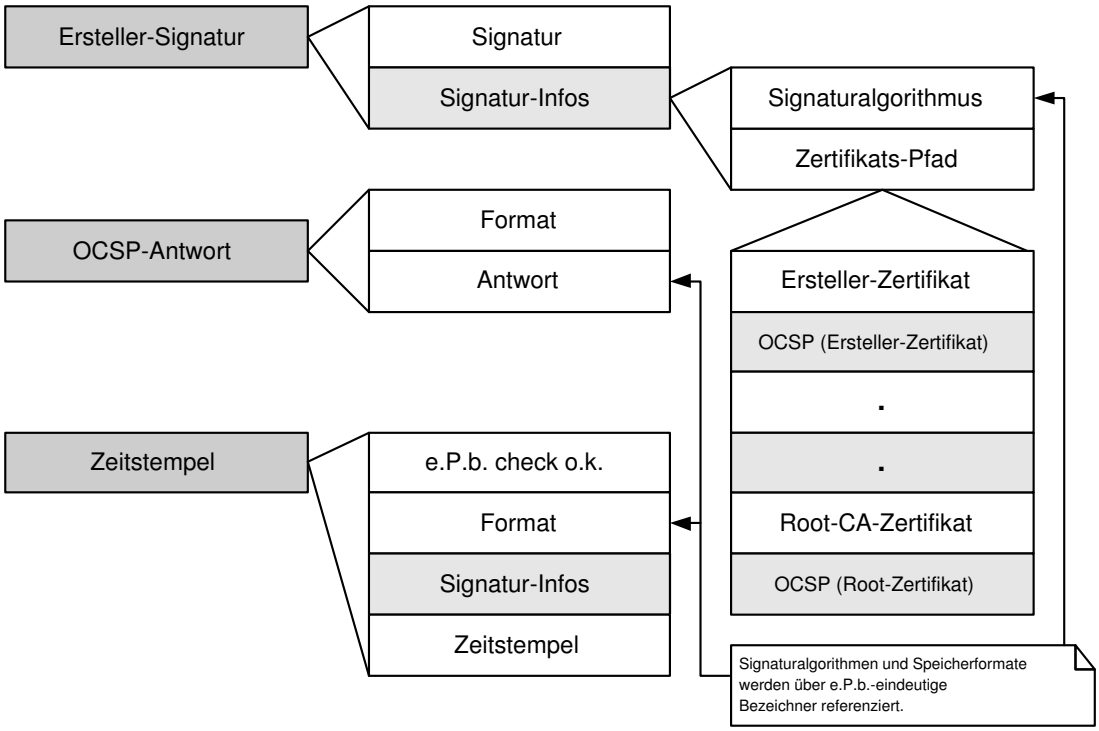


Abbildung 7.8: Das Datenformat für die Signaturparameter.

Zusätzlich zum aktuellen Datenmodell sind hier Verwaltungsinformationen vorgesehen, die Informationen über eventuelle Fortführungen enthalten. Dies wird nötig, da in dieser Variante kein eindeutiger Schlüssel für alle Fortführungen eines Eintrags vorhanden ist. Wird also nach Eintrag XXX gesucht, kann es sein, dass nun Eintrag YYY die aktuellste Fortführung beinhaltet. Die Verwaltungsinformationen in XXX verweisen dann auf YYY.

#### 7.2.2.1 Die Datenstruktur am Beispiel

Zur Illustration der Datenstruktur nun einige Beispiele für gespeicherte Einträge. Die Beispiele bauen aufeinander auf und zeigen den Wandel der Datenstruktur für einen Eintrag über einige Jahre.

**Neuer Eintrag** Abbildung 7.9

Bei der Einspeicherung in das e.P.b.-Archiv wird zunächst die Signatur geprüft. Dann werden alle Validierungsinformationen eingeholt und zum Eintrag abgelegt. Im Einzelnen sind das: ein initialer Zeitstempel und die OCSP-Antworten zu allen Zertifikaten des Zertifikatspfads.

Neben diesen Validierungsinformationen werden auch jeweils die verwendeten Algorithmen und Speicherformate vermerkt. Im Beispiel sind das „RSA+SHA-1“ für die Signatur, “X.509“ für die Zertifikate, „TS1“ für ein Speicherformat für Zeitstempel und „OCSP-AW1“ als Speicherformat für OCSP-Antworten. Diese Informationen werden zur Interpretation der Daten benötigt.

Der Eintrag selbst wird base64 kodiert übermittelt. Das e.P.b. behandelt diese Daten als Rohdaten und hat kein Wissen um den inneren Aufbau des Eintrags. Damit ist das e.P.b. unabhängig von geänderten gesetzlichen Regelungen für den Aufbau von Personenstandseinträgen.

Dem kompletten Eintragsdatensatz wird schließlich ein eindeutiger Schlüssel zugeordnet. Damit wird er eindeutig referenzierbar, sobald er in der Datenbank abgelegt wurde.

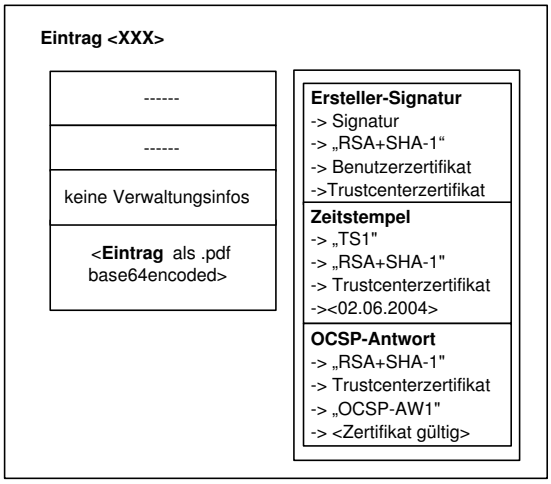


Abbildung 7.9: Ein Datenbeispiel für einen neuen Eintrag

**Fortführung des Eintrags im aktuellen Jahr** Abbildung 7.10

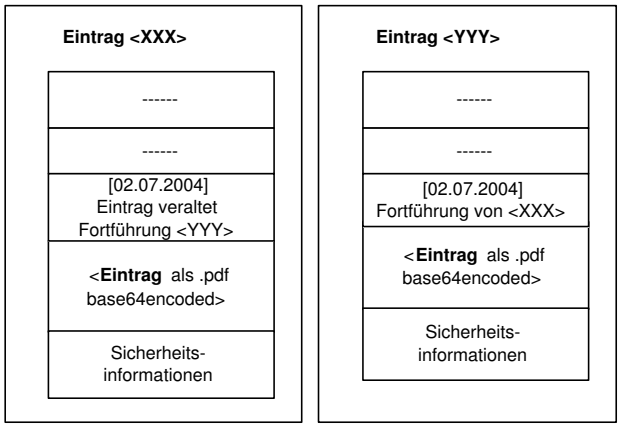


Abbildung 7.10: Der Eintrag wurde fortgeführt.

Nun wird der eben eingespeicherte Eintrag fortgeführt. Dazu wird ein neuer Eintrag erzeugt, der die Fortführung, als zweite Schicht zum alten Eintrag, enthält.

Die Fortführung wird dadurch kenntlich gemacht, dass die Verwaltungsinformationen des alten Eintrags diesen nun als veraltet kennzeichnen und auf die Fortführung verweisen. Ebenso verweisen die Verwaltungsinformationen des neuen Eintrags auf den alten Eintrag.

Bei allen folgenden Zeitstempeln werden die Verwaltungsinformationen später mit einbezogen. Dadurch werden sie vor Veränderung geschützt. Ein Eintrag wird standardmäßig in seiner letzten Form (mit allen Fortführungen) ausgeliefert.

**Zuordnung zu einem Archiv** Abbildung 7.11

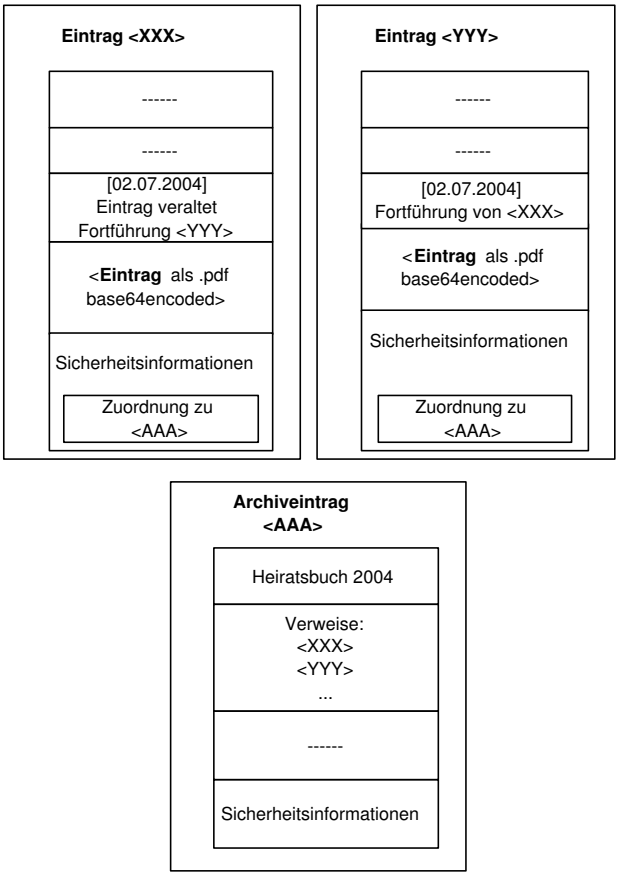


Abbildung 7.11: Alle Einträge aus 2004 wurden einem Archiveintrag zugeordnet.

Zum Jahresabschluss werden alle neuen Einträge zu einem passenden Archiveintrag zugeordnet. In diesem Beispiel gehören der Eintrag und seine Fortführung in das Heiratsbuch. Anfang 2005 werden die beiden Einträge nun einem Archiveintrag mit dem Namen *Heiratsbuch 2004* zugeordnet.

Die Einträge erhalten eine Notiz dieser Zuordnung in deren Sicherheitsinformationen. Das ist nötig, damit bei einer späteren Überprüfung der Signatur alle weiteren Zeitstempel im Archiveintrag gesucht werden.

Die Einträge stehen trotzdem weiterhin als einzelne Datensätze in der Datenbank. Der Zugriff kann weiterhin über den eindeutigen Schlüssel erfolgen. Archiveinträge wirken sich nur auf die Maßnahmen zur Erhaltung der Signatur aus.

**Fortführung des archivierten Eintrags** Abbildung 7.12

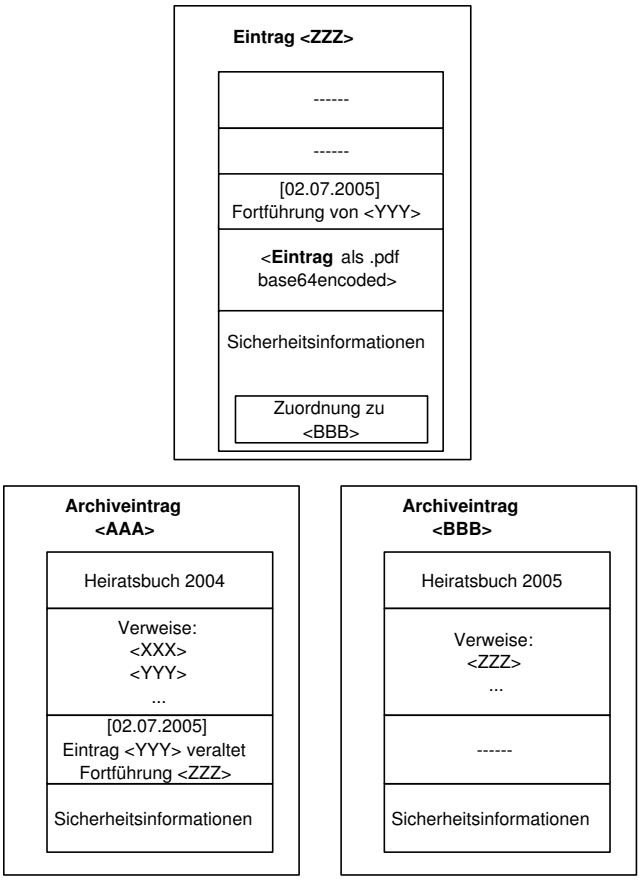


Abbildung 7.12: Nach der Zuordnung zu einem Archiveintrag wird der erste Eintrag wiederum fortgeführt.

Ein Jahr später soll der alte Eintrag wiederum fortgeführt werden. Dazu wird wiederum ein neuer Eintrag erstellt, der die neue Fortführung aufnimmt.

Bei weiteren Fortführungen bleiben alle alten Fortführungen weiterhin gültig. Es wird lediglich wieder eine neue Schicht auf den alten Eintrag plus Fortführungen aufgebracht.

Die letzte Fortführung wird als veraltet gekennzeichnet und mit einem Verweis auf die neue Fortführung versehen. Da die alte Fortführung einem Archivierungseintrag zugeordnet wurde, sind nun die Verwaltungsinformationen im Archiveintrag untergebracht. In den Verwaltungsinformationen der neuen Fortführung wird ein Verweis auf die letzte Fortführung hinzugefügt.

Anfang 2006 werden schließlich wiederum alle neuen Einträge Archiveinträgen zugeordnet. Die neue Fortführung befindet sich dann im Heiratsbuch 2005. Da sie im Jahr 2005 erstellt wurde. Das ist ein Unterschied zur bisherigen Verfahrensweise. In den Büchern wurden die Fortführungen bislang als Stempel zum Originaleintrag hinzugefügt. Nun müssen Fortführungen auch in späteren Personenstandsbüchern gesucht werden.

## 7.3 Anwendungsfälle der Sicherheitskomponente

Die Anwendungsfälle bilden die Grundlage für den Entwurf der Sicherheitskomponente. Nachdem alle Anforderungen an die Komponente ausgemacht wurden, flossen die Erkenntnisse in eine API ein, die im Anhang zu finden ist. Im Folgenden werden die erkannten Anwendungsfälle aufgelistet und danach in übersichtlicher Art und Weise im Detail präsentiert.

Die Anwendungsfälle lassen sich in drei Gruppen einordnen:

1. Anwendungsfälle zur Abdeckung des *Signaturkonzeptes* (Abbildung 7.13)
2. Anwendungsfälle zur Unterstützung der *Benutzerauthentifikation* (Abbildung 7.18)
3. Anwendungsfälle für die *Konfiguration* der Sicherheitskomponente (Abbildung 7.19)

### 7.3.1 Anwendungsfälle für das Signaturkonzept

Die Anwendungsfälle zum Signaturkonzept wurden sehr allgemein definiert, damit später nicht nur das aktuelle Signaturkonzept, sondern auch beliebige andere realisiert werden können.

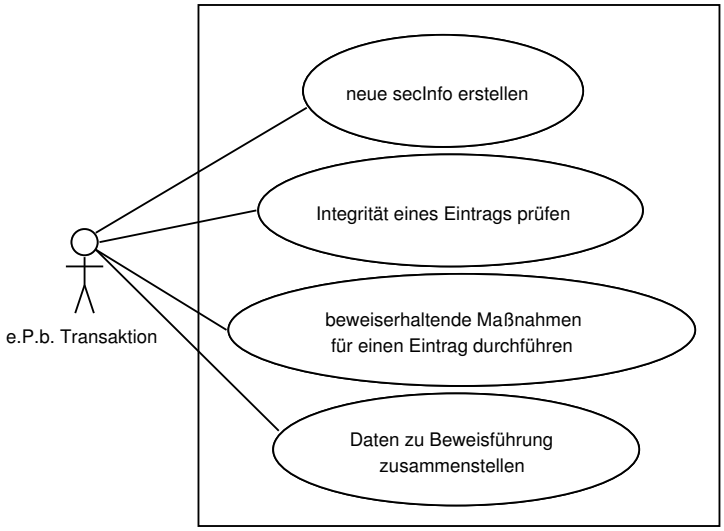


Abbildung 7.13: Die Anwendungsfälle der e.P.b. Sicherheitskomponente zur Unterstützung des Signaturkonzepts.

Folgende Anwendungsfälle wurden identifiziert:

- neue SecInfo erstellen (bei der Einlagerung eines neuen Eintrags)
- Integrität eines Eintrags prüfen
- beweiserhaltende Maßnahmen für einen Eintrag durchführen

- Daten zur Beweisführung zusammenstellen

Im Folgenden werden diese Anwendungsfälle genauer vorgestellt. Dabei wird vorgeführt, wie das konkrete aktuelle Signaturkonzept in den allgemeinen Anwendungsfällen umgesetzt werden kann. Die genaue Umsetzung der Anwendungsfälle befindet sich in der API-Beschreibung im Anhang.

#### 7.3.1.1 neue secInfo erstellen

**Kurzbeschreibung** Allgemein werden bei jedem Signaturkonzept Arbeiten nötig werden, wenn ein neuer Eintrag im Archiv abgelegt werden soll. Im aktuellen Signaturkonzept, müssen die Sicherheitsinformationen, zum Beweis der Rechtsgültigkeit einer Signatur und zum Erhalt dieser Beweisfähigkeit, erstellt werden.

Der Sicherheitskomponente müssen dazu die Rohdaten des neuen Eintrags, der Eintragsschlüssel, die Signatur und das Benutzerzertifikat übergeben werden. Diese prüft die Signatur und das Zertifikat. Dann wird ein neuer Zeitstempel und eine OCSP-Antwort angefordert. Die neuen Sicherheitsinformationen werden dann als neue Einträge in die Tabelle *secInfo* in der Datenbank abgelegt.

**Vorbedingung**

- Einstellungen der Sicherheitskomponente sind vollständig und korrekt.
- Rohdaten des Eintrags wurden übergeben.
- Eintragsschlüssel wurde übergeben
- Signatur wurde übergeben.
- Benutzerzertifikat wurde übergeben.
- Benutzerzertifikat ist bekannt und von einem vertrauenswürdigen Trustcenter unterschrieben. (Ein vertrauenswürdiges Trustcenter ist eines, dessen Zertifikat in der Tabelle *secCACert* eingetragen ist.)

**Nachbedingung**

- Signatur ist mathematisch korrekt und gültig. Sie entspricht den eingestellten Parametern.
- Alle Sicherheitsinformation wurden erstellt und zum Eintrag gespeichert.

**Fehlersituation**

- Signatur ist mathematisch nicht korrekt.
- Verwendetes Zertifikat ist ungültig oder unbekannt.
- Zugriff auf Persistenzkomponente liefert einen Fehler.
- Dienste des Trustcenters sind nicht erreichbar.

**Nachzustand im Fehlerfall**

- Es wurden keine Sicherheitsinformationen erstellt.

**Standardablauf** siehe Abbildung 7.14

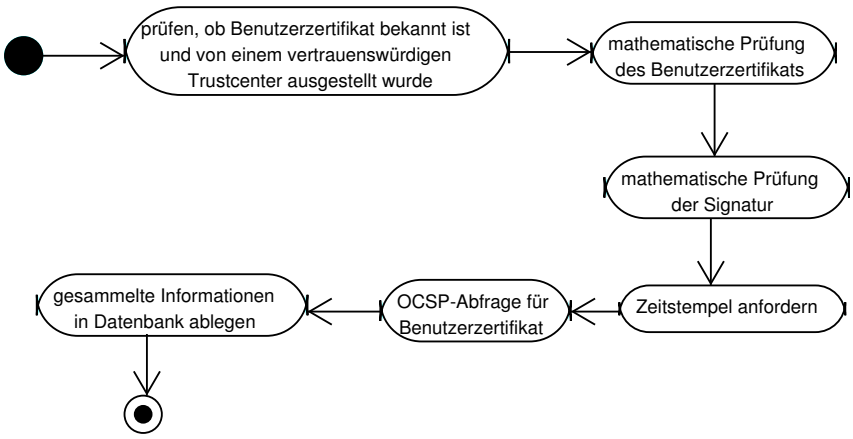


Abbildung 7.14: Der Standardablauf des Anwendungsfalls -neue secInfo erstellen-

#### 7.3.1.2 Integrität eines Eintrags prüfen

**Kurzbeschreibung** Eine wichtige Aufgabe der Sicherheitskomponente ist es, Einträge auf ihre Integrität hin zu untersuchen. Bei jedem Abruf eines Eintrags geschieht dieser Vorgang.

Beim vorliegenden Signaturkonzept wird die Gültigkeit des letzten Zeitstempels dieses Eintrags überprüft. Dazu wird der Sicherheitskomponente der Eintrag mit Eintragsschlüssel übergeben. Die Prüfung des Zeitstempels ist ein rein mathematischer Vorgang, da die Zertifikatskette des Trustcenters auf dem Server hinterlegt sein muss.

**Vorbedingung**

- Einstellungen der Sicherheitskomponente sind vollständig und korrekt.
- Eintrag und Eintragsschlüssel wurde übergeben.
- alle Sicherheitsinformationen zu diesem Eintrag sind vorhanden.

**Nachbedingung**

- Status quo ante.

**Fehlersituation**

- Zeitstempel ist mathematisch nicht korrekt.
- Verwendetes Zertifikat ist ungültig oder unbekannt.
- Zugriff auf Persistenzkomponente liefert einen Fehler.

**Nachzustand im Fehlerfall**

- Status quo ante.

**Standardablauf** siehe Abbildung 7.15

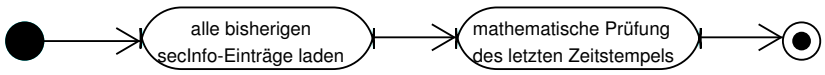


Abbildung 7.15: Der Standardablauf des Anwendungsfalls -Integrität eines Eintrags prüfen-

#### 7.3.1.3 Beweiserhaltende Maßnahmen für einen Eintrag durchführen

**Kurzbeschreibung** Nach aktuellem Kenntnisstand werden auch zukünftig regelmäßige Maßnahmen fällig werden, die die Rechtsgültigkeit der Einträge erhalten.

Im aktuellen Signaturkonzept müssen dazu regelmäßig neue Zeitstempel hinzugefügt werden. Der Sicherheitskomponente wird ein Eintrag mit Eintragsschlüssel übergeben. Diese prüft die Integrität des Eintrags und fügt einen neuen Zeitstempel an.

**Vorbedingung**

- Einstellungen der Sicherheitskomponente sind vollständig und korrekt.
- Eintrag und Eintragsschlüssel wurde übergeben.
- alle Sicherheitsinformationen zu diesem Eintrag sind vorhanden.

**Nachbedingung**

- Eintrag wurde mit einem neuen Zeitstempel versehen.

**Fehlersituation**

- Integrität des Eintrags ist nicht gegeben.
- Dienste des Trustcenters sind nicht erreichbar.
- Zugriff auf Persistenzkomponente liefert einen Fehler.

**Nachzustand im Fehlerfall**

- Es wurde kein neuer Zeitstempel erzeugt.

**Standardablauf** siehe Abbildung 7.16

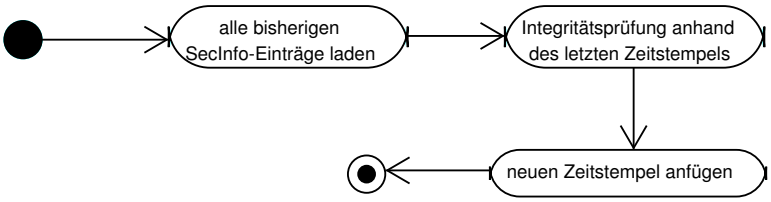


Abbildung 7.16: Der Standardablauf des Anwendungsfalls -beweißerhaltende Maßnahmen für einen Eintrag ergreifen-

#### 7.3.1.4 Daten zur Beweisführung zusammenstellen

**Kurzbeschreibung** Will ein Benutzer die Rechtsgültigkeit eines Eintrags prüfen, so benötigt dieser alle Validierungsinformationen zu diesem Eintrag.

Aktuell sind das alle Signaturen, Zeitstempel, Zertifikatsketten und OCSP-Antworten. Enthalten sind diese Informationen in den Sicherheitsinformationen der Einträge und in den Tabellen für die Zertifikate. Dazu muss die Sicherheitskomponente diese Informationen extrahieren und im gewünschten Format zurückliefern.

**Vorbedingung**

- Einstellungen der Sicherheitskomponente sind vollständig und korrekt.
- Eintrag und Eintragsschlüssel wurde übergeben.
- alle Sicherheitsinformationen zu diesem Eintrag sind vorhanden.

**Nachbedingung**

- Status quo ante.

**Fehlersituation**

- Zugriff auf Persistenzkomponente liefert einen Fehler.

**Nachzustand im Fehlerfall**

- Status quo ante.

**Standardablauf** siehe Abbildung 7.17

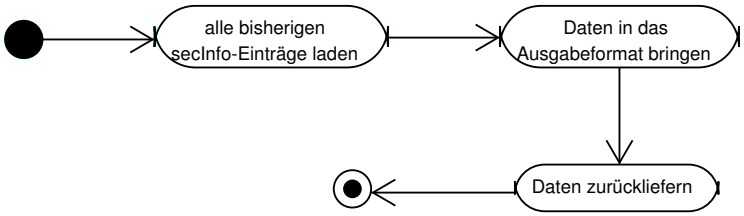


Abbildung 7.17: Der Standardablauf des Anwendungsfalls -Daten zur Beweisführung zusammenstellen-

### 7.3.2 Anwendungsfälle zur Unterstützung der Benutzerauthentifikation

Die Funktionen zur Unterstützung der Benutzerauthentifikation sind lediglich Hilfsfunktionen um andere Komponenten von kryptografischen Anforderungen zu entlasten. Außerdem kann damit eine einheitliche Konfiguration aller Kryptofunktionalität im e.P.b.-System erreicht werden. Die Verwaltung der Benutzerrechte geschieht nicht in der Sicherheitskomponente.

Die Anwendungsfälle wurden hier speziell für die erforderlichen Anforderungen ermittelt. Eine allgemeinere Festlegung ist nicht möglich. Sollte sich das System der Benutzeranmeldung ändern, können Anforderungen hinzukommen, die im Moment noch nicht erkennbar sind.

Folgende Anwendungsfälle wurden identifiziert:

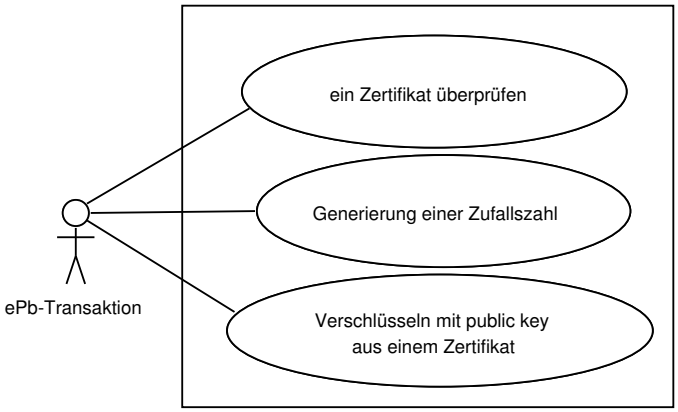


Abbildung 7.18: Anwendungsfälle für die Unterstützung der Benutzerauthentifikation

- ein Zertifikat prüfen (mathematisch und per OCSP)
- Generierung einer Zufallszahl
- Verschlüsselung mit public key aus einem Zertifikat

Um die Gültigkeit des Benutzerzertifikats zu überprüfen, muss zunächst die Zertifikatskette bis zu einem vertrauenswürdigen Trustcenter überprüft werden. Dies geschieht mathematisch und durch OCSP-Abfragen. Die vertrauenswürdigen Trustcenter werden bei der Konfiguration der Sicherheitskomponente eingetragen.

Die Generierung einer Zufallszahl wird für die Erstellung eines Sitzungstickets nötig. Damit kann das Ticket um eine unberechenbare Komponente erweitert werden, die Hackern einen Angriff erschwert. In späteren Authentifizierungskonzepten könnte sich diese Funktionalität ebenfalls als nützlich erweisen.

Das Sitzungsticket wird mit dem öffentlichen Schlüssel des Benutzers verschlüsselt. Kann der Benutzer das Ticket entschlüsseln, ist sichergestellt, dass der er den zum Zertifikat zugehörigen privaten Schlüssel kennt (POP).

Die Verschlüsselung und die Konfiguration für die OCSP-Abfragen legen dieselbe Konfiguration zugrunde, die auch das Signatursystem von e.P.b. verwendet. Mehr dazu im nächsten Abschnitt.

Auf eine detaillierte Auflistung der einzelnen Anwendungsfälle, wie im letzten Abschnitt, wird hier verzichtet. Es handelt sich dabei um relativ simple Wrapperfunktionen für die Kryptobibliothek von Java. Mehr zu diesen Funktionen befindet sich im Anhang bei der *API der Sicherheitskomponente*.

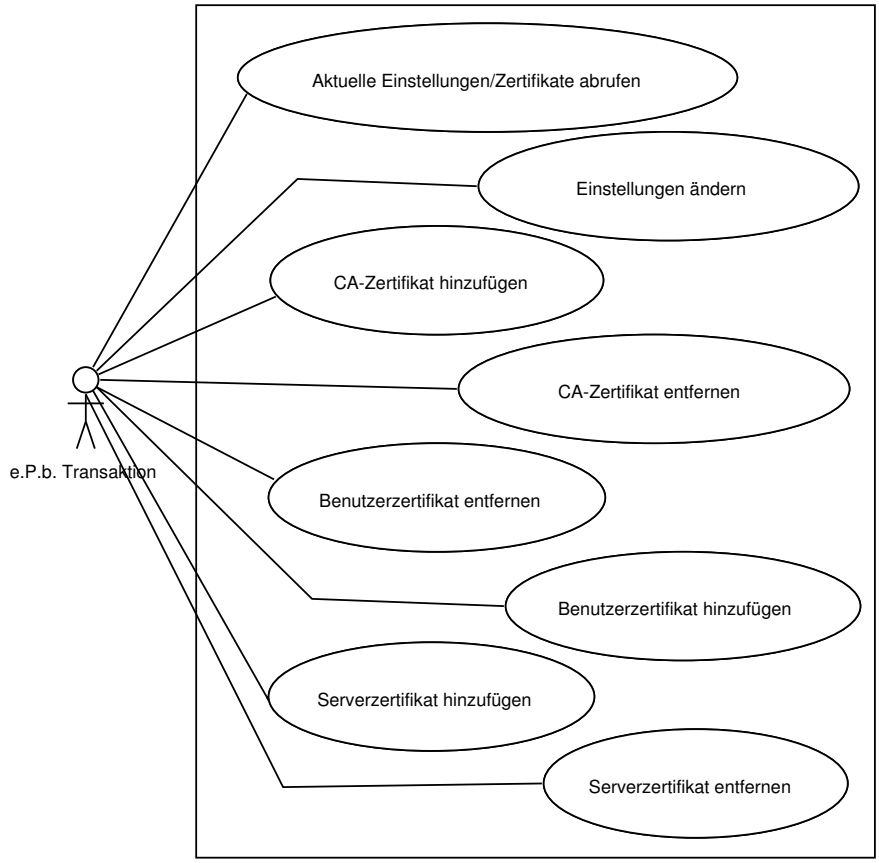


Abbildung 7.19: Anwendungsfälle für die Konfiguration der e.P.b.-Sicherheitskomponente

### 7.3.3 Anwendungsfälle zur Konfiguration der Sicherheitskomponente

Die Anwendungsfälle zur Konfiguration der Sicherheitskomponente dienen dazu, die Signaturparameter anzupassen. Im Laufe der Zeit müssen vor allem Schlüssellängen und Hashfunktionen regelmäßig auf den aktuellen Stand gebracht werden können.

Diese Anwendungsfälle können nicht unabhängig von dem zugrunde liegenden Signaturkonzept definiert werden. Hier müssen alle Einstellungen verändert werden können, die für den Betrieb des Systems entscheidend sind.

Dazu gehören die Zertifikate und Daten (Adressen der Zeitstempeldienste etc.) der als vertrauenswürdig eingetragenen Trustcenter. Diese müssen verwaltet werden können. Alle eingetragenen Benutzerzertifikate müssen von einem dieser Trustcenter ausgestellt worden sein.

Ebenso wie die Trustcenterzertifikate müssen auch die Benutzerzertifikate und die Serverzertifikate verwaltet werden können. Soll ein neues Zertifikat eingespeichert werden, erstellt die Sicherheitskomponente automatisch einen Verweis zu dem Zertifikat des ausstellenden Trustcenters.

Bei den User- und Trustcenterzertifikaten können nur solche entfernt werden, die noch nicht mit anderen Zertifikaten oder secInfo-Einträgen verknüpft sind. Damit ist sichergestellt, dass keine Zertifikate gelöscht werden können, die für die spätere Validierung von Signaturen benötigt werden.

Neben den Zertifikaten müssen weiter Parameter spezifiziert werden. Schließlich kann auch die Eignung der eingesetzten Algorithmen für die digitale Signatur über die Jahre verloren gehen. Deshalb werden auch hier immer wieder Anpassungen nötig sein. Die aktuellen Einstellungen können deshalb abgerufen und editiert werden.

Alle Änderungen dürfen nur von autorisierten Administratoren durchgeführt werden. Es ist dabei nicht Aufgabe der Sicherheitskomponente die Zugriffsberechtigung zu überprüfen.

Diese Anwendungsfälle werden in einer eigenen *Konfigurations-API* umgesetzt. Mehr dazu im Anhang.

## 7.4 Die statische Struktur

Die statische Struktur der Sicherheitskomponente stellt den Rahmen für die Realisierung der ermittelten Anwendungsfälle. Im Mittelpunkt des Klassendiagramms (siehe Abbildung 7.20) steht die Klasse `SecComponent`. Diese beinhaltet die Implementierung des aktuellen Signaturkonzepts, der Hilfsfunktionen zur Authentifikation und Methoden zur Konfiguration.

Die drei hellgrau eingefärbten Interfaces `SecSignature`, `SecAuth` und `SecConfig` gewährleisten stabile Schnittstellen auch bei einer Veränderung des Signaturkonzepts. In diesem Falle kann eine weitere Klasse entwickelt werden, die die Schnittstellen in `SecSignature` implementiert. Die Transaktionen müssen dann die neue Klasse instanziieren, rufen aber weiterhin die in `SecSignature` deklarierten Methoden auf.

Das dunkelgrau eingefärbte Interface `PersistenceInterface` ist die Schnittstelle zur Persistenzkomponente. Über die hier deklarierten Methoden greifen die Objekte der Sicherheitskomponente auf die Datenbank zu. Aus Gründen der Übersichtlichkeit wurden die Methoden ausgeblendet. Mehr dazu befindet sich bei der Implementierung eines Prototypen in einem der nächsten Kapitel.

Die Zuständigkeiten der drei Interfaces im Einzelnen (die genauen API-Methoden sind im Anhang der Arbeit nachzulesen):

- `SecSignature` beinhaltet eine Methode pro spezifiziertem Anwendungsfall für das Signaturkonzept (neue SecInfo erstellen, Integrität eines Eintrags prüfen, beweiserhaltende Maßnahmen für einen Eintrag durchführen, Daten zur Beweisführung zusammenstellen).
- `SecAuth` deklariert drei Methoden zur Unterstützung der Benutzerauthentifikation: Überprüfung eines Zertifikats, Generierung einer Zufallszahl und Verschlüsselung mit dem public key aus einem Zertifikat.
- `SecConfig` deklariert Methoden zur Konfiguration der Sicherheitskomponente. Die aktuelle, sowie frühere Konfiguration kann abgerufen werden, neue Einstellungen können abgespeichert werden und die Liste der Trustcenter-, User- und Serverzertifikate können verwaltet werden.

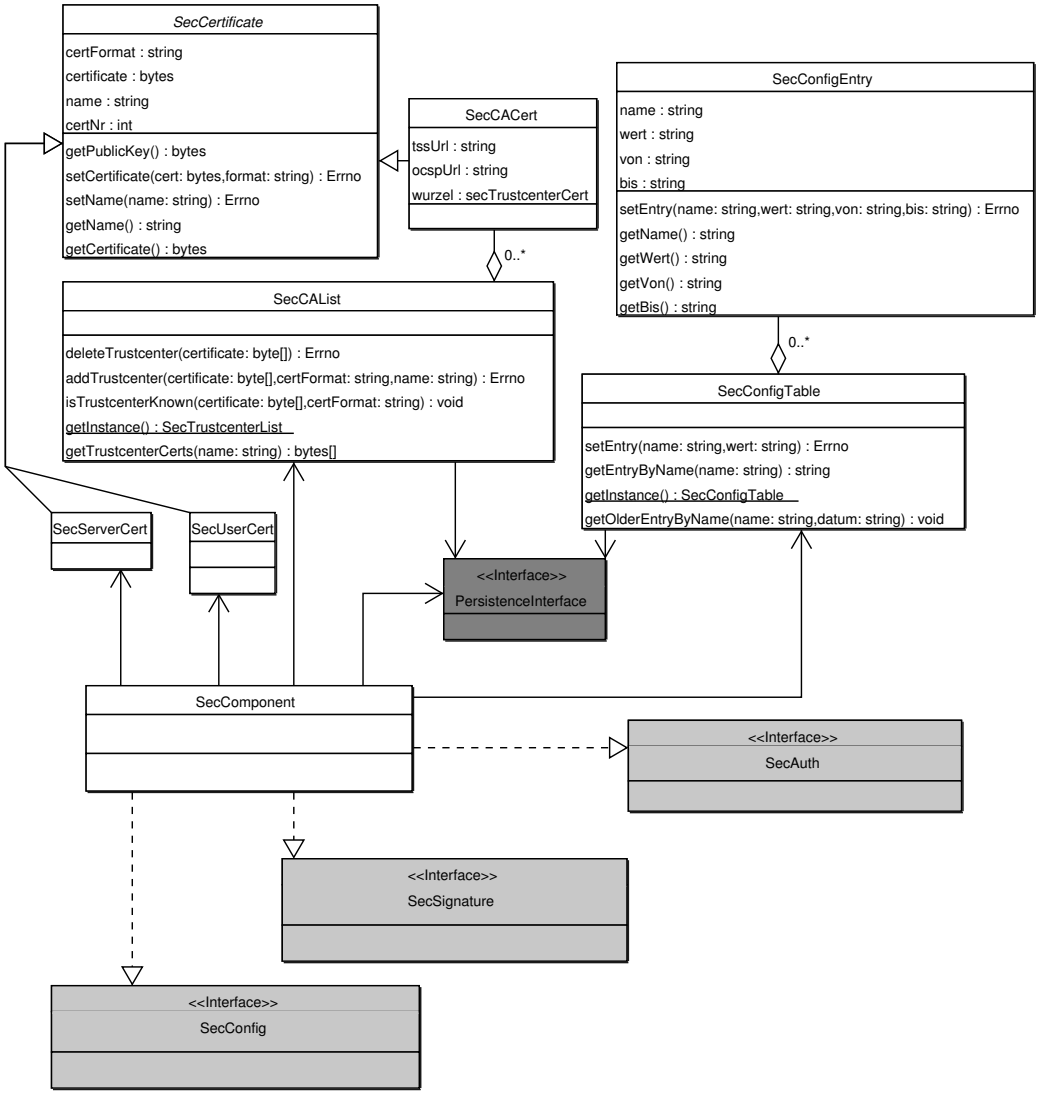


Abbildung 7.20: Das Klassendiagramm der Sicherheitskomponente.

Die persistenten Daten zur Konfiguration (Zertifikate und Einstellungen) werden mittels zweier Klassen verwaltet. `SecCAList` und `SecConfigTable`. Beide sind als Singleton konzipiert. Damit kann es zu einem Zeitpunkt jeweils nur ein Objekt dieser Klassen geben. Dazu wurde ihr Konstruktor als private deklariert. Mit der statischen Methode `getInstance()` kann eine Referenz auf die jeweilige Instanz angefordert werden.

Dies hat zwei Vorteile: Erstens wird Speicherplatz eingespart, da die Daten über Zertifikate und Konfiguration nur einmal im Arbeitsspeicher abgelegt werden. Zweitens kann die Konfliktlösung bei gleichzeitigem Zugriff auf die persistenten Informationen an einem zentralen Ort erfolgen.

`SecConfigTable` wird wie ein Hashtable angesprochen. Die Einstellungen werden als Name - Wert - Paare in der Datenbank abgelegt. `SecConfigTable` verwendet dabei Objekte der Klasse `SecConfigEntry`, die jeweils ein Name - Wert - Paar abbilden.

Über den Namen kann in `SecConfigTable` der Wert abgerufen werden. Dabei wird der aktuellste Wert geliefert. Eine weitere Methode erlaubt die Abfrage der Konfiguration zu einem früheren Zeitpunkt. Wird ein neuer Wert abgelegt, unter dessen Namen bereits ein früherer Wert existiert, so wird der alte Wert aufbewahrt und der neue Wert als aktuelle Konfiguration eingetragen.

Die abstrakte Oberklasse `SecCertificate` deklariert Methoden zum Zugriff auf die Daten eines Zertifikats. Objekte der Unterklasse `SecUserCert` dienen zur Aufnahme von Benutzerzertifikaten, die in `SecComponent` verwendet werden. Objekte der Klasse `SecServerCert` nehmen die Zertifikate externer e.P.b.-Server auf.

Objekte von `SecCACert` nehmen Trustcenterzertifikate auf. Eine Liste dieser Zertifikate wird zur Verwaltung der Trustcenter in `secCAList` angelegt.

Das vorliegende Klassendiagramm soll nur einen Überblick vermitteln. Es fehlen die Exceptions und einige Methoden. Diese müssen in einer später Phase deklariert werden. Die Beschreibung der API im Anhang dieser Arbeit zeigt alle Exceptions und stellt die Methoden der Interfaces vor.

# Teil IV

# Realisierung des Konzeptes

# 8. Java spezifische Kryptoarchitekturen

Die Realisierung der geplanten Funktionalität erfordert unter anderem den Einsatz kryptografischer Funktionen, die Verwaltung von Zertifikaten und den Aufruf externer Dienste. Für fast alle diese Anforderungen bietet das Java SDK 1.4 eine fertige Lösung an. In den folgenden Abschnitten werden die in Java mitgelieferten APIs und Komponenten vorgestellt.

Bietet das Java SDK für eine Kryptomethode keine fertige Lösung, werden weitere Möglichkeiten der Realisierung erörtert. Dies können Java-Erweiterungen von Drittherstellern oder der Verweis auf die zu implementierende RFC sein.

## 8.1 Java Cryptography Architecture

Dieses Kapitel referiert die Inhalte aus der Java Cryptography Architecture (*JCA*) API Specification & Reference [17] und aus dem Java Cryptography Extension (JCE) - Reference Guide [18].

Die JCA stellt das Grundgerüst aller sicherheitsrelevanten APIs in Java. Es besteht aus den Klassen in `java.security` und den zugehörigen Unterklassen. Neben APIs für die Bereitstellung der kryptografischen Funktionalität wird auch eine API für Zertifikatsmanagement geboten. Die eigentlichen APIs für Verschlüsselung, Schlüsselaustausch und Signatur sind in der JCE enthalten, die die JCA um diese Funktionen erweitert.

JCA und JCE arbeiten mit einer Provider-Architektur. Damit wird die eigentliche Implementierung der Funktionen austauschbar. Provider von Drittherstellern können dynamisch eingebunden werden und stellen von da an eine Implementierung zur Verfügung, die über die JCA/JCE -API angesprochen wird.

Es gehört dabei zu den Prinzipien von JCA/JCE, dass die Implementierungen zwar austauschbar sind, diese aber Ergebnisse produzieren, die auch mit anderen Providern weiterverarbeitet werden können. So muss sich eine mit Provider A erstellte Signatur unter Provider B überprüfen lassen können.

Der im Java SDK mitgelieferte Provider stammt von *SUN* und bietet Unterstützung für alle gängigen Kryptoalgorithmen. Lediglich die Schlüssellänge ist, wegen den Exportrestriktionen der USA, stark begrenzt. Abhilfe schafft die Installation der unbeschränkten Nutzungs-Policy von SUN.

Alternativen zum SUN-Provider gibt es einige. Herauszuheben ist hier das OpenSource-Projekt *BouncyCastle*[1]. Es ist kostenlos verfügbar und wird durch eine große Community unterstützt. Der Provider von BouncyCastle unterstützt auch große Schlüssellängen. Neben dem JCE-Provider enthält das BouncyCastle-Paket auch eine alternative API und Unterstützung für das OCSP-Protokoll. Mehr dazu in einem der nächsten Abschnitte.

Die JCA spezifiziert grobkörnige Methoden, wie die Bildung einer Signatur. Bei der JCE handelt es sich um feinkörnigere Funktionen, wie Verschlüsselung oder Bildung eines Hashwerts.

[1] `http://www.bouncycastle.org/` [Stand: 31.07.2004]

Dabei gibt die Architektur nur Schnittstellen für ganze Typen von Kryptooperationen vor. Spezielle Algorithmen werden nicht vorausgesetzt. Am Beispiel der Signatur bedeutet das: Es gibt eine Schnittstelle zum Erstellen und zum Überprüfen einer digitalen Signatur. Mit welchen Algorithmen diese erstellt werden kann, hängt vom installierten Provider ab und ist zur Laufzeit wählbar.

Diese Funktionalitäten wird über so genannte „Engine-Classes“ ermöglicht. Einige Beispiele: Signature, KeyPairGenerator, CertPathValidator (eine vollständige Liste befindet sich in der Literatur).

Jede dieser Engine-Classes besitzt die Fabrikmethode getInstance(...). Der Aufruf dieser statischen Methode liefert ein Objekt der Klasse zurück, mit dem sich tatsächlich die gewünschte Operation durchführen lässt. Dazu muss der Methode meist der gewünschte Algorithmus und (optional) der zu verwendende Krypto-Provider mit der Implementierung übergeben werden. Wird kein Provider spezifiziert verwendet Java die per Default gewählte Implementierung.

Am Beispiel einer Signatur:

```
Signature sig = Signature.getInstance(„SHA-1withRSA“);
```
[2]

Damit erhält man ein Signaturobjekt sig, welches Signaturen erstellen und verifizieren kann, die als Hashalgorithmus SHA-1 und zur Verschlüsselung RSA einsetzen. Nun muss gewählt werden, ob entweder eine Signatur erstellt, oder eine Signatur überprüft werden soll. In diesem Beispiel soll eine Signatur erstellt werden.

```
sig.initSig(privateKey pk);
```

Damit ist das Objekt nun für die Erstellung einer Signatur mit dem privaten Schlüssel pk vorbereitet. Jetzt werden die zu unterschreibenden Daten übermittelt:

```
sig.update(byte[] data);
```

Diese Funktion wird so oft aufgerufen, bis alle Daten vollständig übergeben wurden. Nun kann die Signatur durchgeführt werden:

```
byte[] sigDat = sig.sign();
```

sigDat enthält jetzt die Signatur. Das Format ist abhängig von dem gewählten Algorithmus. Bei RSA handelt es sich um einen *PKCS #1* - Block nach *DER*-Kodierung[3].

Neben den Engine-Classes sind, wie schon im Beispiel zu sehen, Hilfsklassen vorhanden, die z.B. Schlüssel oder Zertifikate aufnehmen. (privateKey, publicKey, Certificate, X509Certificate,...) Diese werden als Parameter für die Kryptomethoden verwendet.

Soviel zu dem allgemeinen Konzept der Java Cryptography Architecture, wo fast alle Methodenaufrufe zum Durchführen einer Kryptooperation nach dem vorgestellten Schema ablaufen. Alle weiteren Methoden und verfügbaren Algorithmen sind in der Literatur nachzulesen.

## 8.2 Java Secure Socket Extension

Dieser Abschnitt referiert die Inhalte aus dem JSSE - Reference Guide[2].

[2] SHA-1withRSA wäre der String, der in der Datenbank als der verwendete Algorithmus einer Signatur abgelegt würde. (siehe weiter oben: Die Datenstruktur für Personenstandseinträge)

[3] mehr dazu in: RSA Laboratory, Public Key Cryptography Standards Note #1

Die Java Secure Socket Extension (*JSSE*) baut auf der JCA auf und verwendet dieselbe Provider-Architektur. Der mitgelieferte Provider von SUN beinhaltet eine umfassende Implementierung von SSL/TLS, die für e.P.b. keine Wünsche offen lässt. So ist neben der obligatorischen Server-Authentifizierung auch eine Client-Authentifizierung möglich, wie sie für das e.P.B. vorgesehen ist.

Hier ein kurzes Beispiel für den Aufbau eines SSL-Sockets:

```
SSLSocketFactory sslFactory =
         (SSLSocketFactory) SSLSocketFactory.getDefault();

SSLSocket socket =
         (SSLSocket) sslFactory.createSocket(adr, port);

OutputStream out = socket.getOutputStream();
InputStream in = socket.getInputStream();

[... Lesen + Schreiben ...]

socket.close();
```

Zuerst wird eine Instanz von `SSLSocketFactory` angefordert. Diese ist dabei mit den Defaulteinstellungen konfiguriert. Die Fabrikmethode `createSocket()` liefert ein `SSLSocket`-Objekt zurück, dass von da an, wie ein „normaler“ `Socket` angesprochen werden kann.

Auf eine umfassende Vorstellung der Möglichkeiten von JSSE wird hier verzichtet. Sie ist in der angegebenen Literatur nachzulesen. Mehr dazu befindet sich auch im Kapitel *SSL mit Tomcat*. Dort wird ein Code-Beispiel mit gegenseitiger Kommunikation vorgestellt, wie es für das e.P.b. benötigt wird.

## 8.3 Keystore / Truststore

Java enthält von Haus aus die Möglichkeit Zertifikate und private Schlüssel zu speichern und vor unbefugtem Zugriff zu schützen. Dies sind die Aufgaben von Keystores und Truststores.

Dabei unterscheiden sich beide in der Bedeutung der in ihnen gespeicherten Einträge. Ein Truststore enthält alle Zertifikate, denen der User vertraut. Das kann ein Zertifikat eines Bekannten sein, am häufigsten wird es sich aber hierbei um das Zertifikat seines Trustcenters handeln.

Ob ein Zertifikat vertrauenswürdig ist, wird dann folgendermaßen entschieden: Zuerst wird die gesamte Zertifikatskette des zu prüfenden Zertifikats bis zu der ausstellenden Instanz gebildet. Befindet sich ein Zertifikat dieser Kette im Truststore, gilt das geprüfte Zertifikat als vertrauenswürdig.

Ein Keystore nimmt neben Zertifikaten auch die zugehörigen privaten Schlüssel auf. Dies sind die Zertifikate des Benutzers. Sie werden demnach nicht zur Überprüfung anderer Zertifikate verwendet, sondern dienen zum Signieren und Verschlüsseln.

Java legt Keystores und Truststores in speziellen Formaten (z.B. „JKS“ - Java Key Store) als Datei ab. Das Format beinhaltet eine Verschlüsselung des Inhalts und die Möglichkeit den Zugriff per Passwort zu schützen. Um den Zugriff auf einen bestimmten Eintrag zu ermöglichen, wird für jeden Eintrag ein Alias-Name vergeben.

In Wirklichkeit muss es sich bei einem Keystore und einem Truststore nicht um verschiedene Dateien handeln. Zertifikate können mit oder ohne privaten Schlüssel in eine Datei gespeichert werden.

Dem Java SDK liegt das Programm *keytool* bei, mit dem die Keystores von der Kommandozeile aus, verwaltet werden können.

Ein Beispiel für die Nutzung von *keytool* und die Java-API für Keystores, folgt im nächsten Kapitel bei der Einrichtung eines Keystores für die Authentifikation gegenüber einem SSL-Webserver.

Weitere Informationen zu *keytool* in der zugehörigen Referenz[4]. Die Beschreibung der Keystore-API befindet sich in[5].

## 8.4 fehlende Funktionalität

Zwei Funktionen, die für das e.P.b. essenziell wichtig sind, finden im aktuellen Java SDK 1.4 keine Unterstützung. Dabei handelt es sich um OCSP-Abfragen und Zeitstempel.

Mit dem Java SDK 1.5, das sich zurzeit noch in der Betaphase befindet (Stand: 03.08.2004), wird die Unterstützung von OCSP zur Überprüfung des Status eines Zertifikats Einzug halten.

Bis dahin muss auf andere Bibliotheken zurückgegriffen werden. BouncyCastle, dessen JCE-Provider schon vorgestellt wurde, bietet daneben eine API, die auch OCSP-Abfragen durchführen kann. Solange keine Unterstützung dieser Funktionalität in Java gegeben ist, bleibt die Bibliothek von BouncyCastle eine Empfehlung.

Schlechter sieht die Situation in Bezug auf Zeitstempel aus. Mir ist keine freie Bibliothek bekannt, die eine Abfrage von Zeitstempeln ermöglicht. Auch ist für absehbare Zeit keine Unterstützung in einem Java SDK vorgesehen. Einige kommerzielle Produkte könnten die Unterstützung bieten. Aufschluss darüber schafft nur eine Anfrage bei dem jeweiligen Hersteller.

Hier bleibt nur der Verweis auf die RFC für Zeitstempel [21]. Solange es keine freie Bibliothek für die Abfrage von Zeitstempeln gibt, muss das RFC in der eigenen Software umgesetzt werden.

[4] `http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html` [Stand: 03.08.2004]

[5] `http://java.sun.com/j2se/1.4.2/docs/api/java/security/KeyStore.html` [Stand: 03.08.2004]

# 9. SSL mit Tomcat

Um die erwartete Sicherheit bei der Kommunikation zu erreichen, ist eine Konfiguration des Webservers nötig. Dieser Abschnitt erläutert diese Konfiguration anhand des verbreiteten Webservers Tomcat. Außerdem wird an einem kleinen Beispiel der sichere Zugriff auf ein Servlet aus einem Java-Client heraus demonstriert.

Bevor hier die tatsächliche Konfiguration besprochen wird, zunächst noch einmal die gewünschten Ziele:

- Verwendung des SSL-Protokolls (besonders https)
- Kommunikation mit starker Verschlüsselung
- gegenseitige Server-/ Client-Authentifikation

Zuerst einige Worte zur gegenseitigen Authentifikation der beteiligten Rechner. Wie schon im Kapitel über die Kommunikationssicherheit erwähnt, wird hier eine Lösung mit einfachen Zertifikaten angestrebt, die keine besondere Infrastruktur (PKI) erfordert.

Dabei liegen Zertifikate und private Schlüssel auf der Festplatte des Benutzers. Der private Schlüssel wird durch ein Passwort geschützt. Nach einmaliger Eingabe dieses Passworts kann die SSL-Implementierung beliebig oft auf diesen Schlüssel zugreifen.

Wird dieser private Schlüssel kopiert und das zugehörige Passwort ausgespäht kann ein Angreifer die Kommunikation des Benutzers mit dem e.P.b. abhören. Will ein Angreifer selbst auf das e.P.b. zugreifen, benötigt er, zum Login, eine qualifizierte Signatur per Smartcard, was eine hohe Sicherheit gegen Missbrauch bietet.

Der Schutz dieser privaten Schlüssel vor dem Kopieren im lokalen Netzwerk liegt nicht im Verantwortungsbereich des e.P.b., sondern erfordert eine sichere Netzinfrastruktur.

Das hier vorgestellte Beispiel arbeitet aus Gründen der Einfachheit mit selbst unterschriebenen Zertifikaten. Das sind Zertifikate, die mit dem zugehörigen privaten Schlüssel selbst unterschrieben sind. Für die Anwendung im lokalen Netzwerk können solche Zertifikate ausreichen.

Die Alternative wäre die Einrichtung eines eigenen „Trustcenters“ mit frei verfügbaren Programmen, wie OpenSSL. Mit diesem Tool kann ein Zertifikat erzeugt werden, mit dem dann alle ausgestellten Zertifikate unterschrieben werden. Dies ermöglicht eine flexiblere Verwaltung der Benutzer.

Vorstellbar wäre auch eine Verwendung von Smartcards, bei denen der private Schlüssel auf der Karte gespeichert ist und diese niemals verlässt. Es ist aber durchaus möglich, dass das SSL-Protokoll öfter auf die Karte zugreifen muss. Dabei muss der Zugriff auch ohne ständige Passworteingabe möglich sein. Eine solche Karte lag mir bei der Erstellung dieser Arbeit nicht vor, weshalb hier die einfachere Lösung vorgestellt wird.

Tomcat ermöglicht auch mit selbst unterschriebenen Zertifikaten eine sichere Authentifikation. Dazu wird ein Keystore angelegt, der alle für alle zugriffsberechtigten User ein Zertifikat enthält. Dies ist das gleiche Zertifikat, dass dieser Benutzer auf seinem Rechner vorhält und über dessen privaten Schlüssel er verfügt.

Soll einem Benutzer die Zugriffsberechtigung entzogen werden, so wird dessen Zertifikat einfach aus dem User-Keystore entfernt. Somit muss auch der Keystore auf dem Server vor unberechtigten Zugriff geschützt werden. Auch hier ist das wieder eine Frage der Netzwerksicherheit.

Dem JavaSDK liegt das Programm `keytool` bei, mit dessen Hilfe Keystores verwaltet werden können. Die folgenden Befehle erzeugen ein selbst unterschriebenes Zertifikat, welches den Tomcatserver identifizieren soll:

```
keytool -genkey -alias tomcat -keystore tomcat.ks
-sigalg "SHA1withRSA" -keyalg "RSA" -keysize 2048
```

```
keytool -selfcert -alias tomcat -keystore tomcat.ks
-sigalg "SHA1withRSA"
```

Damit wurde eine Keystore-Datei `tomcat.ks` erzeugt, die das Zertifikat für den Tomcatserver und den zugehörigen privaten Schlüssel enthält. Den Clients muss dieses Zertifikat bekannt gemacht werden. Dazu wird es aus dem Tomcat Keystore exportiert und in den lokalen Keystore des Clients importiert.

```
keytool -export -alias tomcat -keystore tomcat.ks -file out.zert
```

```
keytool -import -alias tomcat -file out.zert -keystore myKS.ks
```

Das Zertifikat wurde zunächst in die Datei `out.zert` exportiert und dann in den Keystore `myKS.ks` importiert. Der Keystore `myKS.ks` liegt in diesem Beispiel auf der Client-Workstation. Allen dort enthaltenen Zertifikaten wird der Client später als Server vertrauen.

Nun ist das Serverzertifikat beim Client bekannt. Um auch die Authentifizierung in umgekehrter Richtung zu ermöglichen, muss nun ein Clientzertifikat erstellt werden. Hier also im Keystore `myKS.ks`. Dieses wird wiederum exportiert und in einen Keystore auf dem Server importiert.

Für die Benutzerzertifikate auf dem Server wird dabei ein eigener Keystore erstellt. Hier wird dieser `user.ks` genannt. Die Befehle sind dabei ähnlich wie bei der Erstellung des Serverzertifikats.

Nun liegen auf dem Server die beiden Keystores `tomcat.ks` (mit dem Serverzertifikat) und `user.ks` (mit den Benutzerzertifikaten). Auf dem Client liegt der Keystore `myKS.ks` (mit dem Serverzertifikat und dem eigenen Benutzerzertifikat).

Der nächste Schritt ist, Tomcat so zu konfigurieren, dass die gerade erstellten Keystores verwendet werden. Dies geschieht durch Editieren der Konfigurationsdatei `conf/server.xml`. Dort existiert ein auskommentierter Eintrag für SSL, der folgendermaßen angepasst werden muss:

```
<Connector port="8443" maxThreads="150" minSpareThreads="25"
maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="100" debug="0"
scheme="https" secure="true" clientAuth="true" sslProtocol="TLS"
keystoreFile="tomcat.ks" keystorePass="123456"
truststoreFile="user.ing" truststorePass="123456"/>
```

Damit wird der Server angewiesen, auf Port 8443 per https SSL-Verbindungen zu akzeptieren. Dabei wird eine Clientauthentifizierung erzwungen (`clientAuth = true`). Das Attribut `keystoreFile` verweist auf den Keystore mit dem Serverzertifikat. `truststoreFile` verweist auf den Keystore mit den Zertifikaten aller zugriffsberechtigten Benutzer.

Bei `keystorePass` und `truststorePass` müssen die Passwörter eingetragen werden, die bei der Erstellung der Keystores gewählt wurden. Zu beachten ist hierbei, dass ein und dasselbe Passwort für den Zugriff auf den Keystore, wie für den Zugriff auf den privaten Schlüssel gewählt werden muss.

Nun ist der Server komplett konfiguriert und ermöglicht die Kommunikation per SSL mit gegenseitiger Authentifizierung. Um die Erstellung einer SSL-Verbindung zu demonstrieren nun Auszüge aus einem Java-Client. Der komplette Quellcode befindet sich auf der beigefügten CD.

```
[...]

public class SSLClient extends Thread{

  SSLSocket s;
  public SSLClient() { }

  public void run(){
    [...]

    SSLSocketFactory sslFact;
    SSLContext ctx;
    KeyManagerFactory kmf;
    KeyStore ks;
    TrustManagerFactory tmf;


    //SSL-Kontext erstellen und mit Keystore verknuepfen
    ctx = SSLContext.getInstance("TLS");
    kmf = KeyManagerFactory.getInstance("SunX509");
    ks=KeyStore.getInstance("JKS");

    InputStream inStream = new FileInputStream("myKS.ks");
    ks.load(inStream,"123456".toCharArray());

    kmf.init(ks, "123456".toCharArray());
    tmf=TrustManagerFactory.getInstance("SunX509");
    tmf.init(ks);

    ctx.init(kmf.getKeyManagers(), tmf.getTrustManagers(),
             null);
    sslFact = ctx.getSocketFactory();
```

```
    [...]

    //HTTPS-Verbindung aufbauen
    HttpsURLConnection https;
    URL url = new URL("https","127.0.0.1",8443,
                      "/servlets-examples/servlet/SSLTest");
    https = (HttpsURLConnection) url.openConnection();
    [...]
    https.setSSLSocketFactory(sslFact);
    [...]
    https.connect();
    [...]

    //Daten lesen
    byte[] buf=new byte[1024];
    int len= in.read(buf);

    do{
      System.out.println(new String(buf,0,len,"UTF-8"));
      len= in.read(buf);
    }while(len > 0);
    [...]
    HttpsUrlConnection.close();
    [...]

  }

}
```

Der Client stellt eine SSL-Verbindung zum Webserver her und ruft das Servlet SSLTest auf. Dieses befindet sich ebenfalls auf der beigelegten CD. Dieses übermittelt lediglich einen Text.

Der Keystore myKS.ks wird geladen und mit ihm jeweils ein KeyManager und ein TrustManager verknüpft. Beide werden der SSL-Implementierung übergeben.

Damit verfügt die SSL-Implementierung über die notwendigen Daten zur Authentifizierung. Per `HttpsURLConnection` wird dann eine Verbindung zum Server aufgebaut. Am Ende wird die Antwort des Servers ausgelesen und die Verbindung getrennt.

# 10. Smartcard unter Java

In diesem Kapitel wird kein kompletter Prototyp vorgestellt. Im Laufe meiner Diplomarbeit habe ich mit verschiedenen Bibliotheken zur Implementierung der gewünschten Funktionalität experimentiert.

Die Herstellung einer SSL-Verbindung habe ich schon im Kapitel *SSL mit Tomcat* präsentiert. Hier stelle ich nun meine Erfahrungen mit der Ansteuerung einer Signatur-Smartcard aus Java heraus vor.

## 10.1 Signaturumgebung

Für die Entwicklungsphase von e.P.b. wurden uns vom Trustcenter der Deutschen Post „*SignTrust*“ Signaturkarten mit zugehörigen Zertifikaten nach SigG und Kartenleser zur Verfügung gestellt.

Dies ermöglichte uns mit der digitalen Signatur inklusive Zugriff auf OCSP- und Zeitstempeldienste zu experimentieren. Vor allem der Zugriff aus Java heraus war für diese Arbeit von Belang.

An dieser Stelle sei jedoch darauf hingewiesen, dass ein e.P.b. keinesfalls auf ein bestimmtes Trustcenter festgelegt sein wird. Durch die kompatiblen Schnittstellen werden Abfragen auf beliebige Trustcenter möglich sein.

## 10.2 Smartcard aus Java ansteuern

Von sich aus bietet das JavaSDK keine API für Smartcards an. Hier muss auf die Lösungen von Drittherstellern zurückgegriffen werden.

Die erste Bibliothek, die ich untersuchte, war das *OpenCard Framework* von IBM[1]. Es handelt sich hierbei um eine Architektur, die den Zugriff auf Smartcards über eine standardisierte API ermöglicht.

Das dahinter stehende OpenCard Konsortium liest sich wie ein Who-is-Who im Chipkartenbereich:

- American Express Travel Related Services
- Gemplus
- IBM
- Toshiba Corporation

[1] `http://www.opencard.org/` [Stand: 09.08.2004]

- TOWITOKO
- Schlumberger
- Siemens
- Sun Microsystems
- UbiQ Inc.
- Visa International
- und einige andere ...

Das ehrgeizige Ziel sollte dadurch erreicht werden, dass die teilnehmenden Hersteller von Karten und Kartenterminals jeweils Treiber für die OpenCard Architektur schreiben.

Leider musste ich im Laufe meiner Experimente feststellen, dass das Projekt seit 1999 kaum weiterverfolgt wurde. Die Referenzimplementierung bietet rudimentären Zugriff auf Betriebssystemebene und die Unterstützung einiger älterer Kartentypen an.

Es existieren nur wenige Treiber von Kartenherstellern. So war es mir nicht möglich die Signaturkarte von SignTrust zur Zusammenarbeit über die Signatur-API von OpenCard zu bewegen.

Lediglich der Zugriff auf Betriebssystemebene war möglich. Die Ansteuerung einer Karte auf dieser Ebene bietet Stoff für eine komplette Diplomarbeit. Deshalb suchte ich weiter nach Alternativen.

Nach einiger Recherche bei den frei verfügbaren Bibliotheken stieß ich auf *SecSigner* von SecCommerce[2]. SecSigner ist eine Anwendungskomponente in Java, die eine vollständige Abwicklung der Signatur ermöglicht.

Sie lässt sich einfach in ein Java Programm einbinden und präsentiert sich beim Aufruf mit einer eigenen grafischen Benutzeroberfläche. Die Software unterstützt eine große Anzahl an Kartenlesern und Signaturkarten.

Neben der Erstellung digitaler Signaturen wird auch die Überprüfung einer Signatur, inklusive OCSP-Abfragen und das Einholen von Zeitstempeln angeboten.

Die Software ist akkreditiert nach ITSEC E2/HOCH für SigG-konforme Signaturen. Es ist also davon auszugehen, dass sie höchsten Ansprüchen entspricht und keine Sicherheitslöcher aufweist.

Die Lizenz erlaubt eine unbeschränkte Verwendung der Software, auch in kommerziellen Produkten, solange keine Änderungen an der Software vorgenommen werden[3].

Bisher (Stand bei Abgabe der Diplomarbeit) ermöglichte SecSigner jedoch nur die rechtsgültige Signatur von Text oder HTML-Dokumenten. Bei PDF-Dokumenten zeigt die Software den PDF-Quelltext an.

Ein kurzes Beispiel für die Signatur mit SecSigner:

[2] `http://www.seccommerce.de/` [Stand: 12.08.2004]

[3] `http://www.seccommerce.de/de/produkte/webcontrust/secsigner/secsigner_eula.html` [Stand: 12.08.2004]

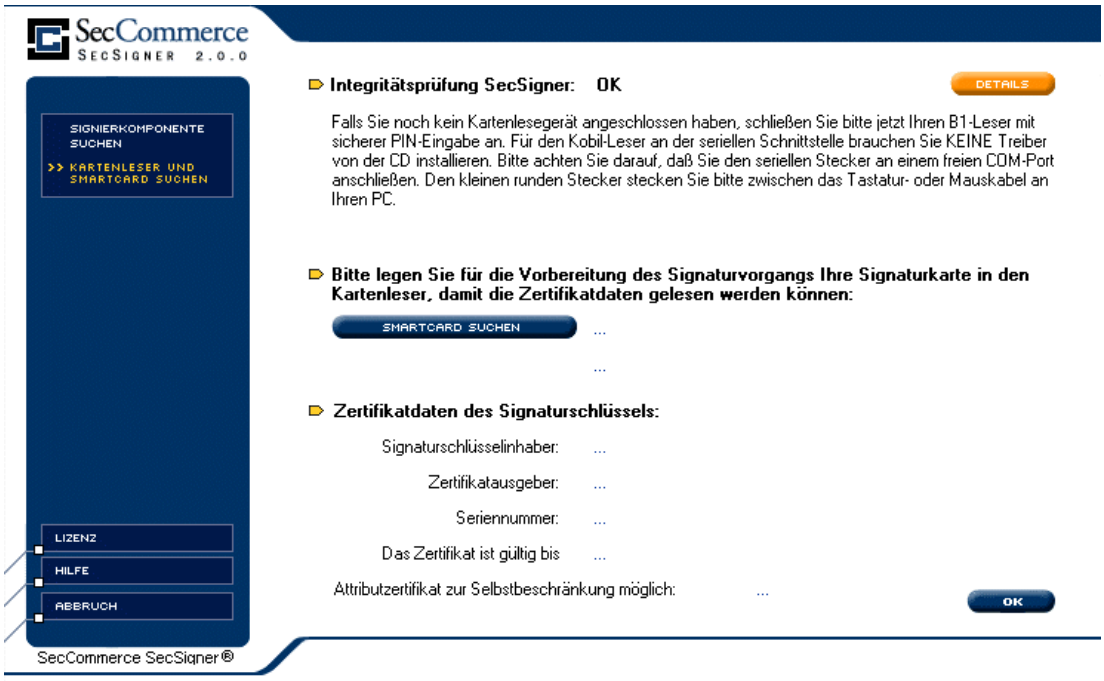


Abbildung 10.1: Die grafische Oberfläche von SecSigner. Entnommen aus http://www.seccommerce.de/

```
/*Das Signatur-Callback-Interface von SecSigner muss
  implementiert werden*/
public class SigTest implements SecSignerCallbackInterfaceSign{

  [...]

  public static void main(String[] args) {

    [...]

    try{

      [...]
      //initialisierung von SecSigner
      mgr.init(new Frame(),secSignerProperties,SECSIGNER_PATH);

      //Quelldatei lesen
      File f=new File("Diplomarbeit.txt");
      FileInputStream fis = new FileInputStream(f);
      byte[] test=new byte[(int)f.length()];
      fis.read(test);
```

```
      //Start der Signatur.
      //Damit erscheint die Oberflaeche von SecSigner.
      mgr.sign(new SigTest(),new Integer(1),test);
      mgr.close();
    }
    catch (Exception e){[..]}
  }

  //Diese Callback-Funktion wird nach vollendeter Signatur
  //aufgerufen.
  public void signCommit(byte[] dataToSign, byte[] signature,
                         byte[] timestamp) {

    try {
      //fertige Signatur (PKCS#7) in Datei schreiben
      sig= (byte[])signature.clone();
      File file=new File("Diplomarbeit.sig");
      FileOutputStream fos=new FileOutputStream(file);
      fos.write(signature,0,signature.length);
      fos.close();

    }
    catch (Exception e) {[...]}
  }

  //weiter Methoden des Signaturinterfaces
  [...]
}
```

Die Tatsache, dass die Lizenz von SecSigner Änderungen an der Software verbietet, schränkt jedoch den Nutzen in Bezug auf ein e.P.b. stark ein. Lediglich auf Clientseite ist der Einsatz von SecSigner vorstellbar.

Auf Serverseite muss ein automatisierter Abruf von Zeitstempeln und eine selbstständige Überprüfung der Signaturen möglich sein. Das Fazit nach diesem Überblick über freie Java-Bibliotheken zur Signatur fällt demnach eher ernüchternd aus.

Die frei verfügbaren Bibliotheken genügen nicht den Ansprüchen eines e.P.b.. Für die Implementierung der geplanten Funktionalität eines e.P.b. wird eine Evaluation von Produkten kommerzieller Anbieter nötig.

# 11. Zusammenfassung

Im Rahmen des derzeitigen Engagements der Fachhochschule Gießen und des Verlags für Standesamtswesen bei der Erarbeitung eines Konzeptes für ein elektronisches Personenstandsbuch war es Aufgabe dieser Diplomarbeit die sicherheitsrelevanten Aspekte für ein solches Archivsystem zu untersuchen.

Die Aufmerksamkeit richtete sich dabei besonders auf das Thema der Langzeiterhaltung von digitalen Signaturen und der sicheren Kommunikation verteilter Komponenten. Daneben spielen Fragen der Benutzerauthentifikation und der Rechteverwaltung eine Rolle.

Vor dem eigentlichen Konzept werden die Hintergründe der digitalen Signatur beleuchtet. Anschließend wirft ein Kapitel einen Blick über den Tellerrand. Es zeigt aktuelle Technologien und veröffentlichte Dokumente, die Konzepte zeigen, die für ein elektronisches Personenstandbuch von Belang sind.

Für die Langzeiterhaltung der digitalen Signaturen wurde ein Konzept, angelehnt an die Empfehlungen des Bundesamtes für Sicherheit in der Informationstechnik, entwickelt. Dabei ist vorgesehen die Signatur mittels Zeitstempel, die im Laufe der Zeit immer wieder erneuert werden müssen, langfristig zu sichern.

Die Zeitstempel versehen die Signaturen dabei mit einem verlässlichen Datum. Damit kann bewiesen werden, dass eine Signatur zu einer Zeit entstanden ist, zu der die Rahmenbedingungen (Zertifikate und Algorithmen) eine rechtsgültige Signatur ermöglichten. Damit wird diese Rechtsgültigkeit für die Zukunft gesichert. Um eine Signatur auch nach Jahren noch auswerten zu können, werden zusätzlich Validierungsinformationen wie Zertifikate archiviert.

Neben der Beschreibung dieses Konzeptes zeigt die Diplomarbeit auch die erforderlichen Programm- und Datenstrukturen, die zu einer Implementierung des Konzeptes herangezogen werden können. Diese entsprechen dem aktuellen Stand bei Abgabe der Diplomarbeit und können bei zukünftigen Weiterentwicklungen als Richtschnur dienen.

Für die Authentifikation der zugriffsberechtigten Benutzer wird ebenfalls auf diese qualifizierten Zertifikate zurückgegriffen. Dadurch wird hierbei eine hohe Sicherheit gewährleistet.

In der Diplomarbeit wird eine SSL-basierte Kommunikation der verteilten Komponenten empfohlen. Ein verteiltes elektronisches Personenstandsbuch ist zwar derzeit nicht geplant, diese Diplomarbeit zeigt dennoch eine konzeptionelle Lösung dafür.

Der letzte Teil befasst sich mit der Implementierung der geplanten Funktionalität in Java. Dabei werden jeweils Teilaspekte herausgegriffen und ihre Unterstützung seitens Java mit Codebeispielen erläutert.

# Teil V

# Anhänge

# A. API der Sicherheitskomponente

Die im Folgenden vorgestellte API der Sicherheitskomponente entspricht dem Stand des e.P.b. bei Abgabe dieser Arbeit. Bei späteren Änderungen am Konzept kann sie als Richtschnur dienen.

Im Klassendiagramm sind diese Schnittstellen in den Interface-Deklarationen zu finden. Die Klasse secComponent soll später die Implementierung dieser Methoden bieten.

Die einzelnen Interfaces stellen folgende APIs:

- *secSignature* - Signatur-API
- *secAuth* - Hilfsfunktionen zur Benutzerauthentifikation
- *secConfig* - Konfigurations-API

## Exceptions und Returncodes

Die API definiert einen Satz von **Exceptions**, die von den Methoden geworfen werden können.

```
SignatureFailedException
CertificateFailedException
AccessFailedException
SecInfoPresentException
NotConfiguredException
ExternalServiceException
NoIntegrityException
WrongFormatException
BadCertificateException
CertificateNotFoundException
CertificateInUseException
```

Des Weiteren können die Methoden folgende **Fehlercodes** zurückliefern:

```
enum Errno{
       OK,
       FAILED
}
```

Welche Exception von einer API-Methode geworfen werden kann, welche Fehlercodes auftreten können und welche Bedeutung sie haben, befindet sich im Folgenden jeweils bei der API-Methode, die diesen Status zurückliefern kann.

## A.1 Archivierungs-API

### Einlagerung eines neuen Eintrags

Zur Einlagerung eines neuen Eintrags existiert die folgende Methode:

```
Errno createNewSecInfo(byte[] entryDat, string entryKey,
                       byte[] signature, int certNr)
      throws SignatureFailedException, CertificateFailedException,
             AccessFailedException, SecInfoPresentException,
             NotConfiguredException, ExternalServiceExcpetion;
```

Die Methode nimmt einen Eintrag als Byte-Array (hier: die PDF-Datei), den zugehörigen Eintragsschlüssel[1], die Signatur und Nummer des eingespeicherten User-Zertifikats entgegen.

Der Ablauf ist dann folgender: Zuerst wird überprüft, ob die Nummer des Zertifikats auf ein gültiges User-Zertifikat verweist, welches von einem als vertrauenswürdig bekannten Trustcenter (eines, das in der Tabelle *secCACert* eingetragen ist) ausgestellt wurde. Dabei wird der Gültigkeitszeitraum des Zertifikats und die Signatur des ausstellenden Trustcenters überprüft.

Anschließend folgt die mathematische Überprüfung der Signatur. Gab es bis zu dieser Stelle keine Unstimmigkeiten mit der Signatur, wird ein Zeitstempel und schließlich eine OCSP-Antwort angefordert.

Alle gesammelten Informationen werden dann über die Persistenzkomponente in der Datenbank abgelegt. Der Eintrag wird dabei nur entweder vollständig oder gar nicht im Archiv gespeichert. Ist bei der Überprüfung der Zertifikate oder der Signatur ein Fehler aufgetreten oder konnte kein Zeitstempel oder OCSP-Antwort eingeholt werden, wird eine Exception geworfen. In diesem Fall wurde nichts in der Datenbank abgespeichert und der Methodenaufruf muss wiederholt werden.

Die Signatur muss dabei dieselben Parameter erfüllen, die auch in der Konfiguration der Sicherheitskomponente gewählt wurden. Dabei handelt es sich um Algorithmen und Schlüssellängen. Sonst kann die Signatur nicht überprüft werden.

Folgende **Fehlercodes** können auftreten:

- **OK**: Die Signatur wurde geprüft und ist gültig. Alle Sicherheitsinformationen wurden erstellt und in der Datenbank abgelegt.

Folgende **Exceptions** können auftreten:

- **SignatureFailedException**: Die Originalsignatur ist nach mathematischer Prüfung nicht korrekt und wurde zurückgewiesen.

[1] Hier wurde zur Vereinfachung string als Typ für den Schlüssel gewählt. In Wirklichkeit handelt es sich dabei um einen zusammengesetzten Datentyp. Wie sich der Schlüssel letztendlich zusammensetzt, soll nicht Bestandteil dieses Dokuments sein.

- **CertificateFailedException**: Das Zertifikat ist entweder abgelaufen, wurde zurückgezogen oder wurde von einem Trustcenter ausgestellt, welches nicht in *secCaCerts* eingetragen ist. Die Signatur wurde zurückgewiesen.
- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.
- **SecInfoPresentException**: Wird versucht für einen Eintrag neue Sicherheitsinformationen zu erstellen, der schon Sicherheitsinformationen besitzt, wird diese Exception geworfen.
- **NotConfiguredException**: Diese Exception wird geworfen, wenn nicht alle für den Betrieb der Sicherheitskomponente notwendigen Einstellungen getroffen wurden.
- **ExternalServiceException**: Bei dem Aufruf eines entfernten Dienstes (OCSP oder Zeitstempel) ist ein Fehler aufgetreten.

### Prüfung der Integrität eines Eintrags

Bevor ein Eintrag vom e.P.b. ausgeliefert wird, muss er auf seine Unverfälschtheit hin untersucht werden. Dies geschieht, indem der letzte Zeitstempel geprüft wird. Dabei handelt es sich um einen rein mathematischen Vorgang, da das Zertifikat des Trustcenters im System bekannt ist.

Diese Methode lädt alle bisherigen Sicherheitsinformationen und überprüft den letzten Zeitstempel, der sich über alle anderen Sicherheitsinformationen erstreckt.

```
Errno checkEntryIntegrity(byte[] entryDat, string entryKey)
      throws AccessFailedException, NotConfiguredException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Der letzte Zeitstempel wurde geprüft und ist gültig. Der Eintrag kann als unverändert angesehen werden.
- **FAILED**: Der Zeitstempel passt nicht zum Eintrag. Die Integrität des Eintrags ist verletzt.

Folgende **Exceptions** können auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.
- **NotConfiguredException**: Diese Exception wird geworfen, wenn nicht alle für den Betrieb der Sicherheitskomponente notwendigen Einstellungen getroffen wurden.

### Beweiserhaltende Maßnahmen ergreifen

In regelmäßigen Abschnitten müssen alle Einträge auf den neuesten kryptografischen Stand gebracht werden. In dem vorliegenden Konzept muss jedem Eintrag ein neuer Zeitstempel angebracht werden, der sich über den alten Eintrag und alle bisherigen Signaturen und Zeitstempel erstreckt.

Die Methode versorgt einen Eintrag mit einem neuen Zeitstempel. Es liegt im Aufgabenbereich der e.P.b.-Transaktionen, diese Methode für alle Einträge aufzurufen, die einen neuen Zeitstempel benötigen.

Gewöhnlicherweise wird ein Administrator diesen Vorgang von Hand in Gang setzen, wenn sich Änderungen bei den Zulassungen für Algorithmen oder bei den Trustcenterzertifikaten ergeben.

Bevor ein neuer Zeitstempel angehängt wird, überprüft die Methode die Integrität des Eintrags anhand des letzten Zeitstempels.

```
Errno secureEntry(byte[] entryDat, string entryKey)
      throws AccessFailedException,NoIntegrityException,
             NotConfiguredException, ExternalServiceException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Die Methode konnte alle Einträge bearbeiten. Das Archiv ist auf dem neuesten Stand.

Folgende **Exceptions** können auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.
- **NoIntegrityException**: Bei der Prüfung der Integrität des Eintrags anhand des letzten Zeitstempels gab es einen Fehler. Deshalb wurde kein neuer Zeitstempel angehängt.
- **NotConfiguredException**: Diese Exception wird geworfen, wenn nicht alle für den Betrieb der Sicherheitskomponente notwendigen Einstellungen getroffen wurden.
- **ExternalServiceException**: Bei dem Aufruf eines entfernten Dienstes (OCSP oder Zeitstempel) ist ein Fehler aufgetreten.

### Informationen zur Beweisführung abrufen

Diese Methode wird dann gebraucht, wenn die Rechtskräftigkeit eines Eintrags in Frage gestellt wird. In einem solchen Fall werden alle Informationen ausgeliefert, die ein unabhängiger Prüfer zur Beweisführung benötigt.

Diese Daten befinden sich in den Sicherheitsinformationen. Die Auslieferung erfolgt als Bytefolge aller Daten, so hintereinander gehängt wie auch die Zeitstempel erzeugt wurden. Die Interpretation der Daten liegt in der Aufgabe des Prüfers. Das genaue Ausgabeformat wurde noch nicht spezifiziert.

```
byte[] getSecInfo(byte[] entryDat, string entryKey)
          throws AccessFailedException, NotConfiguredException;
```

Folgende **Exception** kann auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.
- **NotConfiguredException**: Diese Exception wird geworfen, wenn nicht alle für den Betrieb der Sicherheitskomponente notwendigen Einstellungen getroffen wurden.

## A.2 Hilfsfunktionen zur Benutzerauthentifikation

### Ein Zertifikat auf seine Gültigkeit überprüfen

Mithilfe dieser Methode kann überprüft werden, ob ein eingespeichertes Zertifikat (User-, Trustcenter- oder Serverzertifikat) noch gültig ist. Dazu werden alle nötigen mathematischen Überprüfungen und, wenn es möglich ist, eine OCSP-Abfrage durchgeführt.

```
Errno checkCertificate(int certNr)
      throws AccessFailedException, ExternalServiceException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Das Zertifikat wurde überprüft und ist im Moment gültig.
- **Failed**: Das Zertifikat ist nicht (mehr) gültig.

Folgende **Exceptions** können auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.
- **ExternalServiceException**: Bei dem Aufruf eines entfernten Dienstes (OCSP oder Zeitstempel) ist ein Fehler aufgetreten.

### Eine Zufallszahl erzeugen

Falls ein Generator für kryptografische Zufallszahlen im e.P.b. eingesetzt werden soll, so muss nur die Sicherheitskomponente für den Zugriff darauf angepasst werden. Sie bietet eine Methode, um Zufallszahlen zu erhalten.

Die Methode erzeugt Integer-Zufallszahlen im Bereich von 0 bis zu der mit maxNumber spezifizierten Zahl.

```
int randomNumber(int maxNumber);
```

### Verschlüsseln mit dem public key aus einem Zertifikat

Das aktuelle Konzept zur Benutzeranmeldung erfordert die Verschlüsselung eines Wertes mit dem öffentlichen Schlüssel des Benutzers. Damit wird überprüft ob dieser Benutzer auch über den zugehörigen privaten Schlüssel verfügt.

Der Methode werden Zertifikatsnummer und Wert übergeben. Der verschlüsselte Wert wird zurückgegeben. Neben Userzertifikaten ist diese Methode auch auf Serverzertifikate anwendbar.

```
byte[] encryptByPublicKey(int certNr, byte[] data)
      throws AccessFailedException;
```

Folgende **Exceptions** können auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.

## A.3 Konfigurations-API

Die Methoden der Archivierungs-API benötigen einige globale Parameter, um ihre Funktionen erfüllen zu können. Dabei handelt es sich um die beteiligten Zertifikate, Adressen zum Zugriff auf die Trustcenterdienste und Signaturparameter.

Diese Parameter können sich ändern und müssen dann angepasst werden, möglichst ohne den Quellcode anzupassen. Dazu muss die Sicherheitskomponente möglichst stark konfigurierbar sein.

Alle Einstellungen dürfen nur von autorisierten Administratoren veränderbar sein. Dafür müssen die darüber liegenden Programmschichten sorgen.

### Ausgabe der aktuellen Einstellungen

Diese Methode gibt die komplette Liste aller Parameter - Wert - Paare aus. Dazu wird eine Collection von Objekten der Klasse `SecConfigEntry` zurückgegeben. Es handelt sich dabei um allgemeine Einstellungen der Signaturkomponente. Zur Konfiguration der Zertifikate existieren weitere Methoden.

Für diesen Zweck existieren zwei Methoden. Die erste `getActualConfiguration()` liefert alle aktuellen Einstellungen. Die zweite `getFormerConfiguration()` dient dazu die Konfiguration der Sicherheitskomponente zu einem bestimmten Zeitpunkt zu rekonstruieren. Deshalb muss ihr ein Zeitpunkt übergeben werden. Das Datum wird als string im Format *TT.MM.JJJJ-SS:MM* übergeben.

```
Collection getActualConfiguration()
           throws AccessFailedException;

Collection getFormerConfiguration(string datum)
           throws AccessFailedException
```

Folgende **Exception** kann auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.

### Neue Einstellungen bekanntmachen

Mit dieser Methode können neue Einstellungen übermittelt werden. Diese müssen in Form einer Collection von `SecConfigEntry` Objekten übergeben werden. Es ist dabei zu beachten, dass immer alle Einstellungen auf einmal übermittelt werden müssen. Sonst liefert die Methode *failed* zurück.

In dem Moment, in dem die neuen Einstellungen eingetragen werden, werden die bisherigen Einstellungen als veraltet gekennzeichnet und archiviert.

```
Errno setConfiguration(Collection Entries)
          throws AccessFailedException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Die neuen Einstellungen wurden übernommen.
- **Failed**: Es wurden nicht alle oder falsche Einstellungen übergeben.

Folgende **Exception** kann auftreten:

- **AccessFailedException**: Der Zugriff auf die Persistenzkomponente lieferte eine AccessFailedException. Die Sicherheitskomponente wirft diese Exception weiter.

### Vertrauenswürdige Zertifikate eintragen und verwalten

Die Methode `addCaCert()` wird dazu verwendet, die Zertifikate der Trustcenter einzutragen, die vom System als vertrauenswürdig erkannt werden sollen. Userzertifikate können später nur eingetragen werden, wenn vorher das zugehörige Trustcenterzertifikat eingetragen wurde. Ansonsten würde das Userzertifikat vom System als nicht vertrauenswürdig eingestuft werden.

Es kann in dem speziellen Fall, bei dem viele Trustcenter im Einsatz sind, auch sinnvoll sein, das Zertifikat der Root-Instanz einzutragen. Dann können alle Trustcenterzertifikate bei der Root-Instanz angefordert und überprüft werden.

Beim Einspeichern stellt die Methode die Verbindungen zu eventuell vorhandenen Root-Zertifikaten selbstständig her. Die Zertifikate der Trustcenter können nur aus dem System entfernt werden (mit `deleteCaCert()` ), wenn keine anderen Zertifikate mehr darauf verweisen.

Mithilfe der Methode `configCaCert()` kann das Zertifikat nicht geändert werden. Vielmehr werden über diese Methode die Adressen zum Zugriff auf die Trustcenterdienste konfiguriert.

Um eine Liste aller Zertifikate zu erhalten, wird die Methode `getCaCerts()` aufgerufen. Dieses gibt eine Collection von Objekten der Klasse `SecCACert` zurück. Alternativ kann die Methode `getCaCert()` verwendet werden, um ein bestimmtes Zertifikat abzufragen.

```
Errno addCACert(byte[] certificate, byte[] certFormat)
          throws WrongFormatException, BadCertificateException;

Errno deleteCACert(certNr int)
          throws CertificateNotFoundException;

Errno configCaCert(certNr int, string TSSUrl, string OCSPUrl)
          throws CertificateNotFoundException;

Collection getCaCerts();

SecCaCert getCaCert(int certNr)
          throws CertificateNotFoundException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Das Zertifikat wurde gespeichert und wird ab sofort verwendet [`addCaCert()`]. Das Zertifikat wurde gelöscht [`deleteCaCert()`]. Die Konfiguration wurde eingetragen [`configCaCert()`].

Folgende **Exceptions** können auftreten:

- **WrongFormatException**: Es wurde ein Speicherformat gewählt, welches von dem aktuellen Security Provider nicht zur Verfügung gestellt wird.
- **BadCertificateException**: Das Zertifikat ist nicht gültig oder hat ein falsches Format.
- **CertificateNotFoundException**: Das per certNr angeforderte Zertifikat existiert nicht oder ist kein Trustcenterzertifikat.

### User- und Serverzertifkate eintragen und verwalten

Diese Methoden haben dieselbe Namensstruktur wie die Methoden zur Verwaltung der Trustcenterzertifikate. Deshalb wird hier neben der Nennung der Methoden lediglich auf die Unterschiede eingegangen.

Userzertifikate müssen von einem Trustcenter ausgestellt worden sein, dessen Zertifikat schon im System bekannt ist. Ansonsten werden sie zurückgewiesen. Serverzertifikate müssen nicht von einem solchen Trustcenter ausgestellt worden sein. Hier ist die Verbindung optional. Die Methoden erkennen diese Verbindungen automatisch und tragen sie in die Datenbank ein.

Userzertifkate können nur dann gelöscht werden, wenn kein Eintrag mit diesem Zertifikat unterschrieben wurde. Serverzertifikate können jederzeit gelöscht werden.

Die Methoden zur Abfrage der Zertifikate liefern Objekte der Klasse `SecUserCert` für Userzertifikate und `SecServerCert` für Serverzertifikate zurück.

```
Errno addUserCert(byte[] certificate, byte[] certFormat)
          throws WrongFormatException, BadCertificateException;

Errno deleteUserCert(certNr int)
          throws CertificateNotFoundException,
                 CertificateInUseException;

Collection getUserCerts();

SecUserCert getUserCert(int certNr)
          throws CertificateNotFoundException;

Errno addServerCert(byte[] certificate, byte[] certFormat)
          throws WrongFormatException, BadCertificateException;

Errno deleteServerCert(certNr int)
          throws CertificateNotFoundException;

Collection getServerCerts();

SecUserCert getServerCert(int certNr)
          throws CertificateNotFoundException;
```

Folgende **Fehlercodes** können auftreten:

- **OK**: Das Zertifikat wurde gespeichert und wird ab sofort verwendet [`addUserCert()`; `addServerCert()`].
  Das Zertifikat wurde gelöscht [`deleteCaCert()` , `deleteServerCert()`].

Folgende **Exceptions** können auftreten:

- **WrongFormatException**: Es wurde ein Speicherformat gewählt, welches von dem aktuellen Security Provider nicht zur Verfügung gestellt wird.
- **BadCertificateException**: Das Zertifikat ist nicht gültig oder hat ein falsches Format.
- **CertificateNotTrustedException:** Es konnte kein passendes Trustcenterzertifikat zu einem neuen Userzertifikat ermittelt werden.
- **CertificateNotFoundException**: Das per certNr angeforderte Zertifikat existiert nicht oder ist kein User- oder Serverzertifikat.
- **CertificateInUseException**: Das Userzertifikat konnte nicht gelöscht werden, weil mindestens ein Eintrag damit unterschrieben wurde.

# Abbildungsverzeichnis

# Literaturverzeichnis


[1] *Grundbuchordnung*.

[2] *Java Secure Socket Extension (JSSE) - Reference Guide v1.4.2*.

[3] *Verordnung zur Durchführung der Grundbuchordnung (Grundbuchverfügung - GBV)*.

[4] *BSI-Handbuch für digitale Signaturen*. 1997.

[5] *Begründung zur Verordnung zur elektronischen Signatur*. 2001.

[6] *Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften*. 2001.

[7] *Verordnung zur elektronischen Signatur (Signaturverordnung - SigV)*. 2001.

[8] *KoopA-ADV Handlunsleitfaden für die Einführung der elektronischen Signatur und Verschlüsselung in der Verwaltung*. 2002.

[9] *PKCS #1 v2.1: RSA Cryptography Standard*. 2002.

[10] U. Brandner, R.; Pordesch. *Long-Term Conservation of Provability of Electronically Signed Documents*. 2002.

[11] Bundesministerium des Innern. *SAGA - Standards und Architekturen für eGovernment Anwendungen Version 2.0*. 2003.

[12] Bundesamt für Sicherheit in der Informationstechnik. *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV*. 1999.

[13] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheitsleitlinien der Wurzelzertifizierungsinstanz der Verwaltung*. 2003.

[14] Regulierungsbehörde für Telekommunikation und Post (RegTP) nach Angaben des Bundesamtes für Sicherheit in der Informationstechnik (BSI). *Maßnahmenkatalog für technische Komponenten nach dem Signaturgesetz*. 1998.

[15] IETF. *The TLS Protocol Version 1.0 (RFC 2246)*. 1999.

[16] European Telecommunications Standards Institute. *ETSI TS 101 733 V1.2.2 - Electronic Signature Formats*. 2000.

[17] java.sun.com. *Java Cryptography Architecture - API Specification & Reference*. 2002.

[18] java.sun.com. *Java Cryptography Extension - Reference Guide*. 2002.

[19] OSCI-Leitstelle. *OSCI: Die informelle Beschreibung*. 2001.

[20] PKIX. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP) - RFC2560*. 1999.

[21] PKIX. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) - RFC3161*. 2001.

[22] Klaus Schmeh. *Kryptografie und Public-Key-Infrastrukturen im Internet*. dpunkt.verlag, 2. edition, 2001.

# Index