Zoltán Nochta

Zertifikatsbasierte Zugriffskontrolle in verteilten Informationssystemen

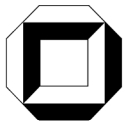
universitätsverlag karlsruhe

Zoltán Nochta

Zertifikatsbasierte Zugriffskontrolle in verteilten Informationssystemen

Zertifikatsbasierte Zugriffskontrolle in verteilten Informationssystemen

von
Zoltán Nochta



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH), Fakultät für Informatik, 2004

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe

www.uvka.de

Print on Demand

ISBN 3-937300-30-9

Zertifikatsbasierte Zugriffskontrolle in verteilten Informationssystemen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Zoltán Nochta

aus Mór/Ungarn

Tag der mündlichen Prüfung:	21. 7. 2004
Erster Gutachter:	Prof. Dr. S. Abeck
Zweiter Gutachter:	Prof. Dr. P. Lockemann

„Die Worte tun dem geheimen Sinn nicht gut,
es wird immer alles gleich ein wenig anders,
wenn man es ausspricht,
ein wenig verfälscht, ein wenig närrisch -
... was eines Menschen Schatz und Weisheit ist,
dem andern immer wie Narrheit klingt."

(H. Hesse)

Vorwort

Diese Dissertation wurde in der Forschungsgruppe Cooperation & Management am Institut für Telematik der Fakultät für Informatik an der Universität Fridericiana zu Karlsruhe (TH) erstellt.

Ich bin dankbar für die Unterstützung, die mir von vielen Seiten zuteil wurde, und die freundschaftliche Atmosphäre am Institut, in der das Arbeiten viel Freude bereitete.

Mein besonderer Dank gilt Herrn Prof. Dr. Sebastian Abeck für die wissenschaftliche Betreuung meiner Arbeit. Ich möchte mich insbesondere für das mir entgegengebrachte Vertrauen und die mir zur Verfügung gestellten Freiheiten bedanken, die mir das selbständige Arbeiten ermöglichten.

Herrn Prof. Dr. Peter Lockemann danke ich herzlich für die freundliche Übernahme des Korreferats. Seine Kommentare und Anregungen trugen dazu bei, die vorliegende Arbeit an vielen Stellen zu verbessern.

Ich bedanke mich bei meinen Kollegen Jodok Batlogg, Mike Becker, Dirk Feuerhelm, Markus Gebhard, Andreas Köppel, Karsten Krutz, Christian Mayerl, Oliver Mehl, Robert Scholderer und bei allen von mir betreuten Studenten, insbesondere Hendrik Bock, Peter Ebinger, Kai Kühn, Jan Oetting und Alexander Viehl für die regen Diskussionen, die wir gemeinsam führten.

Ebenfalls danke ich Herbert Eisenbeis, der mir stets zur Seite stand und mich, wo er nur konnte, unterstützte.

Iris Perner kann ich nicht dankbar genug sein, für ihre Geduld und Liebe.

Inhaltsverzeichnis

1 Einführung ... 1

1.1 Motivation und Problemstellung ... 1
1.2 Zusammenfassung der Ergebnisse ... 5
1.3 Überblick ... 7

2 Grundlagen ... 9

2.1 Zugriffskontrolle und Autorisierung in verteilten Systemen ... 9
2.1.1 Prozess der Zugriffskontrolle ... 10
2.1.2 Modelle zur Realisierung der Zugriffskontrolle ... 11
2.1.3 Autorisierung: Die administrative Grundlage der Zugriffskontrolle ... 13
2.2 Attributszertifikate und relevante Techniken ... 17
2.2.1 Kryptographische Hashfunktionen ... 17
2.2.2 Digitale Signaturen ... 18
2.2.3 Digitale Signaturen mittels Authentifizierungsbäumen ... 20
2.2.4 X.509-Attributszertifikate ... 23
2.3 Realisierung der Zugriffskontrolle mit Attributszertifikaten ... 28
2.3.1 Durchgängiger Schutz von Autorisierungsdaten ... 28
2.3.2 Delegierung von Rechten: Sicherheit durch Attributszertifikate ... 31
2.3.3 Prozess der Prüfung von Attributszertifikaten ... 36

3 Stand der Technik ... 41

3.1 Gültigkeitsprüfung aufgrund von Sperrdaten ... 42
3.1.1 Listenbasierte Sperrtechniken ... 43
3.1.2 Sperrbäume ... 47
3.1.3 Certificate Revocation System ... 49
3.2 Einsatz von Online-Statusdiensten ... 51
3.2.1 Online Certificate Status Protocol (OCSP) ... 52
3.2.2 Data Validation and Certification Server (DVCS) ... 53
3.2.3 Simple Certificate Validation Protocol (SCVP) ... 53
3.3 Gültigkeitsprüfung anhand von Gültigkeitsinformationen ... 54
3.3.1 „Whitelist"-Techniken ... 54
3.3.2 Technik der „Suicide Bureaus" ... 55

4 Häufige Aktualisierung von Attributszertifikaten ... 57

4.1 Certification Verification Tree ... 59
4.2 Einsatz von CVTs für die zertifikatsbasierte Zugriffskontrolle ... 60
4.3 Probleme mit CVTs und Lösungswege ... 63
4.3.1 Effizienz der Prüfung von Zertifizierungspfaden ... 63
4.3.2 Behandlung sicherheitskritischer Probleme ... 66

5 Improved Certification Verification Tree ... 75

5.1 Sichere Existenz- und Nicht-Existenz-Beweise ... 75
5.1.1 Generierung und Prüfung von Existenz-Beweisen ... 79
5.1.2 Generierung und Prüfung von Nicht-Existenz-Beweisen ... 82
5.2 Vollständigkeitsbeweise ... 85
5.2.1 Vollständigkeitsbeweise für Client-Push-Systeme ... 88
5.2.2 Vollständigkeitsbeweise für Server-Pull-Systeme ... 90
5.3 Realisierung auf Basis eines B^+-Baums ... 95
5.3.1 Erzeugung eines I-CVT auf Basis eines B^+-Baums ... 98
5.3.2 Generierung und Verifizierung von Zertifizierungspfaden ... 99
5.3.3 Abhängigkeit von der Sicherheit der verwendeten Hashfunktion ... 104
5.3.4 Generierung und Verifizierung von Vollständigkeitsbeweisen ... 106

6 Konzepte für die effiziente und sichere Ursprungsprüfung 115

6.1 Festhalten der Delegierungshistorie in den Attributszertifikaten 117
6.2 Online-Dienste zur Unterstützung der Ursprungsprüfung 120
6.3 Zertifikatsreduktion 122
6.4 Technik der Offline-Delegierung 125
6.4.1 Methodik zur Pflege von Delegierungsinformationen 128
6.4.2 Betriebliche Auswirkungen der Offline-Delegierung 131

7 PAMINA 135

7.1 Einsatzszenario: Internet-basierter Wissenstransfer 135
7.1.1 ed.tec 136
7.1.2 Zugriffskontrolle für ed.tec 138
7.2 PAMINA Administration Server 141
7.2.1 Bedienung und Zertifizierungsvorgänge im Überblick 142
7.2.2 Interner Aufbau 149
7.3 Privilege Database 154
7.3.1 Interner Aufbau 155
7.4 Certificate Verifier 159
7.4.1 Interner Aufbau 160

8 Ausblick 165

9 Literaturquellen 169

1 Einführung

Der Wohlstand heutiger industrialisierter Gesellschaften fußt auf der technischen Infrastruktur, welche ein Ergebnis wissenschaftlicher Fortschritte und interdisziplinärer Ingenieurkunst ist. Rechnergestützte verteilte Informationssysteme sind in den letzten Dekaden wichtige Mosaiksteine dieser Infrastruktur geworden. Sie bilden das Rückgrat des elektronisierten Bank-, Bildungs-, Gesundheits-, Handlungs-, Verkehrs- und Verwaltungswesens und unterstützen die automatisierte Produktion von Gütern, um nur einige Anwendungsbereiche zu erwähnen. Der alltägliche Gebrauch vielfältiger Dienste dieser Systeme - in deren gewohnten Qualität - ist mittlerweile zu einer Selbstverständlichkeit geworden.
Dem Bedarf entsprechend müssen Hersteller sowie Betreiber neben der Erbringung der jeweilig erwarteten Nutzfunktionalität auch für die Qualitätssicherung ihrer Systeme sorgen. Der qualitätsgesicherte Einsatz von Systemen macht zusätzliche Investitionen erforderlich, welche häufig zu keinem direkt spürbaren Gewinn führen. Die Wettbewerbsfähigkeit der Systembetreiber hängt deshalb immer stärker davon ab, wieweit sie ihre Betriebskosten bei einer ständigen Verbesserung der Dienstqualität senken können. Neben Qualitätsmerkmalen, wie beispielsweise Performanz, Durchsatz, Erreichbarkeit oder Bedienbarkeit, gewinnen heute unterschiedliche Sicherheitsaspekte zunehmend an Bedeutung.
Ein zentrales Problem dabei ist der präventive Schutz wertvoller Ressourcen, das heißt elektronisch erfasster Informationen sowie Hardware- und Softwarekomponenten, gegen potenzielle Bedrohungen. Es gibt eine endlose Anzahl an Beispielen für schädliche Ereignisse, die häufig als Folge beabsichtigter Angriffe gegen ein System auftreten. Die Palette reicht von der unbefugten Nutzung und Manipulation von Daten oder Programmcode bis hin zur Lahmlegung von Diensten. Mangels geeigneter Gegenmaßnahmen können solche Angriffe zu erheblichen Schäden in einem System führen. Gelingt es diese zu verhindern, so tragen die Sicherheitsvorkehrungen zur eigentlichen Wertschöpfung bei.
Die Zugriffskontrolle, welche im Mittelpunkt der Betrachtungen dieser Arbeit steht, ist eine der Sicherheitsmaßnahmen innerhalb eines verteilten Informationssystems. Eine wichtige Aufgabe der Zugriffskontrolle ist zu entscheiden, ob ein aktuell Zugreifender autorisiert ist, die anvisierte Funktion des abgesicherten Systems in Anspruch zu nehmen. Hinter Zugriffskontrollsystemen stehen heute weitestgehend automatisiert ausgeführte Prozesse. Diese Prozesse werten gemäß bestimmter Regeln verschiedene im Voraus erzeugte und abgelegte Autorisierungsdaten, wie beispielsweise Berechtigungen oder Gruppenzugehörigkeiten des Zugreifenden aus. Autorisierungsdaten avancieren somit zur wichtigen Grundlage einer effektiven Zugriffskontrolle. Daher ist ihr Schutz für die zweckmäßige Zugriffskontrolle und damit für die Sicherheit des gesamten Systems von Bedeutung.

1.1 Motivation und Problemstellung

Zugriffskontrollmechanismen wurden bereits in den frühesten (verteilten) Rechnerarchitekturen - hauptsächlich mit militärischem Hintergrund - eingesetzt. Heute etablierte Zugriffskontrollmodelle, sowie die verschiedenen Verfahren und Sprachen zur Beschreibung, Strukturierung und Auswertung von Autorisierungsdaten sind Ergebnisse zahlreicher Forschungsarbeiten.

Mit der sicheren Bereitstellung und Überprüfung von Autorisierungsdaten beschäftigten sich hingegen bisher vergleichsweise wenige Arbeiten. Es wird in den meisten Zugriffskontrollsystemen angenommen, dass die verwendeten Autorisierungsdaten korrekt und aktuell sind, und dass sie unverändert in einer als sicher eingestuften Datenablage vorliegen. In solchen Systemen kann die Zugriffskontrolle beispielsweise durch die unbemerkte Änderung der Autorisierungsdaten manipuliert werden. Da ein Angreifer dazu in der Regel gar nicht in das abgesicherte System aktiv einzugreifen braucht, können Angriffe dieser Art lange unentdeckt bleiben.

Um solchen Problemen entgegenzuwirken, sollten Autorisierungsdaten nach ihrer Erstellung durch eine aus Sicht der Zugriffskontrolle vertrauenswürdige Stelle gegen nachträgliche Manipulationen geschützt werden. Eine Möglichkeit hierfür ist die Verwendung so genannter Attributszertifikate (engl. *Attribute Certificates*). Diese sind digital signierte Dokumente, welche Autorisierungsdaten jeglicher Art und Semantik enthalten können. Die gängigen Zugriffskontrollmodelle, beispielsweise die so genannten diskreten oder systembestimmten Modelle lassen sich problemlos unter Zuhilfenahme von Attributszertifikaten umsetzen. Es scheint diesbezüglich keine prinzipiellen Einschränkungen zu geben.
Attributszertifikate können zur Qualitätserhöhung der Zugriffskontrolle in einem System beitragen: Da sie digital signiert sind, stellen sie eine eindeutige und fälschungssichere Bindung zwischen den zertifizierten Autorisierungsdaten und deren Erzeuger dar. Folglich lassen sich mit deren Hilfe genaue Herkunft sowie Integrität der Autorisierungsdaten überprüfen, bevor diese in die Auswertungsprozesse innerhalb der Zugriffskontrolle einfließen. Außerdem ermöglichen sie eine sichere Nachweisführung in einem System. Anhand von Attributszertifikaten lässt sich leichter entscheiden, wer welche Berechtigungen zu einem bestimmten Zeitpunkt besessen, beziehungsweise an andere vergeben hat.
Da Attributszertifikate fälschungssicher sind, können sie und damit die enthaltenen Autorisierungsdaten über unsichere Medien, wie beispielsweise das Internet gesendet werden, ohne dass befürchtet werden muss, dass sie unentdeckt manipuliert werden. Außerdem können sie in Datenbanken gehalten werden, deren eventuell korrumpierte Betreiber sie ebenfalls nicht unbemerkt verändern können.
Des Weiteren ermöglichen es Attributszertifikate, Berechtigungen in einer sicheren Art und Weise an Andere weiterzugeben, d.h. zu delegieren. Durch Rechtedelegierung kann die Dynamik der heute häufig statischen, von einer zentralen Stelle geregelten Berechtigungsstrukturen erhöht werden. Dies kann in den verschiedensten Szenarien ausgenutzt werden, welche eine Flexibilität der Systeme in dieser Hinsicht erforderlich machen.

Dank der genannten Vorteile setzen sich Attributszertifikate in der Praxis immer stärker durch, wobei noch von keinem echten Durchbruch gesprochen werden kann. Es bedarf womöglich noch einiger Jahre zum einen ähnlich großen Erfolg zu verzeichnen, wie den so genannten Schlüsselzertifikaten (engl. *Public-Key Certificate*) zuteil wurde. Mittels solcher Schlüsselzertifikate lassen sich kryptographische Schlüssel über ungesicherte Kommunikationswege sicher verteilen. Typischerweise bindet ein solches Zertifikat mit Hilfe der digitalen Signatur einer vertrauten Stelle (engl. *Certification Authority*, CA) einen öffentlichen Schlüssel (engl. *Public-Key*) an ein bestimmtes reales Subjekt. Es stellt somit einen digitalen Identitätsnachweis (digitalen Ausweis) dieses Subjekts dar.

Für derart gesicherte öffentliche Schlüssel lassen sich zwei wesentliche Anwendungsgebiete identifizieren. Erstens die Verschlüsselung von Daten für zertifizierte Subjekte. Zweitens die Überprüfung von Echtheit und Ursprung von Daten, welche durch zertifizierte Subjekte digital signiert wurden.
So genannte Public-Key-Infrastrukturen (engl. *Public-Key Infrastructure*, PKI) sind Systeme zur Verwaltung von öffentlichen Schlüsseln beziehungsweise Schlüsselzertifikaten. Die zentrale Aufgabe einer PKI ist die Versorgung eines bestimmten registrierten Benutzerkreises mit stets korrekten Zertifikaten, sowie die Unterstützung der sicheren Überprüfung dieser Zertifikate für Jedermann.

Obwohl es sich in beiden Fällen um das Management von Zertifikaten handelt, eignen sich die heute in PKIs eingesetzten Techniken für die Zwecke einer performanten und sicheren Zugriffskontrolle mittels Attributszertifikaten nur bedingt. Bei der Konstruktion eines Zertifikatsmanagementsystems für Attributszertifikate (engl. *Privilege Management Infrastructure,* PMI) müssen vor allem die in der Natur der zertifizierten Daten (Schlüssel respektive Autorisierungsdaten) liegenden Unterschiede im Auge behalten werden. Die wichtigsten leicht erkennbaren Unterschiede sind:

- Benutzeridentitäten ändern sich in einem gegebenen System normalerweise nur selten. Da Schlüsselzertifikate digitale Benutzeridentitäten repräsentieren, sind sie als weitgehend statische Objekte anzusehen. Dementsprechend werden Schlüsselzertifikate heute mit einer Gültigkeitsperiode ausgestellt, die typischerweise einige Jahre beträgt.
  Im Vergleich dazu verhalten sich Autorisierungsdaten in einem System dynamisch: Benutzer bekommen und verlieren ständig Zugangsberechtigungen zu verschiedenen Systemressourcen, deren Menge und Strukturierung ebenfalls häufigen Veränderungen ausgesetzt ist. Attributszertifikate müssen mit dieser Änderungsdynamik der Autorisierungsdaten Schritt halten. Ansonsten können im System aufgrund veralteter Zertifikate eigentlich zu verhindernde Zugriffe zugelassen beziehungsweise erlaubte Zugriffe verweigert werden. Dies impliziert, dass Attributszertifikate mit einer wesentlich kürzeren Gültigkeitsperiode ausgestellt oder häufiger frühzeitig widerrufen werden müssen.

- Benutzer eines Systems verfügen in der Regel über jeweils eine digitale Identität, die für einen bestimmten Anwendungsfall, wie z.B. für die Datenverschlüsselung verwendet wird. (Das schließt natürlich nicht aus, dass dieselbe Identität für mehrere Anwendungsfälle gilt.) Folglich ist die erwartete Anzahl der Schlüsselzertifikate in einem System vergleichbar mit der Anzahl der Benutzer.
  Anders verhält sich das mit Attributszertifikaten. Deren Anzahl kann bei gleicher Benutzerpopulation im gleichen Anwendungskontext deutlich höher ausfallen. Dies hängt unter anderem von der Anzahl schützenswerter Systemressourcen, der Anzahl der zur Zertifikatsausstellung berechtigten Stellen und der Granularität der Zugriffsrechte ab. In einem Extremfall kann es beispielsweise aus Datenschutzgründen erforderlich sein, atomare Berechtigungen eines Benutzers in jeweils einem separaten Attributszertifikat abzulegen. Erwartungsgemäß kann also mit einem Vielfachen an Attributszertifikaten in einem System gerechnet werden.

- Ein weiterer nennenswerter Punkt ist, dass öffentliche Schlüssel und damit Schlüsselzertifikate nur in den seltensten Fällen Geheimnisse darstellen. Für heute bekannte Angriffe reicht es nicht aus, diese Art von Daten zu kennen.
  Folglich werden sie in öffentlich zugänglichen, das heißt für Jedermann lesbaren Datenbanken gespeichert oder über abhörbare Netze gesendet.
  Autorisierungsdaten sind hingegen häufig vertraulich zu behandeln. Ein Angreifer, der diese Daten kennt, kann Rückschlüsse beispielsweise auf den Aufbau des jeweilig geschützten Systems und die dieses betreibende Organisation ziehen und die gewonnenen Informationen ausnutzen. Es gilt also grundsätzlich den Zugang von Unbefugten zu Attributszertifikaten auszuschließen. Dadurch entsteht ein paradoxes Problem: Die Zugriffe auf Autorisierungsdaten müssen kontrolliert werden, wobei die hierfür benötigten Autorisierungsdaten prinzipiell ebenso schützenswert sind, usw.

Neben den oben erwähnten verwaltungstechnischen Aspekten hängt die Akzeptanz von Attributszertifikaten für eine Zugriffskontrolle maßgeblich davon ab, welchen spürbaren Mehrwert deren Nutzung bringt. Kann die Authentizität und Integrität vorliegender Autorisierungsdaten mittels Attributszertifikaten zufrieden stellend überprüft werden, führt das zu einer Erhöhung der Sicherheit der Zugriffskontrolle.
Dies ist ein klarer Vorteil gegenüber traditionellen Lösungen, die sich auf ungeprüfte Daten verlassen. Dieser Vorteil kann den Betreiber eines existierenden oder neu aufzubauenden Informationssystems dazu motivieren, auf Attributszertifikate zu setzen und eine PMI einzuführen. Ebenfalls kann das Vertrauen der Benutzer gegenüber dem abgesicherten System erhöht werden, wenn dank Attributszertifikaten bestimmte Missbrauchsszenarien ausgeschlossen bleiben.
Allerdings hat die Eliminierung von Sicherheitsrisiken mit Hilfe von Attributszertifikaten ihren Preis. Durch deren Nutzung innerhalb der Zugriffskontrolle entsteht ein erheblicher Aufwand: Die sorgfältige Prüfung einzelner Attributszertifikate ist ein rechen- und folglich zeitaufwändiger Prozess. Diese Prüfung muss erfolgen, bevor die in den Zertifikaten enthaltenen Autorisierungsdaten ausgewertet werden könnten. Dadurch wird die Geschwindigkeit (Performanz) der gesamten Zugriffskontrolle negativ beeinflusst. Dies kann wiederum eine negative Auswirkung auf andere Qualitätsparameter des abgesicherten Systems, wie beispielsweise die Antwortzeit oder den Durchsatz haben.
Reduzierte Risiken auf der einen, erhöhte Kosten auf der anderen Seite erfordern seitens des interessierten Systembetreibers eine Gegenüberstellung dieser beiden Faktoren. Attributszertifikate beziehungsweise eine PMI haben umso mehr Chancen zur Einführung in eine Betriebsumgebung, je kleiner der erwartete Aufwand zur Erreichung der genannten Sicherheitsvorteile ausfällt. Ein rational handelnder Betreiber wird sich für diese Technik erst dann entscheiden, wenn die zu erwartenden Kosten niedriger sind als die geschätzten Schäden (ausfallenden Gewinne), die bei der Verwendung von Attributszertifikaten nun nicht mehr entstehen können.

Die zentrale Zielsetzung dieser Arbeit besteht daher darin, generisch anwendbare Konzepte zu erarbeiten, die ein gutes Kosten-Nutzen-Verhältnis sowie ein hohes Maß an Sicherheit vor allem bei der Überprüfung von Attributszertifikaten aufweisen. Hierbei sind folgende Punkte hervorzuheben:

- Die Reduzierung der durch die Zertifikatsprüfung anfallenden Kosten ist wichtig, um die Performanz der mit dieser Technik abgesicherten Zugriffskontrolle zu erhöhen. Hierfür müssen bei existierenden Ansätzen, die in PKIs und bereits in einigen PMI-Implementierungen zur Zertifikatsprüfung eingesetzt werden, zunächst Defizite aufgedeckt werden.
  Diese Defizite machen sich insbesondere in Systemen bemerkbar, in welchen für Zugriffskontrollentscheidungen jeweils mehrere von derselben Zertifizierungsstelle stammende Attributszertifikate überprüft werden müssen. In solchen Fällen ist meistens eine möglichst schnelle wiederholt ausgeführte Zertifikatsprüfung erforderlich. Ein Aspekt dabei, welcher aber in bisherigen Lösungen unberücksichtigt blieb, ist die Überprüfung, ob der Zugriffskontrolle tatsächlich alle ursprünglich für den jeweiligen Zugriffskontext ausgestellten Attributszertifikate zur Verfügung stehen. Trotz digitaler Signaturen an den Attributszertifikaten ist die Zugriffskontrolle Angriffen ausgesetzt, in welchen eine, die Zertifikate speichernde oder übertragende, maliziöse Instanz bestimmte Zertifikate unerlaubt zurückhält.
  Als Folge fehlender Zertifikate können leicht Fehlentscheidungen während der Zugriffskontrolle getroffen werden.

- Eine besondere Problematik stellt die effiziente Überprüfung von Attributszertifikaten dar, welche durch die oben erwähnte Rechtedelegierung entstanden sind. In heutigen Systemen ist die Entscheidung, ob die eventuell mehrstufig delegierten Rechte tatsächlich in einer der Zugriffskontrolle vertrauten Stelle ihren Ursprung haben, ein komplexer und folglich zeitintensiver Prozess. Auch in solchen Fällen müssen gleichzeitig mehrere Attributszertifikate geprüft werden. Diese können aber von verschiedenen autonom agierenden Stellen stammen. Bekannte Techniken zur Reduktion des hiermit verbundenen Aufwands sind problembehaftet, wie in der vorliegenden Arbeit gezeigt wird.

- Ein wichtiger Aspekt ist die Konformität der Konzepte zu Standards. Ein Standard, der sich im Bereich des Zertifikatsmanagements immer mehr durchsetzt, ist X.509. Dieser beinhaltet neben Empfehlungen für das Management von Schlüsselzertifikaten auch ein Rahmenwerk für Attributszertifikate. Eine Zielsetzung der Arbeit ist, die eigenen Konzepte soweit wie möglich an die Empfehlungen des X.509-Standards anzupassen, um deren Integration in existierende Architekturen bestmöglich zu unterstützen.

- Die Tragfähigkeit der theoretischen Überlegungen soll durch eine prototypisch implementierte PMI nachgewiesen werden. Das entstehende System soll sich in eine auf kommerziellen Komponenten aufbauenden Systemlandschaft integrieren lassen, und dort für eine adäquate Zugriffskontrolle sorgen.

1.2 Zusammenfassung der Ergebnisse

Bekannte Techniken für die Verwaltung und Überprüfung von Zertifikaten wurden zur Unterstützung von PKIs entwickelt. Aufgrund verschiedener Defizite dieser Techniken bezüglich Performanz und Sicherheit wird ein eigener Ansatz vorgestellt. Dieser Ansatz greift auf Vorgängerarbeiten zurück, im Wesentlichen auf Arbeiten von Ralph Charles Merkle [Mer90], Irene Gassko et. al [GGM00] und Ahto Buldas et al. [BLL02].

Einen wichtigen Baustein des vorgestellten Ansatzes bildet die Datenstruktur mit der Bezeichnung *Improved Certification Verification Tree* (I-CVT). Diese ermöglicht die kosteneffiziente und sichere Verwaltung und Überprüfung von Attributszertifikaten und kann die Basis einer PMI bilden. Basierend auf der I-CVT-Technik können die mit Attributszertifikaten verbundenen Verwaltungskosten niedrig gehalten werden.
Wichtiger noch, dass mit Hilfe von I-CVTs die Überprüfung sowohl einzelner als auch gleichzeitig mehrerer Attributszertifikate einer Zertifizierungsstelle effizient und sicher durchführbar ist. Anhand von I-CVTs lassen sich so genannte *Vollständigkeitsbeweise* generieren.
Diese verhindern die unbemerkte Zurückhaltung von auf eine Anfrage passenden Attributszertifikaten durch die diese speichernde Datenbank. Angenommen wird dabei, dass der Anfragende die Anzahl jener Attributszertifikate im Voraus nicht kennt. Einen Spezialfall für Vollständigkeitsbeweise bilden Anfragen, welche höchstens einen Treffer haben können.
In solchen Fällen können so genannte *Existenz-Beweise* beziehungsweise *Nicht-Existenz-Beweise* generiert und vom Anfragenden ausgewertet werden, je nach Erfolg der jeweiligen Suche. Die Möglichkeit, solche Beweise zu generieren macht den Einsatz von I-CVTs neben der gewünschten sicheren Zugriffskontrolle auch für zahlreiche andere Szenarien attraktiv, die auf Datenbankanfragen basieren.

Aus Sicht der performanten Überprüfung während der Zugriffskontrolle haben Attributszertifikate, welche durch Delegierung entstanden, eine Sonderstellung. Bei der zertifikatsbasierten Rechtedelegierung entstehen so genannte Delegierungsnetzwerke, die in den einfachsten Fällen eine Delegierungskette (engl. *Delegation Chain*) bilden. Ein solches Netzwerk besteht aus Zertifikaten, die von verschiedenen Stellen ausgestellt wurden.
Den vertrauten Ursprung solcher Netzwerke, falls es überhaupt einen gibt, während der Zugriffskontrolle zu finden, kann lange Wartezeiten im System verursachen. Ein Vorschlag von Tuomas Aura den hiermit verbundenen Aufwand zu reduzieren, war die Reduktion derartiger Netzwerke auf Attributszertifikate, welche direkt von vertrauten Instanzen ausgestellt werden [Aur99]. Dies hat eine wesentliche Vereinfachung der Ursprungsprüfung delegierter Berechtigungen zur Folge. Einen sicheren Dienst für diesen Zweck innerhalb einer PMI zu konstruieren wirft aber etliche Probleme auf, welche diskutiert werden.
In dieser Arbeit wird deshalb neben anderen Konzepten die Technik der so genannten *Offline-Delegierung* vorgeschlagen. Sie bietet eine einfache Lösung des Problems, indem die gleiche Nutzfunktionalität - sprich die Weitergabe von Berechtigungen - ganz ohne das Entstehen von Delegierungsnetzwerken geschieht. Dies macht die Ursprungsprüfung der Delegierungen einfacher sowie eine nachträgliche Reduzierung von Delegierungsnetzwerken unnötig.

Die kombinierte Verwendung von I-CVTs und der Offline-Delegierung bildet das theoretische Fundament eines prototypisch implementierten Systems mit der Kurzbezeichnung PAMINA (*Privilege Administration and Management INfrAstructure*). Die den jeweiligen Bedürfnissen anpassbare und erweiterbare Komponenten des Systems namens *PAMINA Administration Server, Privilege Database* und *Certificate Verifier* ermöglichen eine flexible Einführung in eine Betriebsumgebung.

Die Tragfähigkeit von PAMINA wurde bei der Absicherung eines an der Universität Karlsruhe entwickelten internetbasierten Lernsystems mit der Bezeichnung ed.tec (*educational technologies*) demonstriert.

1.3 Überblick

Die vorliegende Arbeit gliedert sich in insgesamt acht Kapitel mit folgendem Inhalt:

- Im nächsten Kapitel werden die Grundlagen für die Arbeit geschaffen. Hierbei werden die wichtigsten in der Arbeit verwendeten Begriffe und Basistechniken erklärt. Die operativen Prozesse im Umfeld der zertifikatsbasierten Zugriffskontrolle sowie die dabei auftretenden in der Arbeit behandelten Probleme werden durch Beispiele verdeutlicht. Diese Probleme sind vor allem mit der Gültigkeitsprüfung evtl. durch Rechtedelegierung entstandener Attributszertifikate verbunden.

- Im dritten Kapitel werden aus der Literatur bekannte, teilweise auch standardisierte Techniken zur Überprüfung von Zertifikaten vorgestellt und bezüglich ihrer Performanz und Sicherheit im Kontext der Zugriffskontrolle analysiert. Diese Verfahren basieren auf der Annahme, dass die Zertifikate zu einem gegebenen Zeitpunkt möglicherweise Daten beinhalten, welche den zum Zeitpunkt der Zertifizierung herrschenden Bedingungen nicht mehr entsprechen. Aus diesem Grund wird anhand von zusätzlichen aktuelleren Daten entschieden, ob ein Zertifikat immer noch als gültig akzeptiert werden kann.

- Im vierten Kapitel wird ein Lösungsansatz aufgezeigt, um sämtliche Attributszertifikate in einem System mit einer entsprechenden Häufigkeit zu aktualisieren, so dass auf die oft aufwändige Prüfung von zusätzlichen Daten verzichtet werden kann. Die erreichbare Häufigkeit der Aktualisierungen wird maßgeblich von der Anzahl der jeweiligen zu signierenden Attributszertifikaten eingeschränkt. Durch die Verwendung von so genannten *Certification Verification Trees* (CVT) [GGM00] kann diese Häufigkeit der Aktualisierungen erhöht werden. CVTs stellen einen Anwendungsfall für so genannte Merklesche Authentifizierungsbäume dar [Mer90]. Bei der Verwendung von CVTs treten jedoch implementierungs- und sicherheitstechnische Probleme auf, welche gelöst werden. Ein wesentliches Ziel hierbei ist zu verhindern, dass eine einen CVT speichernde Datenbank die Existenz eines durch den CVT erfassten Zertifikats unentdeckt abstreiten kann. Die zunächst vorgeschlagenen Lösungen kommen ohne Änderungen an CVTs aus, führen aber zu einem erhöhten Aufwand während der Zertifikatsprüfung.

- Im fünften Kapitel werden die Prinzipien diskutiert, auf deren Grundlage Vollständigkeitsbeweise anhand eines CVT konstruiert werden können. Ein Vollständigkeitsbeweis ermöglicht einer Datenbank zu bestätigen, dass keine Daten, welche bestimmte Suchkriterien erfüllen, zurückgehalten wurden. Außerdem kann mit Hilfe eines solchen Beweises überprüft werden, ob die ausgehändigten Daten in einer unveränderten und authentischen Form vorliegen. Um solche Beweise möglichst effizient zu erzeugen, werden Änderungen an CVTs beziehungsweise den diesen zugrunde liegenden Authentifizierungsbäumen vorgeschlagen. Ausgehend von diesen Überlegungen wird eine Datenstruktur mit der Bezeichnung *Improved Certification Verification Tree (I-CVT)* konstruiert.

Dieser kann als eine sicherheitstechnische Erweiterung von in Datenbanken häufig zur Datenindizierung verwendeten B^+-Bäumen betrachtet werden. Aus diesem Grund können I-CVTs und die damit verknüpften Algorithmen neben der zertifikatsbasierten Zugriffskontrolle auch in verschiedenen anderen Bereichen ihre Anwendung finden.

- Im sechsten Kapitel werden Verfahren zur Beschleunigung der Ursprungsprüfung durch Delegierung entstandener Attributszertifikate diskutiert. Ein Konzept sieht hierfür eine Erweiterung der standardisierten Zertifikatsformate vor. Eine andere Idee ist die Auslagerung der Ursprungsprüfung an eine hierfür zuständige zentrale Stelle, die aber dadurch eine praktisch uneingeschränkte Macht im System erhält. Durch die erstmals in [Aur99] diskutierte Technik der Zertifikatsreduktion kann die Ursprungsprüfung vereinfacht werden. Problematisch ist jedoch die Umsetzung dieser Technik in einer verteilten Umgebung. Durch die in dieser Arbeit vorgeschlagene Methodik mit der Bezeichnung Offline-Delegierung vereinfacht sich die Ursprungsprüfung ebenso, ohne dass die aufgezeigten Probleme entstehen.

- Im siebten Kapitel wird das System PAMINA für das Management von Attributszertifikaten vorgestellt. Es erfolgt eine Beschreibung der wichtigsten Systemkomponenten sowie der Abläufe innerhalb des Systems. Dieses Kapitel weist auch die praktische Einsetzbarkeit und Tragfähigkeit der Konzepte nach, indem PAMINA, integriert in einem realen Anwendungsszenario, dort für die sichere Zugriffskontrolle sorgt.

- Die Arbeit schließt im achten Kapitel mit einem Ausblick, in dem die Ergebnisse bewertet und einige offene Fragen diskutiert werden.

2 Grundlagen

Zunächst werden in Kap. 2.1 die Zugriffskontrolle sowie als deren administrative Grundlage die Autorisierung im Allgemeinen vorgestellt. Kap. 2.2 und 2.3 widmen sich hauptsächlich der Beantwortung der folgenden Fragen: Was sind Attributszertifikate? Wie können diese sicher ausgestellt und geprüft werden? Wie werden mit ihrer Hilfe strukturierte Autorisierungsdaten für die Zugriffskontrolle bereitgestellt? Wie lassen sich Attributszertifikate in den Zugriffskontrollprozess integrieren? Welche Probleme entstehen dadurch in einem abgesicherten System?

2.1 Zugriffskontrolle und Autorisierung in verteilten Systemen

Die Zugriffskontrolle (engl. *Access Control*) in einem verteilten Informationssystem ist ein operativer Prozess mit zwei wesentlichen Zielen: Es muss entschieden werden, ob einem aktuell zugreifenden Subjekt der Zugriff auf eine bestimmte Systemressource gestattet oder verweigert wird. Außerdem muss die jeweilige Entscheidung wirksam durchgesetzt werden. Im Rahmen dieser Arbeit werden vor allem Aspekte der Zugriffskontrollentscheidungen näher betrachtet. Die Qualität der Zugriffskontrolle lässt sich daran messen, ob es gelingt, nur berechtigte Zugriffe zuzulassen und alle anderen Zugriffsversuche zu verhindern. Angriffe gegen die Zugriffskontrolle versuchen, das System entweder zum Zurückweisen legitimer oder zum Erlauben unberechtigter Zugriffe zu veranlassen.

Um zu entscheiden, ob ein Zugriffsversuch in einem gegebenen System berechtigt ist, muss die Menge aller denkbaren Zugriffe innerhalb dieses Systems bestimmt werden. Dies kann durch die Spezifizierung des Tupels *{S, O, T}* erfolgen [HRU76].
S bezeichnet hier die Menge aller Subjekte (z.B. Menschen, Prozesse, Prozessoren, Rechner usw.), die auf das System zugreifen könnten. *O* symbolisiert die Menge der von der Zugriffskontrolle geschützten atomaren Objekte (Systemressourcen). Beispielsweise können Speicheradressbereiche, Dateien, Festplattenblöcke oder TCP/IP-Ports als atomare Objekte in einem System identifiziert werden. *T* steht für die Menge der ebenso atomaren Zugriffstypen, welche durch Subjekte aus *S* an den Objekten aus *O* durchgeführt werden können. Beispielsweise können in einem Dateisystem Operationen wie „*Verzeichnis löschen*“ oder „*Datei lesen*“ als atomare Zugriffstypen gelten.
Ein Tripel *(s, o, t)* mit $s \in S$, $o \in O$ und $t \in T$ wird als Zugriffsweg (engl. *Access Path*) bezeichnet. In einem System können entweder die erlaubten und/oder die verbotenen Zugriffswege (engl. *Permission* resp. *Denial*) explizit angegeben werden. Solche Zugriffswege können in der Zugriffsrelation *ZR* zusammengefasst werden. Diese ist die Menge aller Tripel *(s, o, t)* mit $s \in S$, $o \in O$, $t \in T$, für die gilt, dass *(s, o, t)* ein erlaubter (oder unerlaubter) Zugriffsweg ist. Die informelle Bedeutung eines solchen Tripels aus *ZR* ist, dass das Subjekt *s* berechtigt (oder unberechtigt) ist, die Operation *t* am Objekt *o* auszuführen. Die übliche Darstellungsform einer Zugriffsrelation *ZR* ist die Zugriffskontrollmatrix *ZM* [Lam71] [McL94]. Für die zm_{ij} Elemente dieser Matrix gilt $zm_{ij}=\{t_1, ..., t_k\}$, wobei $1 \le i \le |S|$, $1 \le j \le |O|$, $0 \le k \le |T|$ und $(s_i, o_j, t_1), ..., (s_i, o_j, t_k) \in ZR$. Die Elemente einer Zugriffskontrollmatrix enthalten also jeweils eine Liste von Zugriffstypen, die einem gegebenen Subjekt an einem gegebenen Objekt erlaubt oder eben unerlaubt sind.

Die Betrachtung solch statischer Zugriffswege reicht in einem System nicht aus, wenn sich dynamisch ändernde Faktoren einen Einfluss auf die Zugriffsentscheidungen haben. Dies kann das Einführen eines Prädikats zu den einzelnen Zugriffswegen notwendig machen [Den82]. Ein bedingter Zugriffsweg $zw_i=(s, o, t)_{Pi}$ wird genau dann gültig, wenn das Prädikat $P_i(x_1, x_2, ..., x_n)$ über den Größen $x_1, x_2, ..., x_n$ wahr ist. Für x_i können beispielsweise Datum und Uhrzeit oder aktuelle Auslastung des Systems von Interesse sein. Ausgehend von dem bisher Gesagten kann der Begriff Zugriffskontrolle präzisiert werden:

Def. 2-1: Zugriffskontrolle

Die Zugriffskontrolle stellt in einem System die Einhaltung der Zugriffsrelation ZR sicher. Hierbei muss bei jedem Zugriff (s, o, t) für s aus S, o aus O und t aus T geprüft werden, ob (s, o, t) ∈ ZR.

2.1.1 Prozess der Zugriffskontrolle

Die Realisierung der Zugriffskontrolle in einem System bedarf der Etablierung eines (linearen) Prozesses, welcher aus mehreren Teilprozessen aufgebaut werden kann. In Abb. 2-1 wird der schematische Prozess der Zugriffskontrolle gezeigt. Der Prozessablauf beginnt, sobald ein Zugriffsversuch das System erreicht. Wird am Ende des Prozesses der Zugriff verweigert, bekommt der Zugreifende in der Regel eine Fehlermeldung. Ansonsten führt das Zielsystem die gewünschte Aktion durch. Bei der Vorstellung der Teilprozesse wird angenommen, dass die Zugriffsversuche im Sinne von Def. 2-1 der Strukturierung *(Subjekt, Objekt, Zugriffstyp)* folgen. Beispielsweise kann der Zugriffsversuch *(„Bob“, „foo.com/letters/loveletter.htm“, „delete“)* einen Webserver erreichen.

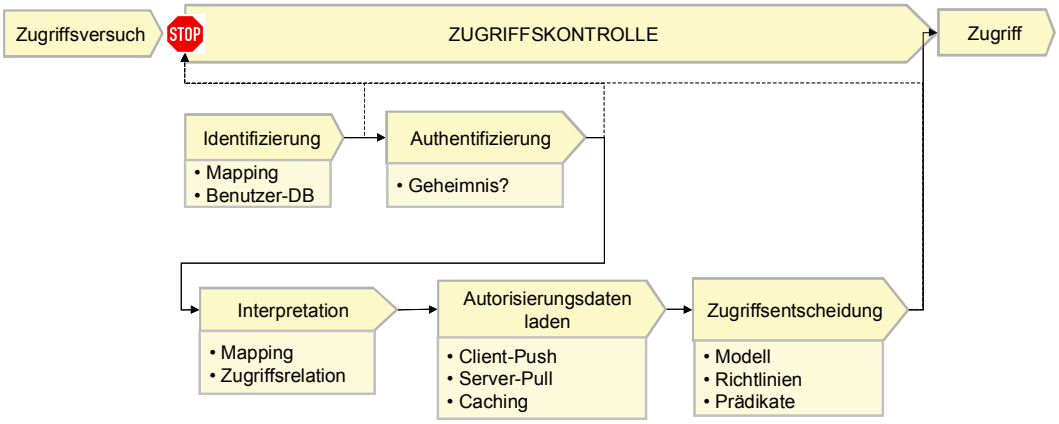


Abb. 2-1 Prozessablauf der Zugriffskontrolle

- **Identifizierung:** Um den Zugriff zu erlauben oder zu verhindern, muss zunächst eine Identifizierung des Zugreifenden erfolgen [JH94]. Dabei wird geprüft, ob das *Subjekt* dem System überhaupt bekannt ist. Dies kann durch eine Suche nach dem Zugreifenden in einer Liste bekannter Benutzer (Benutzerdatenbank) erfolgen. Dazu muss häufig eine Abbildung (engl. *Mapping*) des verwendeten Benutzernamens (hier *„Bob“*) auf eine der Zugriffskontrolle verständliche Kennung, wie z.B. *„12345“* erfolgen. Ist der Zugreifende dem System unbekannt, wird der Zugriff verweigert, ansonsten wird der Prozess fortgesetzt.

- **Authentifizierung:** Für die darauf folgende Authentifizierung des *Subjekts* muss es seine Identität üblicherweise mit Hilfe eines Geheimnisses, welches mit ihm eindeutig in Verbindung gebracht werden kann, bestätigen. Die heute am häufigsten verwendete Methode hierfür ist die Angabe und Prüfung eines Passworts. Kann das *Subjekt* seine Authentizität nicht nachweisen, wird der Zugriff verweigert.

- **Interpretation:** Ist die Authentifizierung erfolgreich, muss in vielen Fällen eine Interpretation des Zugriffsversuchs erfolgen. Dabei werden das genaue Zielobjekt, sowie die durchzuführende Aktion zwecks Vergleich mit der vorhandenen Zugriffsrelation ermittelt. Beispielsweise kann hier eine Transformation des virtuellen Namens des Zielobjekts *„foo.com/letters/loveletter.htm“* auf eine lokale Ressourcenadresse, wie z.B. */pub/letters/loveletter.htm* erforderlich sein. Bei Misserfolg der Interpretation wird der Prozess abgebrochen bzw. der Zugriff abgelehnt.

- **Laden von Autorisierungsdaten:** Die administrative Grundlage der Zugriffsentscheidung bilden Autorisierungsdaten, welche verschiedene Teilmengen der oben behandelten Zugriffsrelation sein können. Die Quelle der jeweilig benötigten Autorisierungsdaten ist typischerweise eine hierfür vorgesehene Datenbank im System. Diese Datenbank liefert auf entsprechend gestellten Anfragen die jeweilig gewünschten Autorisierungsdaten automatisch. Alternativ dazu kann auch das zugreifende *Subjekt* aufgefordert werden, die benötigten Autorisierungsdaten an die Zugriffskontrolle zu übermitteln. Diese Vorgehensweisen werden *Server-Pull* respektive *Client-Push* genannt.
  Bei der Beschaffung der Daten kann beispielsweise der Strategie des Minimalprinzips gefolgt werden. Dabei werden nur die Autorisierungsdaten an die Zugriffskontrolle übermittelt, anhand welcher der aktuelle Zugriffswunsch erlaubt oder verweigert werden kann. Im Gegensatz dazu steht der Ansatz, in dem beim ersten Zugriffsversuch des *Subjekts* gleich ein so genanntes Benutzerprofil (engl. *User Profile*) erzeugt wird. Dieses Profil beinhaltet sämtliche Zugriffswege, welche für das *Subjekt* vorgesehen wurden. Je nach Strukturierung und Granularität der Autorisierungsdaten können hier beispielsweise Gruppenmitgliedschaften oder Rollenzugehörigkeiten des *Subjekts* von Interesse sein.

- **Zugriffskontrollentscheidung:** Im letzten Schritt wird nun geprüft, ob die ermittelten Autorisierungsdaten des authentifizierten *Subjekts* tatsächlich ausreichen, um den Zugriff zuzulassen. Hierbei erfolgt gemäß vordefinierter Regeln beziehungsweise Richtlinien (engl. *Policy*) ein Abgleich der vorliegenden Autorisierungsdaten und des Zugriffswunsches. Erweist sich der Zugriffswunsch als zulässig, kann der Zugriff auf die jeweilige Zielressource erfolgen.

2.1.2 Modelle zur Realisierung der Zugriffskontrolle

Im ISO-Standard [ISO89] wurde ein Referenzmodell für Sicherheitsarchitekturen zunächst zur Absicherung von Netzwerkdiensten konzipiert. Die Vorschläge fanden eine weite Akzeptanz und liegen mittlerweile mehreren Industriestandards und folglich vielen Systemen zugrunde (siehe z.B. den ITU-Standard [ITU01] und den CORBA Sicherheitsstandard [OMG02]). Eine wesentliche Botschaft dieses Standards ist die funktionale Trennung der Sicherheitsdienste, wie z.B. der Zugriffskontrolle von der eigentlichen Nutzfunktionalität des Systems.

Dies ist vergleichbar mit dem Ansatz der so genannten Kern- und Zusatzdienste [HAN99] [MNM+00]. Dabei greift das abgesicherte System (Kerndienst) über wohldefinierte Schnittstellen auf die in diesem Sinne externen Sicherheitsdienste (Zusatzdienste) zu. Dies impliziert, dass ein abgesichertes System über keine explizit eingebaute Sicherheitsfunktionalität zu verfügen braucht.

Die Zugriffskontrolle kann auf verschiedene Art und Weise in ein, in dieser Hinsicht „passives" System integriert werden. Ein bekanntes Konzept sieht hierfür einen so genannten Referenzmonitor vor [Amo94]. Dieser ist sehr eng (technisch wie semantisch gesehen) an das abgesicherte System gekoppelt: Neben der Überprüfung der Zugriffsversuche auf ihre Zulässigkeit lenkt der Referenzmonitor sämtliche Zugriffe an die jeweilig betroffenen Zielressourcen um. Hierfür muss ein Referenzmonitor tief ins abgesicherte System eingreifen, was den Erwartungen des erwähnten ISO-Standards [ISO89] eigentlich nicht entspricht.
Im ITU-Standard mit dem Kennzeichen X.812 wird für die Erbringung der Zugriffskontrolle ein Zusammenspiel von verteilt erbrachten Zugriffskontrolldurchsetzungsfunktionen (engl. *Access Control Enforcement Function,* ACEF) und Zugriffskontrollentscheidungsfunktionen (engl. *Access Control Decision Function,* ACDF) vorgesehen (siehe Abb. 2-2) [ITU95]. Eine ACEF sorgt dabei für die technische Umsetzung der Zugriffskontrolle (Schalt- und Lenkfunktionalität) und ist, dem Referenzmonitor ähnlich, stark an die jeweilige Anwendung gekoppelt. Diese Funktion verlässt sich aber dabei auf Antworten von ihr logisch und evtl. auch räumlich getrennten ACDFs, welche in weiteren Teilen der Arbeit als Verifizierer (engl. *Verifier*) bezeichnet werden. Ein solcher Verifizierer übernimmt die technische Abwicklung des in Kap. 2.1.1 behandelten Zugriffskontrollprozesses, wobei die wirksame Durchsetzung der hieraus resultierenden Zugriffsentscheidungen Aufgabe der ACEF ist. Wichtig ist, dass eine Durchsetzungsfunktion sich ausschließlich auf die Antworten bestimmter hierfür vorgesehener Verifizierer verlässt.

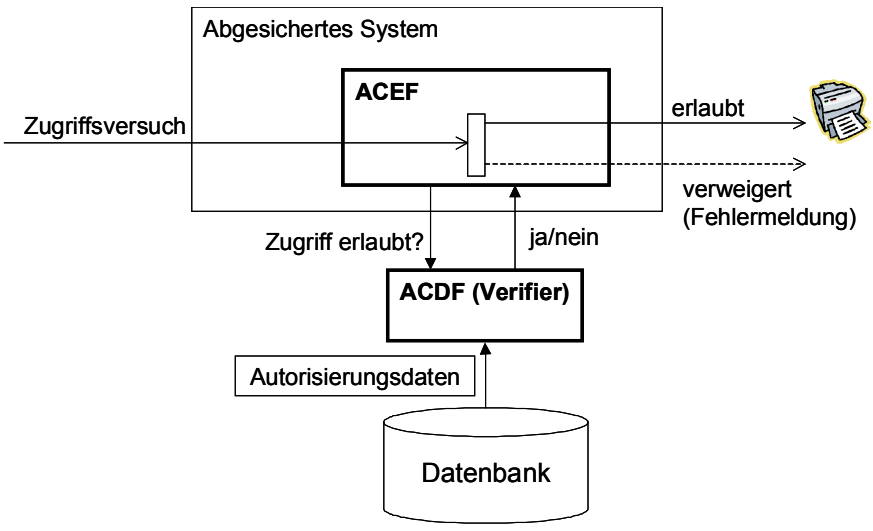


Abb. 2-2 Zugriffskontrolle mittels Entscheidungs- und Durchsetzungsfunktionen

Dieses Modell ermöglicht eine saubere funktionale Trennung zwischen den Zugriffskontrollentscheidungen und der Anwendung. Es fördert dadurch die komponentenorientierte Realisierung und Einbindung der Zugriffskontrolle in einem System. Zugriffskontrollentscheidungen können dabei auch durch das Zusammenwirken mehrerer verteilter ACDF getroffen werden.

Eine Verteilung dieser Funktionalität kann zu einer besseren Skalierbarkeit der Zugriffskontrolle führen, aber gleichzeitig die Komplexität des Systems erhöhen. Theoretisch kann natürlich auch eine einzige zentrale ACDF sämtliche Systeme innerhalb einer Betriebsumgebung bedienen.

Für die Platzierung der Zugriffskontrolle innerhalb von verteilten Systemen sind aus betrieblicher Sicht die folgenden Ansätze möglich [Bla99]:

1. **Serverseitig**: Die Zugriffskontrolle wird in den meisten „offenen Systemen“ auf der Seite der diensterbringenden Instanz (Server) platziert. Hier wird davon ausgegangen, dass sich die schützenswerten Ressourcen auf der Seite des jeweiligen Dienstgebers befinden. In solchen Systemen wird dem Zugreifenden (Client) keinerlei Vertrauen bezüglich der sicheren Dienstnutzung zuteil.

2. **Clientseitig**: In einigen Fällen kann es durchaus sinnvoll sein, die Zugriffskontrolle auf der Dienstnehmerseite auszuführen. Dadurch können diensterbringende Komponenten vom mit der Zugriffskontrolle verbundenen Aufwand (auf Kosten der Dienstnehmer) entlastet werden. Problematisch bleibt, sicher zu stellen, dass entfernte Client-Systeme, denen in dieser Hinsicht vertraut werden muss, stets korrekte Zugriffskontrollentscheidungen treffen, und dass bereits kontrollierte Dienstaufrufe das Zielsystem auf eine sichere Art und Weise erreichen. Angesichts dieser Probleme kommt diese Art Zugriffskontrolle nur in so genannten „geschlossenen“ Systemen in Frage.

2.1.3 Autorisierung: Die administrative Grundlage der Zugriffskontrolle

Die Autorisierung (engl. *Authorization*), welche auch als Rechtevergabe bezeichnet wird, bildet die administrative Grundlage der operativen Zugriffskontrolle. Die Aufgabe der Autorisierung ist die Erstellung und durchgängige Pflege der Zugriffsrelation und folglich der aus dieser abgeleiteten in die Zugriffskontrolle tatsächlich einfließenden Autorisierungsdaten. Die Autorisierung wird wie folgt definiert:

Def. 2-2: Autorisierung

Die Autorisierung sorgt für die korrekte und sichere Durchführung der Operationen Einfügen e und Löschen l einzelner Elemente der Zugriffskontrollrelation ZR.

Laut dieser Definition müssen Elemente $(s, o, t) \in ZR$, die modifiziert werden sollten, zunächst aus der Zugriffsrelation entfernt und durch jeweils ein neues Element ersetzt werden. Die Grundlage der (kontrollierten) Autorisierung kann die Vergaberelation $VR=\{(s, o, t) | \ s \in S_A \subseteq S, \ o \in ZR, \ t \in \{e, l\} \}$ bilden. Diese legt fest, welche Subjekte aus der Menge $S_A \subseteq S$ (häufig als Administratoren bezeichnet) welche Tripel der Zugriffsrelation einfügen oder löschen dürfen. Ein paradox erscheinendes Problem hierbei ist, eindeutig zu bestimmen, welches Subjekt welche Operationen an der Vergaberelation durchführen darf. Analog zu Zugriffsrelationen können Vergaberelationen durch Matrizen veranschaulicht werden.

Es existieren unterschiedliche, in der Praxis etablierte Strategien für die Autorisierung in einem System. Ausführlichere Beschreibungen der nachfolgend erläuterten Modelle finden sich in [Amo94], [McL94], [SS94] und [Sum97].

- **Diskrete Autorisierung:** Zugriffskontrollsysteme basieren heute überwiegend auf der diskreten Autorisierung (engl. *Discretionary Access Control,* DAC). In diesem Modell werden Einträge der Zugriffsrelation objektbezogen gepflegt. Das heißt, es wird für jedes zu schützende Objekt im System individuell und meist lokal festgelegt, welche Operationen einzelne Subjekte an diesem Objekt durchführen dürfen. Dieses Modell wird häufig in Kombination mit dem so genannten Eigentümerprinzip verwendet:
  Der Besitzer (engl. *Owner*) eines Objekts $o \in O$ darf sämtliche Elemente der Zugriffsrelation *ZR* gemäß der Selektion $\sigma_{O=o}(ZR)$ einfügen beziehungsweise löschen. Die Pflege der Vergaberelation ist hier die Aufgabe von Administratoren, die primär die Besitzer der einzelnen Objekte bestimmen.
  DAC-Modelle sind vor allem wegen ihrer hohen Flexibilität beliebt. Sie weisen aber Schwächen bei der Durchsetzung globaler Zugriffsrichtlinien in einem System auf: Es ist schwierig zu kontrollieren, ob Objektbesitzer oder lokale Administratoren Zugriffsrechte tatsächlich im Einklang mit einer beispielsweise unternehmensweiten Richtlinie vergeben.

- **Systembestimmte Autorisierung**: Im Gegensatz zur diskreten Autorisierung haben systembestimmte Modelle (engl. *Mandatory Access Control,* MAC) das Ziel, systemweite Richtlinien wirksam durchzusetzen. Dabei können ausgehend von existierenden Hierarchiestrukturen unter den Subjekten und Objekten systemweite Regeln definiert werden, von denen niemand während der Rechtevergabe und Zugriffskontrolle abweichen kann.
  Solche Modelle wurden ursprünglich im militärischen Umfeld entwickelt, um die Vertraulichkeit sensibler Dokumente (Dateien) zu schützen. Beispielsweise werden im Bell-LaPadula-Modell [BL73] Subjekte sowie Objekte geordneten Sicherheitsklassen zugeordnet, wie z.B. „*Public*“, „*Secret*“, „*Top Secret*“. In Abhängigkeit von der jeweiligen Sicherheitsklasse des Objekts und des zugreifenden Subjekts gibt eine Regelsammlung die erlaubten Zugriffstypen an. So darf ein als „*Secret*“ eingestuftes Subjekt nur Objekte mit der Einstufung „*Public*“ oder „*Secret*“ aber keine mit „*Top Secret*“ lesen.
  Im kommerziellen Sektor hat sich dieses Modell bis jetzt nur mit mäßigem Erfolg durchgesetzt [Lip82] [Lee88] [DG97], da im zivilen Bereich meistens keine strikten Hierarchien herrschen, was die Erzeugung von Sicherheitsklassen erschwert.

Beide Modelle können in Kombination mit verschiedenen Administrationsparadigmen verwendet werden:

- **Zentrale Autorisierung**: In diesem Modell gibt es eine einzige systemweit anerkannte Administrationsinstanz, welche sämtliche Autorisierungsdaten festlegt und pflegt. Dieser Ansatz ist theoretisch zwar gut geeignet, um systemweite Richtlinien durchzusetzen, ist aber in komplexen, heterogenen Umgebungen unpraktikabel.

- **Dezentrale Autorisierung**: Es ist in den meisten Fällen sinnvoller, eine Aufgabenteilung innerhalb der Autorisierung vorzunehmen, um einerseits die Arbeitslast der Administratoren zu verteilen und andererseits Fehlkonfigurationen mangels Fachkenntnisse zu vermeiden.

Eine sichere Dezentralisierung der Rechtevergabe kann auf verschiedene Art und Weise umgesetzt werden:

- **Rechtedelegierung**: Eine Methode zur schrittweise Dezentralisierung ist die Delegierung der Rechte. Hierbei kann die Zugriffsrelation zunächst durch einen einzigen Administrator initialisiert werden. Er kann seine Rechte teilweise oder vollständig an würdige Subjekte delegieren, die dann wiederum ihre Rechte weitergeben können.

- **Eigentümerbasierte Autorisierung**: Wie schon gesagt, werden in diesem Modell jedem Objekt im System ein oder mehrere Subjekte (Eigentümer) zugeordnet, welche die entsprechenden Teile der Zugriffsrelation autonom pflegen. Heute wird dieses Modell vor allem in Betriebssystemen und Datenbanken eingesetzt, wobei der Eigentümer eines Objekts zunächst dessen Erzeuger ist. Ähnlich zu anderen administrativen Rechten ist es natürlich denkbar, Eigentümerrechte an Andere zu delegieren.

- **Mehr-Augen-Prinzip**: Bei der Durchführung besonders kritischer Autorisierungsvorgänge kann es sinnvoll sein, die „Macht" einzelner Administratoren einzuschränken. Entsprechend sichere Verfahren (z.B. basierend auf Verschlüsselung) können dafür sorgen, dass solche Aktionen nur in Anwesenheit oder in Kenntnis mehrerer (m) Administratoren aus einer Gesamtbelegschaft (n) getätigt werden können [Sha79].

Eine strukturierte Ablage der Zugriffsrelation (und/oder Vergaberelation) kann verschiedene operative sowie administrative Vorteile mit sich bringen. Es können dadurch beispielsweise bestimmte Daten, wie z.B. sämtliche Berechtigungen eines Benutzers, schneller gefunden und damit die Zugriffskontrolle beschleunigt werden. Durch Strukturierung kann auch die Übersichtlichkeit der Autorisierungsdaten erhöht und damit deren Pflege erleichtert werden. Gängige Praxis zur Strukturierung sind verschiedene Gruppierungen der Einträge einer Zugriffsrelation *ZR*.

- **Fähigkeitsliste**: Fähigkeitslisten (engl. *Capability Lists*, CL) sind von Vorteil, wenn die Zugriffskontrolle identitätsbasiert, beispielsweise auf Basis eines Benutzerprofils erfolgen soll. Um hier eindeutige Zugriffsentscheidungen zu treffen, müssen alle existierenden Berechtigungen des Zugreifenden zur Verfügung stehen. Die Fähigkeitsliste CL_s eines Subjekts s entsteht durch die Selektion und Projektion $CL_s=\pi_{O,T}(\sigma_{S=s}(ZR))$ auf *ZR*. Sie listet also alle die für s zugänglich gemachten Objekt-Zugriffstyp-Paare auf, welche man Fähigkeiten oder auch Privilegien (engl. *Privilege*) beziehungsweise Rechte (engl. *Right*) nennt. Ein weiterer Vorteil von Fähigkeitslisten ist der reduzierte Aufwand, wenn Operationen, wie z.B. das Einfügen/Löschen eines Subjekts samt seiner Rechte ausgeführt werden müssen.

- **Zugriffskontrollliste**: Zugriffskontrolllisten (engl. *Access Control List*, ACL) werden hingegen eingesetzt, wenn die Zugriffskontrolle beziehungsweise die Autorisierung objektbezogen erfolgt. Eine Zugriffskontrollliste ACL_o eines Objekts o entsteht durch die Selektion und Projektion $ACL_o=\pi_{S,T}(\sigma_{O=o}(ZR))$ auf *ZR*. Sie beinhaltet also alle Zugriffstypen sämtlicher Subjekte, welche auf o zugreifen dürfen.

Während der Zugriffskontrolle kann die ACL zum anvisierten Objekt nach dem erforderlichen Subjekt-Zugriffstyp-Paar effizient durchsucht werden. Zugriffskontrolllisten sind des Weiteren vorteilhaft, falls häufig Objekte zusammen mit sämtlichen passenden Elementen in (aus) *ZR* eingefügt (entfernt) werden.

Zur Reduzierung der Anzahl und/oder Länge einzelner Fähigkeits- respektive Zugriffskontrolllisten ist die Bildung von so genannten Rollen (engl. *Role*), Benutzergruppen (engl. *User Group*) und Objektgruppen (engl. *Domain*) genannt sinnvoll:

- **Rolle**: Rollen sind Fähigkeitslisten, welche gleichzeitig mehreren Subjekten zugeordnet werden können. Eine Rolle listet dementsprechend Objekt-Zugriffstyp-Paare auf, ohne Angabe der Mitgliedersubjekte dieser Rolle. Subjekte, welche identische Fähigkeiten zugewiesen bekommen sollen, werden entsprechend erzeugten Rollen mittels der Zuordnungsrelation $RZ=\{(s, r)| s \in S, r \in R\}$ zugeordnet. Dabei bezeichnet *R* die Menge aller Rollen im System. Beispielsweise können mehrere Personen innerhalb einer Firma der Rolle „Sekretärin“ oder „Chef“ zugeordnet werden. Mit Hilfe von Rollen lässt sich vor allem die Anzahl redundanter Fähigkeitslisten im System senken. Erfolgt die Zugriffskontrolle anhand von Rollen, so kann mit Hilfe der Zuordnungsrelation *RZ* festgestellt werden, welche Rollenzugehörigkeiten der Zugreifende hat. Anschließend kann innerhalb dieser Rollen nach den zum jeweiligen Zugriff erforderlichen Fähigkeiten gesucht werden. Rollen können auch geschachtelt werden, wodurch Rollenhierarchien entstehen: Die einer dominanten Rolle zugeordneten Subjekte bekommen dabei automatisch auch die Fähigkeiten der dieser Rolle untergeordneten Rollen [SC+96].

- **Benutzergruppe**: Eine Benutzergruppe $g \in G$ mit $G \subseteq \wp(S)$ ist eine Menge von Subjekten, die auf die gleiche Art und Weise auf ein bestimmtes Objekt *o* zugreifen dürfen. Für jedes Subjekt $s_i \in g$ gilt, dass $(s_i, o, t_1), (s_i, o, t_2), \ldots, (s_i, o, t_k) \in ZR$. Durch die Einführung von Benutzergruppen kann die Länge der Zugriffskontrolllisten im System reduziert werden: In diesen können statt einzelner Subjekte die diese zusammenfassenden Benutzergruppen reflektiert werden. Um die Gruppenzugehörigkeit(en) eines Subjekts *s* zu ermitteln, kann eine Zuordnungsrelation $GZ=\{(s, g)| s \in S, g \in G\}$ hilfreich sein. Ähnlich zu den Rollen ist es möglich, Benutzergruppen zu schachteln, um etwa Hierarchien abzubilden.

- **Objektgruppe**: Es kommt häufig vor, dass auf bestimmte Objekte des Systems mehrere (evtl. alle) Subjekte die gleichen Zugriffsmöglichkeiten erhalten sollen. Um in solchen Fällen die Anzahl (redundanter) Zugriffskontrolllisten zu reduzieren, können Objektgruppen (Domänen) $og \in OG$ mit $OG \subseteq \wp(O)$ gebildet werden. Auch Rollen können sich auf Objektgruppen statt auf Einzelobjekte beziehen.

Bezüglich der Verteilung von Autorisierungsdaten, beziehungsweise wie diese zum Zugriffskontrollsystem gelangen, kann zwischen zwei Ansätzen unterschieden werden:

- **Client-Push-Modell**: In diesem Modell sind Autorisierungsdaten, die sich auf ein bestimmtes Subjekt beziehen, wie z.B. Fähigkeitslisten oder Rollenzugehörigkeiten, auch tatsächlich in Besitz des Subjekts. Beim Zugriff „schiebt“ (engl. *pushes*) das Subjekt die jeweils benötigten Daten zur Zugriffskontrolle, die diese auswertet.

Die Client-Systeme kann man sich hier als Komponenten einer (massiv) verteilten Autorisierungsdatenbank vorstellen.
Ein Vorteil hierbei ist, dass während der Zugriffskontrolle kein drittes System kontaktiert werden muss, um Autorisierungsdaten zu erhalten. Dies kann zur Reduzierung der Kommunikationskosten führen. Als ein weiterer Vorteil wird häufig der gute Datenschutz genannt: Die Zugriffskontrolle kann im Prinzip nur solche Daten erhalten, die der Zugreifende ihr zukommen lässt [Sam02].
Fraglich ist jedoch, woher der Zugreifende weiß, welche Daten für einen bestimmten Zugriff erforderlich sind. Wird er etwa von der Zugriffskontrolle aufgefordert, bestimmte Daten zu übersenden, so muss er dem Zugriffskontrollsystem bezüglich der Minimalität solcher Anfragen vertrauen. Des Weiteren sind Client-Systeme häufig unbefriedigend gesichert, d.h. Unbefugte können die zu schützenden Daten sichten oder stehlen. Außerdem müssen Client-Systeme mit (gültigen) Autorisierungsdaten versorgt werden, um diese später an die Zugriffskontrolle schicken zu können. Dies kann aber kostenintensiv und auch aus Erreichbarkeitsgründen problematisch sein.

- **Server-Pull-Modell**: Angesichts obiger Probleme erscheint es in den meisten Umgebungen sinnvoller, Autorisierungsdaten in einer hierfür vorgesehenen, entsprechend verwalteten und geschützten Datenbank zu halten. Die (serverseitige) Zugriffskontrolle „zieht“ (engl. *pull*) dabei die zur jeweiligen Zugriffsentscheidung benötigten Daten von der Datenbank.
  Dieses Modell impliziert, dass Client-Systeme von der Verwaltung der Autorisierungsdaten verschont bleiben [NEA02]. Sie sind einfacher zu implementieren, da sie in den Zugriffskontrollprozess nur als Initiator eingebunden werden. Außerdem können durch Caching-Mechanismen die Kommunikationskosten sowie die Last seitens der Datenbank reduziert werden. Eine Gefahrenquelle ist die Abhängigkeit der Zugriffskontrolle von der Verfügbarkeit der Datenbank, die dementsprechend robust und ausfallsicher aufgebaut werden muss.

2.2 Attributszertifikate und relevante Techniken

Im Folgenden werden zuerst einige Verfahren aus dem breiten Spektrum der Sicherheitstechnik vorgestellt, auf welche in dieser Arbeit stets zurückgegriffen wird. Dem folgt die Einführung des Begriffs „Attributszertifikat“ sowie die Vorstellung eines standardisierten Formats für Attributszertifikate.

2.2.1 Kryptographische Hashfunktionen

Der Ursprung heutiger kryptographischer Hashfunktionen liegt in den frühen 80-er Jahren (siehe z.B. [PBD97]). Diese Funktionen gelten als kryptographische Primitive und sind vor allem beim Schutz der Integrität und Authentizität von Daten von Nutzen.

Def. 2-3: Hashfunktion

Sei X eine Menge von Bitfolgen beliebiger Länge und Y eine Menge von Bitfolgen der festen Länge l. Eine Hashfunktion h: $X \rightarrow Y$ ist dann definiert als eine Pseudozufallsfunktion, die Elemente x aus X auf Elemente y aus Y abbildet.

Ein Element $x \in X$ wird dabei als Urbild, und ein Element $y \in Y$ mit $y=h(x)$ wird als Hashwert (Fingerabdruck, Prüfsumme, etc.) des Urbilds x bezeichnet.

In kryptographischen Anwendungen eingesetzte Hashfunktionen müssen strengen Kriterien entsprechen:

1. **Einwegfunktionalität:** Es soll mit unvertretbarem Aufwand verbunden sein, zu h eine Umkehrfunktion $h^{-1}: Y \rightarrow X$ anzugeben. Außerdem soll es schwierig sein, zu einem gegebenen Hashwert y ein Urbild x zu finden, so dass $h(x)=y$, falls zu y im Voraus kein passendes Urbild bekannt war. Die Wahrscheinlichkeit ε, ein passendes Urbild $x \in X_0$ innerhalb einer beliebig gewählten Teilmenge der möglichen Urbilder $X_0 \subseteq X$ zu finden, beträgt im so genannten „Zufallsorakel-Modell" $\varepsilon=1-(1-1/2^l)^q$, wobei $q=|X_0|$ gilt [BR93]. Ein passendes Urbild mit einer Wahrscheinlichkeit von $\varepsilon=½$ zu finden, wird also aufwändig, wenn l groß genug gewählt wird. Schätzungen zufolge sollte deshalb heute eine Hashwertlänge von $l>90$ gewählt werden [Sch00].

2. **Schwache Kollisionsresistenz:** Es soll ebenfalls schwierig sein, für ein gegebenes $x \in X$ ein zweites Urbild $x' \in X$ zu finden, so dass $h(x)=h(x')$. Da die Kardinalität der Menge der möglichen Urbilder unendlich ist, wird es (unendlich) viele x' mit dieser Eigenschaft geben. Eine Hashfunktion erfüllt dieses Kriterium also dann, wenn die Wahrscheinlichkeit, dass $h(x)=h(x')$ zutrifft, für alle $x' \in X$ gleich (klein) bleibt. Eine schwach kollisionsresistente Hashfunktion ist nicht notwendigerweise auch eine Einwegfunktion, und umgekehrt, aus der Einwegfunktionalität folgt keine Kollisionsresistenz [Sti02].

3. **Starke Kollisionsresistenz:** Das systematische Auffinden von zwei beliebigen Werten $x \in X$ und $x' \in X$, die den gleichen Hashwert $h(x)=h(x')$ ergeben, soll ausgeschlossen sein. Ausgehend vom „Geburtstagsparadoxon" [Sti02] beträgt die Anzahl der Versuche für eine Trefferwahrscheinlichkeit von $\varepsilon=½$ näherungsweise $1.17 \times 2^{l/2} \approx 2^{0.3+l/2}$. Hieraus und aus Punkt 1 oben folgt, dass heute eine Hashwert-Länge von etwa $l=2 \times 90$ Bit wünschenswert ist.

Def. 2-4: Kollisionsresistente Hashfunktion [MOV96]

Eine schwach und stark kollisionsresistente Hashfunktion mit Einwegfunktionalität heißt kollisionsresistente Hashfunktion.

In kommerziellen Anwendungen eingesetzte Hashfunktionen sind MD2 [Kal92], MD4 [Riv91], MD5 [Riv92], SHA-1 [NIST95], SHA-2 [NIST01] und RIPEMD-160 [DBP96]. MD2, MD4 und MD5 liefern einen Hashwert von jeweils 128 Bit Länge, welche für eine mittel- bis langfristige Datensicherung nicht mehr ausreicht. Außerdem bietet beispielsweise MD4 bezüglich der starken Kollisionsresistenz keine hinreichende Sicherheit mehr [Dob96]. RIPEMD-160 sowie SHA-1 generieren Hashwerte der Länge 160 Bit. Durch die Hashwertlänge von wahlweise 256, 384 oder 512 Bit ist SHA-2 womöglich die beste Wahl für sicherheitskritische Anwendungen.

2.2.2 Digitale Signaturen

Eine seit den Anfängen der Menschheitsgeschichte verwendete Methode zur Geheimhaltung (geschriebener) Informationen ist die Verschlüsselung. Traditionelle so genannte Secret-Key-Verfahren benötigen zur Verschlüsselung eines Datums sowie zur Entzifferung der Chiffre entweder identische Schlüssel oder unterschiedliche Schlüssel, welche aber aus demselben Geheimnis abgeleitet werden.

Die Sicherheit solcher Verfahren, wie z.B. Rijndael [DR00], DES (siehe z.B. [Sti02]), IDEA [Lai92] oder CAST [AT93] hängt dementsprechend unmittelbar von der effektiven der Geheimhaltung der Schlüssel ab.
In den 1970-er Jahren wurde die Idee präsentiert, verschiedene beziehungsweise voneinander stärker entkoppelte Schlüssel zur Verschlüsselung und Dechiffrierung zu benutzen [DH76] [Mer78]. In einem solchen System benötigt der Benutzer *Bob* zwei Schlüssel. Der öffentliche Schlüssel *(*engl. *Public Key)* wird benutzt, um für *Bob* Nachrichten zu verschlüsseln: *Alice* berechnet $C=E_{Bob}(M)$ um die Nachricht *M* für *Bob* zu verschlüsseln, und sendet die Chiffre *C* an *Bob*.
Hierbei bezeichnet $E_{Bob}(M)$ die mit dem öffentlichen Schlüssel von *Bob* parametrisierten Chiffrierfunktion (engl. *Encryption*), welche auf *M* angewendet wird. Mit Hilfe seines privaten Schlüssels (engl. *Private Key)* kann *Bob* die für ihn chiffrierte Nachricht durch $M=D_{Bob}(C)$ ermitteln. Hierbei bezeichnet $D_{Bob}(C)$ die mit dem privaten Schlüssel von *Bob* parametrisierte Dechiffrierfunktion (engl. *Decryption*), welche auf *C* angewendet wird. Ein Vorteil dieses Verfahrens ist, dass kein sicherer Kanal etabliert werden muss, um Verschlüsselungsschlüssel auszutauschen. Statt dessen kann *Bob* seinen öffentlichen Schlüssel an einem für andere Teilnehmer zugänglichen Ort ablegen oder auf Wunsch direkt zuschicken. Die Sicherheit des Verfahrens hängt mit der Geheimhaltung der privaten Schlüssel zusammen, welcher entsprechend geschützt werden muss.
An praktisch einsetzbare Public-Key-Verfahren werden folgende allgemeine Anforderungen gestellt [MOV96]: Es muss rechnerisch unmöglich sein, den privaten Schlüssel aus dem passenden, bekannten öffentlichen Schlüssel abzuleiten. Gelingt das einem Angreifer, so kann er verschlüsselte Nachrichten dechiffrieren. Zu jeder Nachricht *M* muss genau eine eindeutige Chiffre *C* existieren, so dass $M=D_X(E_X(M))$ gilt. Das systematische Erraten der Klartextnachricht zu einer bekannten Chiffre muss ausgeschlossen bleiben. In Kenntnis des öffentlichen Schlüssels des Empfängers darf nicht etwa durch systematisch gewählte Klartextnachrichten die gesuchte Klartextnachricht ermittelt werden können (Schutz gegen so genannte *Known-Plaintext*-Angriffe).
Seit der Veröffentlichung der Grundideen wurden zahlreiche Public-Key-Verfahren entwickelt, siehe z.B. [MH78] [RSA78] und [ElG85]. Das im kommerziellen Umfeld mit Abstand erfolgreichste Verfahren ist RSA, welches eine so genannte Einweg-Falltürfunktion darstellt [RSA78].

Neben der Verschlüsselung sind digitale Signaturen (engl. *Digital Signature*) ein weiteres Einsatzfeld für Public-Key-Verfahren. Digitale Signaturen werden oft fälschlicherweise als ein elektronisches Pendant herkömmlicher Unterschriften betrachtet. Eine handschriftliche Unterschrift wird verwendet, um die Identität des Signierenden eines (papierbasierten) Dokuments, wie z.B. eines Vertrags im Nachhinein feststellen zu können. Diese Sicherung ist vielen Angriffen ausgesetzt: Eine Unterschrift kann auf beliebige Dokumente übertragen, d.h. kopiert werden. Nachträgliche Änderungen an einem signierten Dokument bleiben unbemerkt, es können beispielsweise komplette Seiten ins Dokument eingefügt werden. Ebenfalls kann eine Unterschrift als solche gefälscht werden (viele Kinder können die Unterschrift ihrer Eltern reproduzieren).
Von digitalen Signaturen (im Weiteren kurz Signaturen) wird hingegen im Allgemeinen erwartet: Die Identität des Signierenden muss im Nachhinein eindeutig feststellbar sein (Authentizität). Dies impliziert auch, dass der Signierende die Erstellung der Signatur nicht leugnen kann (Unabstreitbarkeit). Des Weiteren darf niemand in der Lage sein, ein signiertes Datum im Nachhinein unbemerkt zu verändern (Integrität) oder zu einer bestimmten Signatur ein passendes Datum zu erstellen (Nichtübertragbarkeit).

Eine weitere technische Forderung ist, dass eine Signatur im Vergleich zum signierten Datum kurz sein muss (Effizienz). Signaturen, die diesen Forderungen gerecht werden, können auch für die Zwecke der Authentifizierung innerhalb des Zugriffskontrollprozesses verwendet werden (siehe Kap. 2.1.1). Dabei kann der Zugriffswunsch durch den jeweilig Zugreifenden signiert werden.

Unterschiedliche Methoden sind zur Erzeugung von Signaturen bekannt, siehe z.B. [KPS95], [Sch96] und [Tan96]. Die zur Zeit verbreitetste Methode basiert auf der Kombination einer kollisionsresistenten Hashfunktion und eines Public-Key-Verfahrens. Abb. 2-3 zeigt schematisch die Erzeugung und Verifizierung einer Signatur nach dieser Methode.

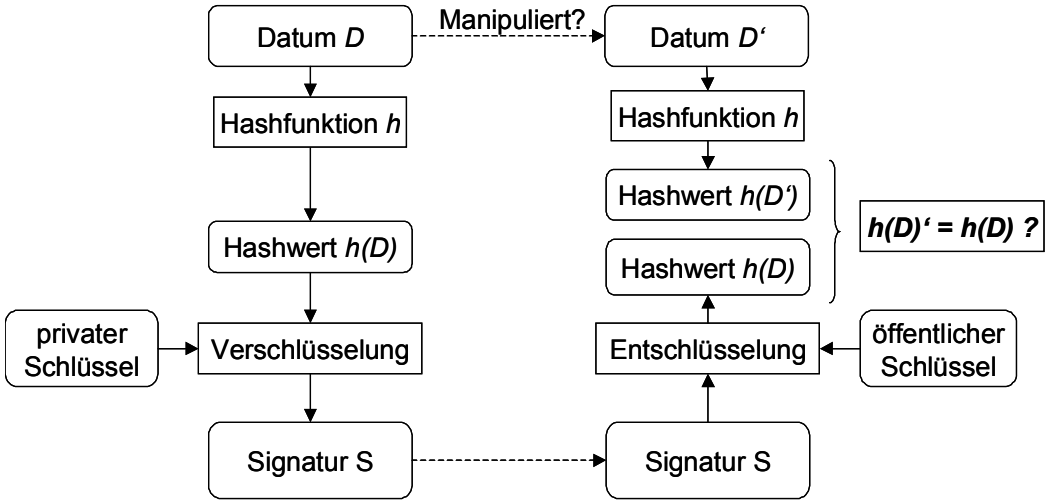


Abb. 2-3 Digitale Signaturen mit Hashfunktion und Public-Key-Verfahren

Eine Signatur *S* von *Bob* an einem Datum *D* wird durch $S=D_{Bob}(h(D))$ berechnet. Wie man auch in Abb. 2-3 links sieht, wird dabei mit Hilfe des privaten Schlüssels von *Bob* der Hashwert des Datums *D* verschlüsselt. Die entstehende Signatur *S* kann nur zusammen mit dem signierten Datum überprüft werden. Die hierfür benötigte Übermittlung wird in Abb. 2-3 durch die gestrichelten Linien angedeutet.
Um die Signatur *S* zu validieren, muss zunächst $E_{Bob}(S)=h(D)$ berechnet werden. Anschließend muss der Hashwert *h(D')* des dem Empfänger vorliegenden Datums *D'* berechnet werden. Von diesem Datum weiß man zunächst nicht, ob es mit dem ursprünglich signierten *D* identisch ist. Das ist genau der Fall, wenn *h(D')=h(D)*. Genau in diesem Fall ist die Signatur *S* gültig, andernfalls ist sie ungültig.
Die Sicherheit einer auf diese Weise erzeugten Signatur hängt von der Sicherheit des jeweiligen Public-Key-Verfahrens aber auch von der gewählten Hashfunktion ab. Gelingt es einem Angreifer zu einem Datum *D* ein anderes Datum $D' \neq D$ zu finden, so dass *h(D)=h(D')* (Verletzung der schwachen Kollisionsresistenz), so bestätigt die Signatur *S* fälschlicherweise auch die Gültigkeit von *D'*.

2.2.3 Digitale Signaturen mittels Authentifizierungsbäumen

In [Mer90] wurde erstmals vorgeschlagen, gleichzeitig mehrere Datensätze mit Hilfe eines Authentifizierungsbaums (engl. *Authentication Tree, Hash Tree, Merkle Tree*) durch eine einzige an der Wurzel dieses Baums getätigte Signatur zu signieren. Dies kann in unterschiedlichsten Szenarien genutzt werden, um größere Datenmengen, die aus vielen kleineren Datensätzen bestehen, effizient zu signieren.

Innerhalb der Sicherheitstechnik wurden den Bedürfnissen entsprechend angepasste Baumkonstruktionen erstmals in Zeitstempeldiensten angewandt [BHS92] [BLS00]. Die Idee ist aber auch für die Zwecke des Zertifikatsmanagements nützlich [Mic97] [ALO98] [Koc98] [GGM00] [BLL00] [BLL02].

In [Mer90] wurde hierfür ursprünglich eine binäre Baumstruktur vorgesehen (siehe Abb. 2-4). In den Knoten (*knoten*) wird jeweils ein Hashwert der Hashwerte der beiden Kinder dieses Knotens abgelegt. In jedem Blatt (*blatt*) wird der Hashwert eines bestimmten Datums ($datum_i$) aus der Gesamtmenge der mittels dieses Baums signierten Daten gespeichert. Zu beachten ist, dass die signierten Daten selbst nicht Bestandteil des Authentifizierungsbaums sind (siehe die Trennlinie in Abb. 2-4). Ein solcher Authentifizierungsbaum lässt sich durch die folgende Definition angeben:

Def. 2-5: Binärer Authentifizierungsbaum [Mer90]

$knoten.wert = h((knoten.kind_{links}).wert \,||\, (knoten.kind_{rechts}).wert)$
$blatt.wert = h(datum_i)$

wobei „||“ die Operation der Verkettung symbolisiert. Die in [Mer90] betrachtete binäre Struktur ist ein Spezialfall. Es sind durchaus Authentifizierungsbäume denkbar, deren Knoten maximal k ($k \geq 2$) statt maximal zwei Kinder (*kind*) haben.

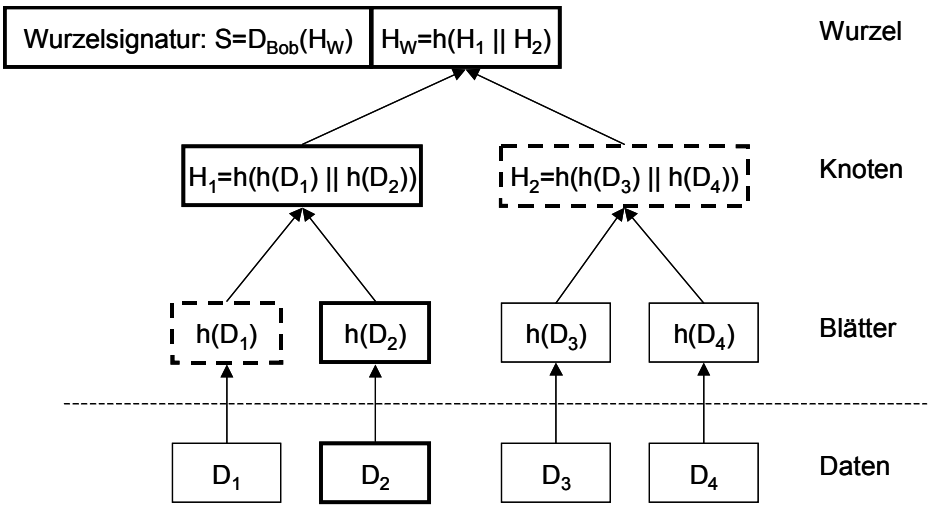


Abb. 2-4 Binärer Authentifizierungsbaum

Ist *knoten* die Wurzel des Baums, dann wird *knoten.wert* als der Wurzelhashwert (engl. *Root Hash*) des Baums genannt. Dieser wird im Beispielbaum in Abb. 2-4 mit H_W bezeichnet. Signiert wird der Wurzelhashwert des Baums H_W, indem *Bob* unter Einbeziehung seines privaten Schlüssels $S=D_{Bob}(H_W)$ berechnet (siehe Abb. 2-4).
Die entstehende Signatur S wird sinngemäß Wurzelsignatur (engl. *Root Signature*) genannt. Die Wurzelsignatur kann als ein gemeinsamer Signaturwert aller durch den Baum signierten Daten angesehen werden.
Um nun prüfen zu können, ob ein Datum D durch die Wurzelsignatur erfasst wurde, wird der so genannte Authentifizierungspfad (engl. *Authentication Path,* AP) $AP(D)$ zu D benötigt. Dieser $AP(D)$ enthält die Wurzelsignatur S und den so genannten Hashpfad (engl. *Hash Path,* HP) $HP(D)$ zu D. Wie der Name schon sagt, besteht ein Hashpfad aus verschiedenen im Baum enthaltenen Hashwerten.

Diese Hashwerte sind in den Geschwisterknoten der Knoten des Pfads von der Wurzel bis zum Blatt, welches *h(D)* enthält, zu finden.

Beispielsweise um die Signatur von D_2 aufgrund des Authentifizierungsbaums in Abb. 2-4 zu prüfen, wird der Authentifizierungspfad

$$AP(D_2)=\{ S=D_{Bob}(H_W) \,||\, HP(D_2)=[h(D_1) \,||\, H_2=h(h(D_3)||\, h(D_4)]\}$$

benötigt.
Wie man in Abb. 2-4 sieht, führt der Pfad zu dem Blatt, welches den Hashwert $h(D_2)$ von D_2 beinhaltet, über die Wurzel des Baums und über den Knoten, welcher den Hashwert H_1 beinhaltet (siehe die fettgedruckten Knoten in siehe Abb. 2-4).
$AP(D_2)$ enthält neben der Wurzelsignatur $D_{Bob}(H_W)$ den Hashpfad $HP(D_2)$. Der Hashpfad $HP(D_2)$ enthält die Hashwerte aus den Geschwisterknoten der Knoten des Pfads zum Blatt mit $h(D_2)$. Diese Hashwerte sind auf der „Blattebene" $h(D_1)$ und eine Ebene höher H_2 (siehe die fett und gestrichelt gedruckten Knoten).
Dieser Authentifizierungspfad muss den Hashwert $h(D_2)$ von D_2 nicht beinhalten, da dieser während der Verifizierung ausgehend vom vorliegenden Datum D_2 berechnet werden kann. Gleiches gilt für den Wurzelhashwert H_W, welcher während der Prüfung der Wurzelsignatur ermittelt wird. Während der Verifizierung von $AP(D_2)$ wird zunächst

$$H_W' = h(h(h(D_1) \,||\, h(D_2)) \,||\, H_2)$$

berechnet. Dabei wird anhand der im Hashpfad $HP(D_2)$ enthaltenen Hashwerte der Wurzelhashwert des Baums rekonstruiert. Anschließend muss die in $AP(D_2)$ enthaltene Wurzelsignatur $D_X(H_W)$ mit

$$H_W = E_{Bob}(D_{Bob}(H_W))$$

verifiziert werden. Hierfür muss natürlich der öffentliche Schlüssel von *Bob* vorliegen.
Falls

$$H_W = H_W'$$

gilt, so ist der Authentifizierungspfad $AP(D_2)$ und damit die Signatur an D_2 gültig, andernfalls nicht.

Beim Einsatz von Authentifizierungsbäumen in verteilten Systemen (beispielsweise für die Zwecke des Zertifikatsmanagements) müssen folgende Aspekte beachtet werden:

- **Kollisionsresistenz:** Die Hashfunktion *h* muss im Sinne von Def. 2-4 kollisionsresistent sein. Ansonsten lassen sich sowohl der Baum wie auch einzelne Authentifizierungspfade im Nachhinein fälschen. Zusätzlich gilt, dass in die Hashwertberechnung eines Knotens insgesamt 2 (k) Hashwerte der Länge l einfließen. Damit gibt es in diesem Fall insgesamt 2^{2l} (2^{kl}) mögliche Urbilder für h. Betrachtet man Def. 2-3, so ist dies eine Einschränkung der Möglichkeiten, um das Finden einer „passenden" Kollision von h auszuschließen.

- **Dynamische Erzeugung der Authentifizierungspfade:** Bei Änderungen am Baum, wie z.B. beim Einfügen oder Entfernen eines Datensatzes, ändert sich der Wurzelhashwert und folglich die Wurzelsignatur. Damit verändern sich sämtliche Authentifizierungspfade zu allen durch den Baum signierten Daten.
  Dies erschwert eine Zwischenspeicherung im Voraus berechneter gültiger Authentifizierungspfade und macht in den meisten Fällen eine dynamische Erzeugung der Authentifizierungspfade anhand eines abgespeicherten Baums erforderlich. Um hierbei die Suche nach den jeweils erforderlichen Hashwerten des Baums effizient zu gestalten, bedarf es einer entsprechenden Indizierung der Baumknoten.

- **Nutzung in Offline-Online-Systemen:** Authentifizierungspfade können von einer beliebigen Stelle erstellt werden, die lesenden Zugriff auf die Knoten eines Authentifizierungsbaums hat. Dies kann in verteilten so genannten Offline-Online-Systemen ausgenutzt werden. Hierbei wird ein Authentifizierungsbaum durch eine überwiegend Offline-Instanz erzeugt und signiert und (z.B. periodisch) an eine Online-Instanz übermittelt. Diese Online-Instanz (typischerweise eine Datenbank) speichert den Baum und kann auf Anfragen Dritter die jeweilig benötigten aktuell gültigen Authentifizierungspfade dynamisch erzeugen und aushändigen.

- **Kosten einer Signaturprüfung:** Wurde ein Datensatz D mit Hilfe eines Authentifizierungsbaums signiert, fällt der rechnerische Aufwand der Signaturprüfung höher aus als wäre D gemäß der in Kap. 2.2.2 angegebenen Formel $D_{Bob}(h(D))$ signiert. Signiert ein binärer (k-fach verzweigter) Authentifizierungsbaum insgesamt n Datensätze, so ist die Prüfung des Pfads zu D mit etwa $O(\log_2 n)$ ($O(\log_k n)$) Hashwertberechnungen verbunden. Wie in Kap. 2.2.2 gezeigt wurde, bedarf eine „einfache“ Signaturprüfung genau einer Hashwertberechnung. Da der Authentifizierungspfad zu D etwa $O(\log_2 n)$ $(O(\log_k n))$ Hashwerte beinhaltet, erhöhen sich entsprechend die Übertragungskosten.

- **Datenbedarf zur Prüfung der Wurzel:** In einem praktischen Einsatzfall sollten neben dem betrachteten Wurzelhashwert H_W auch weitere Informationen *Infos* in die Wurzelsignatur des Baums einfließen.
  Beispiele sind Datum und Uhrzeit der Erzeugung der Wurzelsignatur oder der Name des jeweilig verwendeten Signierverfahrens. Eine solche Wurzelsignatur S kann bspw. die Form $S=D_X(h(H_W \,||\, Infos))$ haben. Um S prüfen zu können, müssen die Authentifizierungspfade *Infos* enthalten.

2.2.4 X.509-Attributszertifikate

Eine Signatur kann die Authentizität der signierten Daten nur dann zufriedenstellend bestätigen, wenn der zur Signaturprüfung (siehe Abb. 2-3) verwendete öffentliche Schlüssel authentisch ist. Gelingt es nämlich *Mallory* sein eigenes Schlüsselpaar beispielsweise unter dem Namen „*Bob*“ unentdeckt zu benutzen, so kann er in *Bobs* Namen Daten signieren. Es sind zahlreiche Ansätze zur authentischen Verteilung öffentlicher Schlüssel bekannt, siehe z.B. [Sch96] [MOV96] [Coc01].
Heute wird üblicherweise eine den Teilnehmern vertraute Zertifizierungsstelle (engl. *Certification Authority, CA*) [Koh78] in die Schlüsselverteilung einbezogen. Eine CA bescheinigt, dass ein bestimmter öffentlicher Schlüssel einem bestimmten Subjekt (Mensch, Maschine, usw.) gehört. Hierfür wird ein so genanntes Schlüsselzertifikat (engl. *Public-Key Certificate*) ausgestellt, welches ein geordnetes Tripel *{S, I, N}* ist.

Hier bezeichnet *I* sämtliche mit dem jeweiligen öffentlichen Schlüssel zusammenhängende Informationen. *N* (Name) ist die jeweilig eindeutige Repräsentation des jeweiligen Schlüsselinhabers. $S=D_{CA}(h(I,N))$ ist eine Signatur über *I* und *N*, welche unter der Verwendung des privaten Schlüssels der CA (häufig als „Root-Schlüssel" bezeichnet) berechnet werden muss.
Alice, die *Bob* im Voraus nicht kennen muss, kann nun die Authentizität von *Bobs* öffentlichem Schlüssel relativ einfach sicherstellen. Hierfür muss sie ein auf *Bobs* Namen ausgestelltes Schlüsselzertifikat von einer CA erhalten. Um dieses Zertifikat zu überprüfen, muss der öffentliche Schlüssel dieser CA im Voraus sicher ermittelt werden. Dies impliziert, dass im Prinzip ein einziger sicherer Schlüsselaustausch mit dieser CA genügt, um mit beliebig vielen (unbekannten) Teilnehmern öffentliche Schlüssel austauschen zu können.
Die Verwendung von Zertifikaten beschränkt sich mittlerweile nicht mehr auf die Sicherung von Schlüsseln. Außerdem entstanden in den letzten Jahren unterschiedliche, teilweise standardisierte Zertifikatsformate, wie z.B. X.509 [ITU01] und SPKI [Ell99]. Deshalb lässt Def. 2-6. Strukturierung sowie Verwendungszweck des Zertifikats offen:

Def. 2-6: Zertifikat

Ein Zertifikat ist eine durch eine vertrauenswürdige Instanz digital signierte Beglaubigung von sicherheitskritischen Informationen.

Im Allgemeinen sollten Zertifikate folgendem Kriterium entsprechen [MOV96]: Es muss jederzeit eindeutig festzustellen sein, ob ein Zertifikat von einer vertrauenswürdigen Stelle ausgestellt, d.h. signiert wurde. Hierfür muss der öffentliche Schlüssel dieser Stelle in einer authentischen Form vorliegen. Eine prinzipielle Frage ist, was in diesem Kontext unter „Vertrauen" verstanden wird. Jemandem hinsichtlich Zertifizierung zu vertrauen bedeutet zu glauben, dass er stets korrekte Informationen zertifiziert und primär nicht beabsichtigt, andere zu betrügen. Dies schließt natürlich nicht aus, dass z.B. versehentlich falsche Daten zertifiziert werden. Um dem entgegengebrachten Vertrauen gerecht zu werden, muss eine als vertrauenswürdig eingestufte Stelle dafür sorgen, dass nur sie als gültig akzeptierte Zertifikate ausstellen kann. Dies impliziert unter anderem den adäquaten Schutz des privaten Schlüssels der Zertifizierungsstelle.

Durch Zertifikate können auch Autorisierungsdaten gegen nachträgliche Manipulationen geschützt werden. Ein Anwendungsgebiet für solche Zertifikate ist die sichere Zugriffskontrolle [LN99] [HMM+00] [MGP+01] [GS02]. In der Literatur werden solche Zertifikate häufig als „Attributszertifikat" (engl. *Attribute Certificate*), „Autorisierungszertifikat" (engl. *Authorization Certificate*) oder „Privilegienzertifikat" (*Privilege Certificate*) bezeichnet. In Anlehnung an den ITU-Standard [ITU01] mit dem Kennzeichen X.509 wird in dieser Arbeit der Begriff „Attributszertifikat" verwendet.

Def. 2-7: Attributszertifikat

Ein Attributszertifikat ist ein Zertifikat, welches während der Zugriffskontrolle ausgewertete Autorisierungsdaten beinhaltet.

Diese Definition grenzt den Verwendungsbereich von Attributszertifikaten auf die Zugriffskontrolle ein, lässt aber die Formatierung sowie Strukturierung der zertifizierten Autorisierungsdaten offen. Die Strukturierung der zertifizierten Autorisierungsdaten wird durch die gewählten Autorisierungsmodelle (siehe Kap. 2.1.3) bedingt.

Die Formatierung solcher Zertifikate wird in verschiedenen Standards, unter anderem im nachfolgend betrachteten X.509-Standard festgelegt.
Eine weitere erwähnenswerte Norm für die zertifikatsbasierte Zugriffskontrolle ist *Simple Public Key Infrastructure/Simple Distributed Security Infrastructure* oder kurz SPKI/SDSI. Dieser wurde im universitären Umfeld entwickelt [Ell99], [EFL+99], [Myr00], [HM01], [DE02] und fand in kommerziellen Anwendungen bis jetzt vergleichsweise wenig Akzeptanz.

Der X.509-Standard behandelt sowohl Schlüssel- als auch Attributszertifikate. Beide Arten von Zertifikaten werden von hierfür spezialisierten Systemen verwaltet, die man Public-Key Infrastructure (PKI) respektive Privilege Management Infrastructure (PMI) nennt.
Das Hauptziel dieser Standardisierung ist die Interoperabilität zwischen autonom betriebenen PKIs und PMIs zu fördern. Der X.509-Standard [ITU01] unterscheidet zwischen zwei Arten von Attributszertifikaten:

1. **Um Autorisierungsdaten ergänzte Schlüsselzertifikate**: Für die Zugriffskontrolle können erweiterte Schlüsselzertifikate (engl. *Extended Public Key Certificates*) eingesetzt werden [ITU01]. Die hierin enthaltenen Schlüssel können innerhalb der Zugriffskontrolle zur sicheren Anmeldung, d.h. Authentifizierung der Zugreifenden verwendet werden (siehe auch Kap. 2.2.2). In den meisten Systemen kommen heute bekanntlich Passwort-basierte Lösungen zum Einsatz. In solchen Systemen spielen Schlüsselzertifikate aus Sicht der Authentifizierung der Zugreifenden keine Rolle. Folglich werden auch keine erweiterten Schlüsselzertifikate benötigt, um deren Autorisierungsdaten zu sichern.
   Die Nutzung erweiterter Schlüsselzertifikate impliziert außerdem, dass die Zertifizierungsstellen sowohl für die Schlüsselverwaltung als auch für die Autorisierung der Benutzer zuständig sind [ITU01]. Solche zentralisierten Verwaltungsinstanzen könnten leicht zum (administrativen) Flaschenhals in einem System werden:
   Da Autorisierungsdaten meistens einer wesentlich höheren Änderungsdynamik als öffentliche Schlüssel unterliegen, müssten die erweiterten Schlüsselzertifikate häufig erneuert und verteilt werden. Das heißt, wegen ständiger Änderungen der Autorisierungsdaten müssten auch stets die Schlüsselpaare der Benutzer erneuert beziehungsweise neu zertifiziert werden.

2. **Attributszertifikate mit Autorisierungsdaten**: Aus den oben genannten Gründen werden in weiteren Teilen der Arbeit Attributszertifikate betrachtet, die primär keine Informationen zur Benutzerauthentifizierung enthalten. Solche Attributszertifikate werden von hierfür spezialisierten als vertrauenswürdig eingestuften so genannten *Source of Authorities* (SOA) ausgestellt und verwaltet [ITU01].
   Eine SOA verkörpert laut des Standards [ITU01] den Eigentümer bestimmter schützenswerter Objekte in einem System, zu welchen die von dieser SOA signierten Attributszertifikate Zugriff gewähren können. Dies impliziert, dass in einem gegebenen System durchaus gleichzeitig mehrere SOAs existieren können, die jeweils eine Teilmenge der Objekte unter ihrer Kontrolle halten.

Ein X.509-Attributszertifikat hat die Bezeichnung *AttributeCertificateInfo* und enthält eine Sequenz aus Pflicht- und optionalen Feldern, wie das in Abb. 2-5 gezeigt wird.

Als abstrakte Spezifikationssprache für Attributszertifikate wurde die „*Abstract Syntax Notation One*“ (ASN.1) [ITU02] gewählt. Zur Kodierung und Dekodierung der ASN.1-Strukturen wird die so genannte „*Basic Encoding Rule*“ (BER) empfohlen.

```
AttributeCertificateInfo ::= SEQUENCE {
   version AttCertVersion,
   holder Holder,
   issuer AttCertIssuer,
   signature AlgorithmIdentifier,
   serialNumber CertificateSerialNumber,
   attrCertValidityPeriod AttCertValidityPeriod,
   attributes SEQUENCE OF Attribute,
   issuerUniqueID UniqueIdentifier OPTIONAL,
   extensions Extensions OPTIONAL  }
```

Abb. 2-5: Felder eines X.509-Attributszertifikats

Die Felder eines X.509-Attributszertifikats haben die folgende Bedeutung:

- **Version:** Dieses Feld muss seit 1999 den Wert *2* haben, um das neue Format von der ersten nicht mehr kompatiblen Version zu unterscheiden.

- **Holder:** Zertifizierte Zugreifende werden durch dieses Feld systemweit eindeutig identifiziert. Es ist eine Sequenz optionaler Einträge, von denen mindestens einer vorhanden sein muss:

  ```
  Holder ::= SEQUENCE {
                baseCertificateID [0]IssuerSerial OPTIONAL,
                entityName [1]GeneralNames OPTIONAL,
                objectDigestInfo [2]ObjectDigestInfo OPTIONAL }
  ```

  BaseCertificateID soll benutzt werden, wenn der Zugreifende auch ein Schlüsselzertifikat besitzt, welches z.B. zu seiner Authentifizierung verwendet wird. In diesemn Fall werden hier der Name des Ausstellers (engl. *Issuer*) sowie die Seriennummer dieses Schlüsselzertifikats angegeben (*IssuerSerial*).
  EntityName kann hingegen in Systemen benutzt werden, welche die Benutzer nicht mittels Schlüsselzertifikaten authentifizieren. Dieser Eintrag ist also ein eindeutiger nicht zertifikatsgebundener Identifikator von Holder.
  ObjectDigestInfo kann direkt zur Authentifizierung des Zugreifenden verwendet werden, indem hier z.B. ein Hashwert (Digest) seines Passworts abgelegt wird.

- **Issuer:** Auf eine ähnliche Weise wird der Aussteller des Attributszertifikats angegeben. Hierbei beinhaltet *issuerName* den systemweit eindeutigen Namen des Ausstellers. Da das Attributszertifikat mit Hilfe seines öffentlichen Schlüssels geprüft wird, kann unter *baseCertificateID* das passende Schlüsselzertifikat angegeben werden.

AttCertIssuer ::= SEQUENCE {
issuerName [1]GeneralNames OPTIONAL,
baseCertificateID [0]IssuerSerial OPTIONAL,
objectDigestInfo [2]ObjectDigestInfo OPTIONAL }

- **Signature:** Dieses Feld gibt die zum Signieren des Zertifikats verwendeten Algorithmen an, z.B. in der Form „*SHA-1withRSA*“, um die Überprüfung des Attributszertifikats zu ermöglichen. Die genaue Kodierung dieser Information wurde nicht festgelegt, um verschiedene Systeme und Schemata zum Signieren von Daten verwenden zu können.

- **SerialNumber:** Dieses Feld enthält eine ganze Zahl. Diese muss durch den Aussteller eindeutig vergeben werden, so dass Attributszertifikate mit identischem *Issuer*-Feld voneinander über diese Zahl eindeutig unterschieden werden können. Dieses Feld wurde zur Unterstützung der Suche nach Attributszertifikaten und auch zu ihrer Referenzierung vorgesehen. Es kann innerhalb einer PMI vorkommen, dass unterschiedliche Aussteller dieselbe *SerialNumber* mehrfach vergeben. Das impliziert, dass Attributszertifikate innerhalb einer PMI über das Felderpaar (*Issuer, SerialNumber*), welches man *IssuerSerial* nennt, eindeutig zu unterscheiden sind.

- **AttrCertValidityPeriod:** Die vorgesehene Gültigkeitsperiode des Attributszertifikats wird durch zwei Felder definiert:

  AttCertValidityPeriod ::= SEQUENCE {
  notBeforeTime GeneralizedTime,
  notAfterTime GeneralizedTime }

  Es kann natürlich nicht garantiert werden, dass das Zertifikat nicht frühzeitig für ungültig erklärt werden muss, beispielsweise wegen Änderungen der Zugriffsrechte des Zertifikatsbesitzers.

- **Attributes:** Die zertifizierten Autorisierungsdaten werden unter diesem Feld angegeben. Die Autorisierungsdaten können beliebiger Semantik sein, der Standard behandelt dies wegen der Anwendungsabhängigkeit nicht weiter. Prädestiniert sind Attributszertifikate jedoch für die Angabe einer Fähigkeitsliste (siehe Kap. 2.1.3) des jeweiligen *Holders*.

- **IssuerUniqueID:** Dieses Feld kann optional benutzt werden, um den Aussteller des Attributszertifikats zu identifizieren, wenn das Feld *Issuer* hierfür aus irgend einem Grund, wie es im Standard heißt, nicht ausreichen sollte.

- **Extensions:** Um die Flexibilität des Formats zu erhöhen, sind optionale Erweiterungen der Attributszertifikate möglich. Auf einige wird noch weiter unten eingegangen.

Ein Beispielzertifikat des vorgestellten Formats zeigt Abb. 2-6. Da der X-509 Standard als primäre Ablage für Attributszertifikate ein zum ITU-Standard X.500 konformes Directory vorsieht, werden die verschiedenen Bezeichner (Namen) gemäß der dort behandelten Konventionen (engl. *Distinguished Name*) vergeben [ITU93].

```
ACInfo:
        Version: 2
        Holder:
                BaseCertificateID: null
                EntityName: [CN=Bob,OU=CM,OU=Uni-Ka,C=de]
                ObjectDigestInfo: null
        Issuer:
                IssuerName: [CN=SOA,OU=CM,OU=Uni-Ka,C=de]
                BaseCertificateID: null
                ObjectDigestInfo: null
        Signature: SHA1withRSA, OID = 1.2.840.113549.1.1.5
        SerialNumber: 0x10
        AttrCertValidityPeriod:
                NotBeforeTime: Wed Dec 24 09:27:16 CET 2003
                NotAfterTime: Wed Dec 24 19:27:16 CET 2003
        Attributes: 1
        [1]: Type: ObjectId: 1.3.6.1.4.1.1234.8.1.1
        Values: 1
        Values[0]: AccessPrivilege://www.cm-tm.uka.de/
        IssuerUniqueID: null
        Extensions: null
```

Abb. 2-6: Beispiel für ein X.509-Attributszertifikat

Die Formatierung von Attributszertifikaten kann (muss) den Bedürfnissen der jeweiligen Anwendung angepasst werden (engl. *Profiling*). Im Wesentlichen betrifft das die Auswahl der jeweils benötigten optionalen Felder sowie die Interpretation von Einträgen wie z.B. *Attributes, EntityName, ObjectDigestInfo* oder *Signature*.
Außerdem ist es erlaubt, *AttributeCertificateInfo* in eine übergeordnete Datenstruktur einzufügen und diese zu signieren (engl. *Enveloping*). An einem eigenen Profil für Attributszertifikate arbeitet aktuell die Gruppe namens „*Public-Key Infrastructure X.509*“ (PKIX) der „*Internet Engineering Task Force“* (IETF) [FH02], [FP03], [CS03].

2.3 Realisierung der Zugriffskontrolle mit Attributszertifikaten

Eine wichtige Voraussetzung für eine erfolgreiche Zugriffskontrolle ist, dass die Attributszertifikate stets aktuelle (gültige) Autorisierungsdaten enthalten. Ansonsten werden innerhalb der Zugriffskontrolle zwar gesicherte aber inhaltlich falsche Daten ausgewertet. Aus diesem Grund muss die Zertifizierung mit der Änderungsdynamik der ihr in diesem Sinne übergeordneten Autorisierungsdaten Schritt halten. Die Nutzung von Attributszertifikaten macht außerdem eine Erweiterung des in Kap. 2.1.1 vorgestellten Zugriffskontrollprozesses erforderlich. Diese sowie Aspekte der zertifikatsbasierten Delegierung von Berechtigungen werden nachfolgend diskutiert.

2.3.1 Durchgängiger Schutz von Autorisierungsdaten

Prinzipiell lassen sich Autorisierungsdaten beliebiger Natur und Semantik auf Attributszertifikate abbilden und schützen, diesbezüglich scheint es keine Einschränkungen zu geben.

Um dies zu untermauern, wird im Folgenden gezeigt, wie die in Kap. 2.1.3 vorgestellten Fähigkeitslisten beziehungsweise Rollen mit Attributszertifikaten umgesetzt werden können [NA+01a].

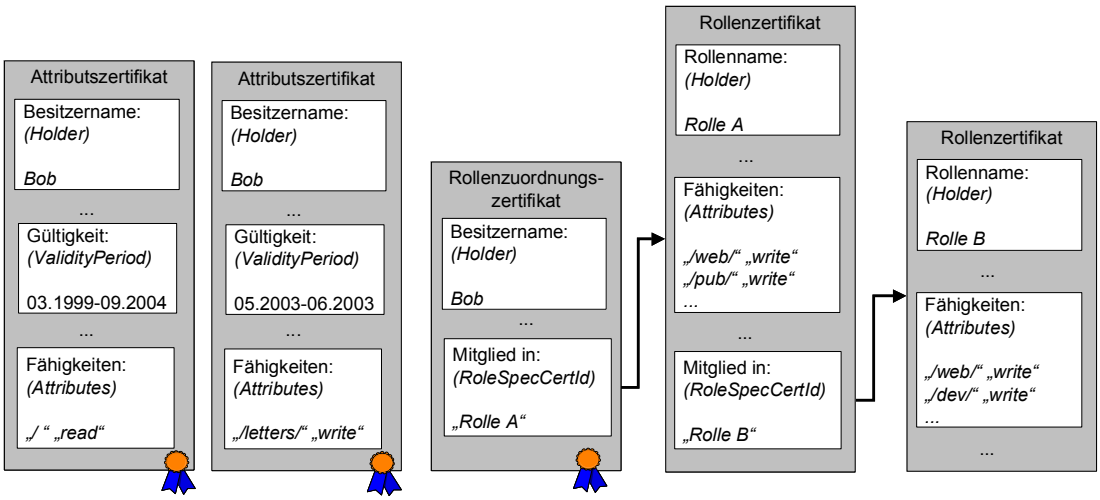


Abb. 2-7: Rollenbasierte Autorisierung mit Attributszertifikaten

Abb. 2-7 zeigt eine denkbare Konfiguration, in welcher der Benutzer *Bob* über verschiedene Fähigkeiten sowie Rollenmitgliedschaften innerhalb eines Dateisystems verfügt. Wie man in Abb. 2-7 sieht, besitzt *Bob* drei auf seinen Namen ausgestellte Zertifikate. Die beiden Zertifikate in Abb. 2-7 links bescheinigen nicht rollengebundene Berechtigungen von *Bob*, welche in Form von Fähigkeitslisten repräsentiert wurden. *Bob* darf demnach sämtliche Verzeichnisse unterhalb der Wurzel lesen (*„/“ „read“*) sowie das Verzeichnis */letters/* schreiben *(„/letters/“, „write“*).
Wie aus Abb. 2-7 ersichtlich wird, wurden diese Berechtigungen auf jeweils unterschiedliche Zeitintervalle eingeschränkt. Das dritte Zertifikat von *Bob* ist ein so genanntes Rollenzuordnungszertifikat (engl. *Role Assignment Certificate*). Mit Hilfe dieses Zertifikats wurde *Bob* der Rolle mit der Bezeichnung *Rolle A* zugeordnet.
Diese Rolle sowie die Rolle namens *Rolle B* wurden wiederum in zwei so genannten Rollenzertifikaten (engl. *Role Specification Certificate*) spezifiziert. Die Rollen *A* und *B* bilden eine einfache zweistufige Hierarchie, in welcher *Rolle B* eine der *Rolle A* übergeordnete Rolle darstellt (siehe auch Kap. 2.1.3).
Durch diese Rollenhierarchie werden *Bob* sämtliche in den beiden Rollen aufgelisteten Fähigkeiten zugeordnet, d.h. er kann dadurch weitere Verzeichnisse (*/web/, /pub/, /dev/*) schreiben. Für die hierbei benötigte Referenzierung der Rollenzertifikate wurde im X.509-Standard die Erweiterung *RoleSpecCertIdentifier* eingeführt [ITU01].

```
RoleSpecCertIdentifier ::= SEQUENCE {
        roleName [0] GeneralName,
        roleCertIssuer [1] GeneralName,
        roleCertSerialNumber [2] CertificateSerialNumber OPTIONAL,
        roleCertLocator [3] GeneralNames OPTIONAL  }
```

Dieses Erweiterungsfeld beinhaltet eine Sequenz (Liste) von Angaben zu Rollenzertifikaten. Um diese Rollenzertifikate während der Zugriffskontrolle zu finden, müssen diese durch systemweit eindeutige Bezeichner repräsentiert werden.

Dementsprechend wird neben dem Namen (*roleName*) der jeweiligen Rolle auch der Name des Ausstellers (*roleCertIssuer*) sowie (optional) die Seriennummer (*roleCertSerialNumber*) des jeweiligen Rollenzertifikats angegeben.
Die Verwaltung von Rollen beziehungsweise Rollenzertifikaten kann in einem System durch eine so genannte Rollenautorität (engl. *Role Authority*) erfolgen [ITU01]. Dieser muss bezüglich der Korrektheit der jeweilig festgelegten Rollen vertraut werden.
Alternativ können die bereits erwähnten *Source of Authorities* (SOAs) als „lokale" Rollenautoritäten auftreten. Dabei können sie Rollen definieren, Rollenzertifikate ausstellen sowie den eigens erzeugten Rollen Benutzer zuordnen. In diesem Fall müssen während der Zugriffskontrolle nur von diesen ohnehin schon als vertrauenswürdig eingestuften Stellen stammende Attributszertifikate akzeptiert werden.

Gelingt es in einem System die Autorisierungsdaten mittels Attributszertifikaten zu sichern, so muss auch dafür gesorgt werden, dass diese stets aktuelle Autorisierungsdaten enthalten. Ansonsten können die Zugriffskontrolle veraltete Daten erreichen, was natürlich unerwünscht ist. Außerdem ist es oftmals nützlich entscheiden zu können, wer welche Berechtigungen zu einem gegebenen früheren Zeitpunkt besaß (Nachweisführung). Gemäß dieser Erwartungen lassen sich verschiedene relevante „Lebensphasen" für Attributszertifikate identifizieren, die in Abb. 2-8 gezeigt werden [CFS+03]. Eine PMI kann je nach Bedarf die mit diesen Phasen zusammenhängenden administrativen Vorgänge vollständig oder teilweise implementieren. Die einzelnen Phasen werden anhand der in Abb. 2-7 gezeigten direkt für *Bob* ausgestellten Zertifikate (mit *Holder="Bob"*) vorgestellt.

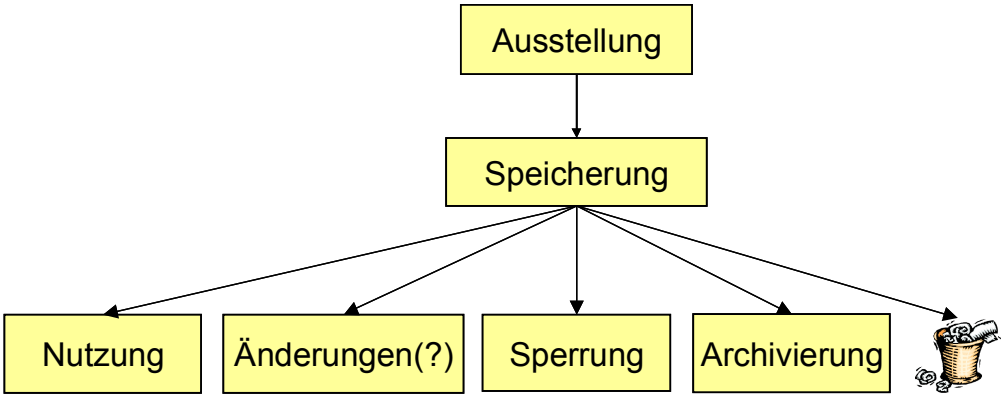


Abb. 2-8: Lebensphasen eines Attributszertifikats

- **Ausstellung:** Zuerst muss entschieden werden, welche Fähigkeiten und Rollenzugehörigkeiten *Bob* für welchen Zeitraum erhalten soll. Erst danach können die Attributszertifikate durch die zuständige SOA erzeugt (d.h. editiert) und anschließend signiert werden. Da die Sicherheit der Zugriffskontrollentscheidungen unmittelbar mit der hierbei getätigten Signatur zusammenhängt, muss die Ausstellung des Zertifikats auf einem von der Außenwelt bestmöglich abgeschotteten, d.h. offline System erfolgen.

- **Speicherung:** Das unter sicheren Bedingungen ausgestellte Attributszertifikat wird nun für die Zugriffskontrolle durch ein Online-System „freigegeben". Je nach gewählter Strategie bedeutet dies die Ablage der Zertifikate in einer hierfür vorgesehenen Zertifikatsdatenbank oder deren Aushändigung direkt an *Bob*.

Im ersten Fall kann während der Zugriffskontrolle diese Datenbank nach *Bobs* Zertifikaten gefragt werden (Server-Pull-Ansatz). Im zweiten Fall muss *Bob* selbst sein Zertifikat der Zugriffskontrolle zukommen lassen (Client-Push-Ansatz). Ein besonderer Aspekt bei der Speicherung von Attributszertifikaten ist deren Geheimhaltung. Im Gegensatz zu Schlüsselzertifikaten, welche veröffentlicht werden können, stellen Attributszertifikate häufig Geheimnisse dar.
Dies impliziert, dass der lesende Zugang zu diesen Zertifikaten durch den Aussteller auf hierfür Vorgesehene eingeschränkt werden sollte. Hierfür bietet sich eine entsprechende Verschlüsselung der Zertifikate an.

- **Nutzung:** Die Nutzung der ausgestellten Attributszertifikate bedeutet hauptsächlich deren Überprüfung während der Zugriffskontrolle. Dabei gilt im Wesentlichen zu entscheiden, ob die Zertifikate in einer unveränderten und authentischen Form vorliegen, und ob ihre Inhalte aktuell als gültig akzeptiert werden können. Auf den genaueren Ablauf dieses Prozesses wird noch in Kap. 2.3.3 detailliert eingegangen.

- **Änderungen:** Im Laufe der Zeit kann es erforderlich sein, die Inhalte eines Attributszertifikats vor dessen Ablaufdatum zu ändern. In diesem Fall kann das Zertifikat geändert, neu signiert und die neue Version abgespeichert werden. Diese Vorgehensweise ist aber aus sicherheitstechnischen Gründen bedenklich: Sollte *Bob* eine seiner Fähigkeiten auf diese Weise frühzeitig entzogen werden, so müsste sichergestellt werden, dass die Zugriffskontrolle stets die neuere Version des geänderten Zertifikats zu sehen bekommt. Angenommen *Bob* hält seine Zertifikate unter Kontrolle (Client-Push-Modell), so kann es nicht ausgeschlossen werden, dass er „bei Bedarf" die ältere (mächtigere) Version des geänderten Zertifikats vorzeigt.

- **Sperrung:** Eine bessere Strategie ist es, ein Attributszertifikat, welches vor seinem Ablaufsdatum geändert werden soll, explizit für ungültig zu erklären. Sollte dieses Zertifikat jedoch einen Nachfolger im System haben, dann sollte hierfür ein eindeutig vom alten unterscheidbares neues Zertifikat ausgestellt werden. Bei dem obigen Beispiel bleibend, muss *Bobs* ursprüngliches Zertifikat für die Zugriffskontrolle sicher überprüfbar für ungültig erklärt, beziehungsweise für *Bob* muss ein neues Zertifikat mit den geminderten Fähigkeiten ausgestellt werden.

- **Archivierung:** Eine Archivierung frühzeitig oder ordnungsgemäß ungültig gewordener Zertifikate kann aus unterschiedlichen Gründen erforderlich sein. Sollte beispielsweise *Bob* vor Gericht behaupten, zu einem bestimmten Zeitpunkt keine Zugangsberechtigungen zu einer bestimmten Ressource gehabt zu haben, so könnte der Systembetreiber das damals für *Bob* ausgestellte, inzwischen archivierte Zertifikat vorlegen, um das Gegenteil zu beweisen.

Am Ende der vorgesehenen Lebens- beziehungsweise Nutzungsperiode können bekannte existierende (archivierte) Exemplare des Zertifikats aktiv zerstört werden.

2.3.2 Delegierung von Rechten: Sicherheit durch Attributszertifikate

Die in Kap. 2.1.3 bereits erwähnte und hier vertieft behandelte Delegierung von Rechten ist eine attraktive Anwendungsmöglichkeit für Attributszertifikate in verteilten Informationssystemen.

Dabei sind Attributszertifikate nicht nur als gesicherte Transport- beziehungsweise Speicherungsmittel verschiedener Autorisierungsdaten, sondern auch als sicherer Ursprungsnachweis früher getätigter Delegierungen anzusehen. Die Delegierung wird für die Zwecke dieser Arbeit folgendermaßen definiert:

Def. 2-8: Delegierung

Unter Delegierung wird das Übertragen einer Teilmenge der Zugriffsberechtigungen eines Delegierungsgebers auf einen Delegierungsnehmer verstanden.

Dieser zunächst einfach klingende Vorgang - unterstützt durch Attributszertifikate - kann in unterschiedlichen sicherheitsrelevanten Szenarien eingesetzt werden, wie zum Beispiel:

- **Dezentralisierung der Autorisierung:** Mithilfe von geeigneten Delegierungsmechanismen kann die Autorisierung in einem Zugriffskontrollsystem schrittweise dezentralisiert werden. Beim Aufsetzen eines Systems ist es meistens noch unklar wer welche Berechtigungen künftig besitzen beziehungsweise vergeben darf. In den meisten Fällen wird heute während deren Initialisierung eine Art allmächtiger „Super-Benutzer“ (genannt „*Root*“, „*Domain Administrator*“, „*Security Officer*“, usw.) erzeugt. Dieser kann als anfangs einzig Berechtigter gemäß der jeweilig herrschenden lokalen und/oder globalen Sicherheitsrichtlinien die Zugriffsrelation des Systems umsetzen. Wie bereits in Kap. 2.1.3 erwähnt, kann dies mit wachsender Komplexität, Dynamik und Menge der Autorisierungsdaten nur bedingt funktionieren. Früher oder später werden weitere Subjekte (Administratoren) benötigt, um die jeweilige Zugriffsrelation zu pflegen. Sie können durch Delegierungen unterschiedliche Kompetenz- beziehungsweise Zuständigkeitsbereiche von „Super-Benutzer“ schrittweise übernehmen. Beispielsweise kann ein „*Rollenadministrator*“ die Pflege sämtlicher Rollen (Rollenzertifikate) im System erledigen. Eine derartige Aufteilung administrativer Zuständigkeiten kann sich mit der Zeit dynamisch ändern, was weitere Delegierungen, aber auch deren Rückzug erforderlich machen kann. Bei der Abwicklung solcher Delegierungen können Attributszertifikate bescheinigen, dass ein bestimmtes Subjekt bestimmte administrativen Rechte besitzt, aber auch dass diese Rechte vom „Super-Benutzer“ stammen.

- **Lösung für Vertreterprobleme:** Ebenso interessant erscheinen Probleme innerhalb der heute üblichen so genannten Mehrschichten-Architekturen [Inc02]. Solche Systeme können beispielsweise aus den funktional getrennten Schichten zur Präsentation, Geschäftslogik und Datenhaltung bestehen.
  End-Benutzer kommunizieren hierbei mit Komponenten der Präsentationsschicht, wie z.B. einem Webserver. Sie haben meist gar keine Möglichkeit direkt auf die (eigenen) Daten in der Datenhaltungsschicht zuzugreifen. Statt dessen werden sie dort von Komponenten der Geschäftslogikschicht vertreten, welche „in ihren Namen“ Daten aus der (in die) Datenhaltungsschicht abholt (ablegt). Der Austausch der Daten zwischen deren Konsumenten und deren Ablage erfolgt also über Vertretersysteme der Geschäftslogikschicht. Dabei müssen die Vermittlersysteme die Vereinigungsmenge der Berechtigungen sämtlicher End-Benutzer und damit praktisch uneingeschränkte Zugriffsmöglichkeiten auf die geschützten Daten erhalten. Durch die Verwendung von Attributszertifikaten kann hier eine Machteinschränkung der Zwischenschichten erreicht werden.

Die End-Benutzer können hierbei (etwa für die Dauer einer Sitzung gültige) Attributszertifikate an die Vertretersysteme ausstellen, um diesen Zugriff auf die eigenen Daten zu gewähren. Somit kann eine End-Benutzer-bezogene Zugriffskontrolle an der Datenhaltungsschicht durchgeführt werden.

In den meisten Szenarien ist es sinnvoll, die Freiheiten der an Delegierungen beteiligten Delegierungsgeber und –nehmer zu beschränken. Die (systemweiten) Regeln dazu können in einer so genannten Delegierungsrichtlinie (engl. *Delegation Policy*) festgelegt werden. Hierbei können folgende Aspekte berücksichtigt werden [BS00] [SWS+01]:

- **Mögliche Delegierungsgeber und -nehmer**: Um willkürliche Delegierungen zu unterbinden, kann der Kreis der möglichen Delegierungsgeber sowie –nehmer in einem System eingeschränkt werden. Dabei können explizite Angaben zu den hierfür vorgesehenen aber auch zu den hiervon ausgeschlossenen Entitäten (Subjekte) des Systems gemacht werden.

- **Administration:** Die tatsächliche Abwicklung (d.h. Administration) der Delegierungen kann teilweise oder ganz hierfür vorgesehenen Instanzen (Agenten) überlassen werden, um beispielsweise die Systeme der Delegierungsgeber zu entlasten.

- **Delegierung unter Vorbehalt**: Manchmal ist es sinnvoll, dass ein Delegierungsgeber auch solche Rechte delegieren kann, welche er zum Zeitpunkt der Delegierung gar nicht besitzt. Eine solche Delegierung sollte jedoch erst wirksam werden, nachdem der Delegierungsgeber das betroffene Recht erhielt.

- **Einverständnis des Delegierungsnehmers**: Die Delegierung kann grundsätzlich mit oder ohne Kenntnis beziehungsweise Einverständnis des beabsichtigten Delegierungsnehmers erfolgen.

- **Mehr-Augen-Prinzip**: Um die Delegierung bestimmter (z.B. hochsensibler) Berechtigungen geltend zu machen, kann ein Einverständnis mehrerer diese Rechte ebenfalls besitzender Subjekte erforderlich sein. Auf diese Weise kann eine gegenseitige Kontrolle der Delegierungsgeber erreicht werden.

- **Delegierungstiefe**: Delegierungen können mehrstufig fortgeführt werden. Das heißt, ein einmal an ein Subjekt delegiertes Recht kann von diesem an Andere delegiert werden. Dabei kann die maximale Anzahl an Delegierungsschritten, d.h. die maximale Delegierungstiefe, festgelegt werden. In diesem Kontext wird zwischen direkten und indirekten Delegierungen unterschieden. Eine direkte Delegierung zeichnet sich dadurch aus, dass sich zwischen Delegierungsgeber und –nehmer kein weiterer Delegierungsgeber befindet. Ansonsten wird die Delegierung indirekt genannt. In einem System, welches keine mehrstufige Delegierung unterstützt, kann eine jede Rechtevergabe als eine direkte Delegierung angesehen werden.

- **Permanenz:** Delegierungen können zeitlich befristet oder unbefristet gelten. Im ersten Fall wird eine Delegierung nach Ablauf einer definierten Gültigkeitsperiode ungültig. Ein weiteres Merkmal ist hier, ob die Gültigkeitsperiode eines durch Delegierung erhaltenen Rechts über die des delegierten Rechts hinausgehen darf.

- **Delegierungswiderruf:** Oftmals ist es erforderlich (zeitlich eingeschränkte) Delegierungen (frühzeitig) rückgängig zu machen. Dabei werden delegierte Rechte typischerweise durch den jeweiligen Delegierungsgeber entzogen.
  Falls der Delegierungsnehmer ein solches Recht zwischenzeitlich an Andere delegiert hat, ist zu bestimmen, welche dieser späteren Delegierungen rückgängig gemacht werden müssen. Eine mögliche (strenge) Vorgehensweise ist, sämtliche solche Delegierungen nichtig zu machen, insbesondere wenn Delegierungen stets bis auf ihren gemeinsamen Ursprung zurückgeführt werden müssen.

- **Monotonität:** Wenn ein Delegierungsgeber die jeweilig delegierten Rechte weiterhin behält, bezeichnet man die Delegierung als monoton. Dies kann jedoch unerwünscht sein, um z.B. Rechte garantiert nur einmal delegieren zu können.

- **Totalität:** Im Falle der Totalität kann ein Delegierungsgeber entweder alle seine Rechte delegieren oder gar keins. Die Delegierung einer (echten) Teilmenge seiner Rechte ist damit nicht zulässig.

Der X.509-Standard [ITU01] unterstützt grundsätzlich die Delegierung mit Hilfe von Attributszertifikaten. Dabei wird vor allem vorgeschrieben, dass während der Zugriffskontrolle festgestellt werden muss, ob ein durch Delegierung entstandenes Zertifikat von einer vertrauten Stelle (*Source of Authority*, SOA) stammt. Diese Ursprungsprüfung ist am einfachsten im Fall von direkten Delegierungen durchzuführen.

Wurden hingegen auch indirekte Delegierungen getätigt, so muss aus den von unvertrauten Stellen ausgestellten Zertifikaten klar hervorgehen, ob diese sich auf eine vertrauenswürdige Stelle zurückführen lassen. Zertifizierungsstellen, die im Zuge einer Delegierung Zertifikate ausstellen dürfen, welchen aber kein Vertrauen während der Zugriffskontrolle zuteil wird, werden im X.509-Standard *Attribute Authorities* (AA) genannt.
Eine AA darf ihre direkt von einer SOA oder indirekt von anderen AAs erhaltenen Berechtigungen delegieren und entsprechend Zertifikate signieren. Während der Zugriffskontrolle wird ihr jedoch nicht vertraut, da sie kein Besitzer der von der Delegierung betroffenen Ressourcen ist. Vielmehr können AAs als „Makler“ von Berechtigungen zu den Ressourcen der einzelnen SOAs angesehen werden.
Es kann in einem System natürlich vorkommen, dass eine Entität gleichzeitig Ressourcenbesitzer, d.h. eine SOA, sowie Delegierungsgeber zu fremden Ressourcen, d.h. eine AA ist.
Zur Unterstützung der Delegierung und der hiermit verknüpften Ursprüngprüfung wurden im X.509-Standard [ITU01] die Ergänzungsfelder *BasicAttConstraints*, *AuthorityAttributeIdentifier*, *DelegatedNameConstraints* und *AcceptableCertPolicies* vorgesehen. Hier werden die ersten beiden Felder näher behandelt:

- **BasicAttConstraints**: Dieses Feld enthält die beiden Einträge *Authority* und *PathLenConstraint***.** Authority nimmt den Wert *TRUE* an, wenn der Besitzer (*Holder*) des Attributszertifikats die im Zertifikat enthaltenen Berechtigungen (*Attributes*) grundsätzlich delegieren darf. In diesem Fall wird er also als eine AA im System anerkannt. Ansonsten gilt *Holder* als eine so genannte End-Entität (engl. *EndEntity*), welche keine Rechte delegieren darf.

BasicAttConstraintsSyntax ::= SEQUENCE {
Authority BOOLEAN DEFAULT FALSE,
PathLenConstraint INTEGER (0..MAX) OPTIONAL}

PathLenConstraint bestimmt die maximale Tiefe der Delegierungen welche von diesem Zertifikat ausgehen können. Nach einer Delegierung muss dieser Wert durch den Zertifikatsaussteller um eins verringert werden. Ist dieser Wert=0, so sind weitere Delegierungen unzulässig, unabhängig davon, ob *Holder* eine AA ist.

- **AuthorityAttributeIdentifier:** Dieses Feld funktioniert als eine Referenzliste auf die Zertifikate des Delegierungsgebers (*Issuer*), welche die jeweilig delegierten Rechte (*Attributes*) enthalten.

AuthorityAttributeIdentifierSyntax ::=
SEQUENCE SIZE (1..MAX) OF AuthAttId
AuthAttId ::= IssuerSerial

Hierbei identifiziert der Eintrag *AuthAttId* ein Attributszertifikat des jeweiligen Delegierungsgebers systemweit eindeutig: Er gibt den eindeutigen Namen des Ausstellers (*Issuer*) sowie die Seriennummer (*SerialNumber*) des Attributszertifikats an, welches die delegierten Rechte ursprünglich enthält. Da diese Erweiterung die Referenzierung mehrerer Zertifikate erlaubt, können baumartige Delegierungsstrukturen festgehalten werden.

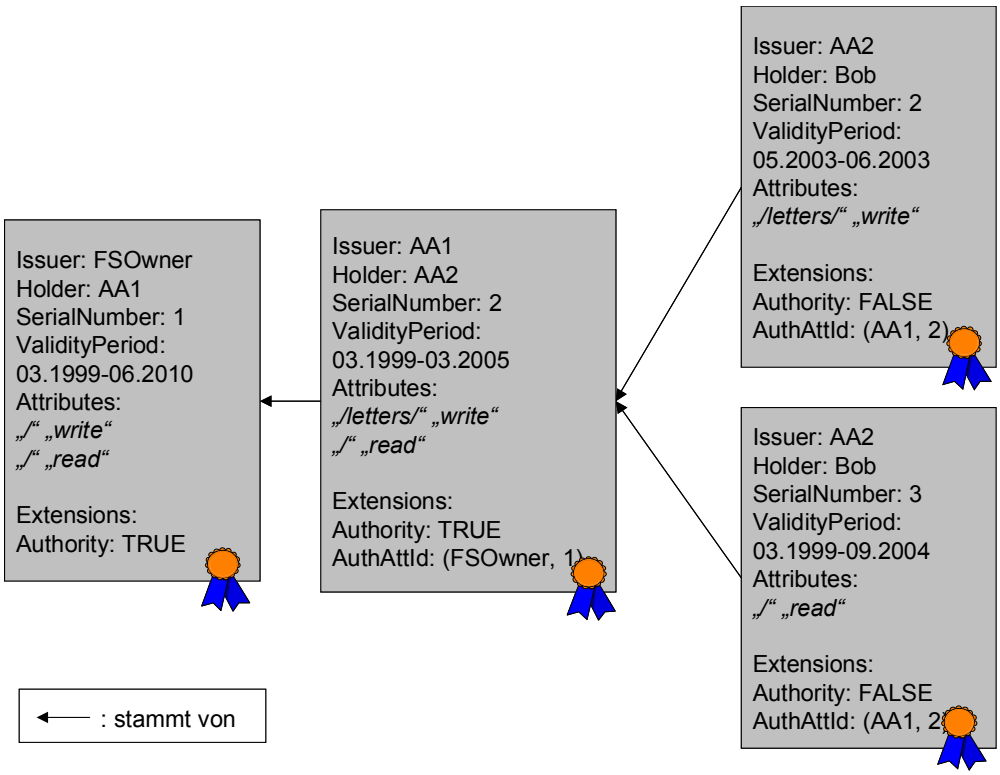


Abb. 2-9: Beispieldelegierungen mittels Attributszertifikaten

Abb. 2-9 zeigt durch Delegierungen entstandene Attributszertifikate. Die aus Abb. 2-7 bekannten Zertifikate von *Bob* (siehe Abb. 2-9 rechts) sind Ergebnisse mehrstufiger Delegierungen.

Diese wurden mit Hilfe der oben besprochenen Erweiterungsfelder (*Extensions*) in den Attributszertifikaten festgehalten. Dadurch lässt sich die so genannte Delegierungshistorie (engl. *Delegation History*) der jeweils delegierten Berechtigungen beziehungsweise der entsprechenden Zertifikate rekonstruieren:
Zunächst hat der Besitzer des Dateisystems namens *FSOwner* die Zertifizierungsstelle *AA1* zum Schreiben *(„write“)* und Lesen (*„read“*) des Wurzelverzeichnisses (*„/“*) und damit automatisch sämtlicher Unterverzeichnisse autorisiert. Außerdem wurde *AA1* erlaubt, Teile der erhaltenen Rechte zu delegieren (*Authority=TRUE*). Dies wird durch das Zertifikat mit einer Gültigkeitsperiode von März 1999 bis Juni 2010 bescheinigt (siehe Abb. 2-9 links).
AA1 hat einen Teil dieser Rechte noch im gleichen Monat an *AA2* delegiert, so dass diese das Verzeichnis */letters/* schreiben beziehungsweise das Wurzelverzeichnis lesen kann. Um festzustellen, woher diese Berechtigungen stammen, wurde in das Zertifikat von *AA2* der Verweis *AuthAttId=(FSOwner, 1)* eingefügt (siehe Abb. 2-9 Mitte). Dieser Verweis referenziert das entsprechende von *FSOwner* an *AA1* ausgestellte Zertifikat mit der Seriennummer *1* systemweit eindeutig.
Bob, der in diesem Fall eine End-Entität ist (*Authority=FALSE*), erhielt von *AA2* Teile deren Berechtigungen. Die beiden auf seinen Namen ausgestellten Zertifikate erlauben ihm das Schreiben in */letters/* sowie das Lesen der Wurzel (siehe Abb. 2-9 rechts). Wie man sieht, gelten diese Berechtigungen unterschiedlich lang, was seitens *AA2* die Ausstellung von zwei Attributszertifikaten erforderlich machte. Entsprechend weisen diese beiden Zertifikate durch *AuthAttId=(AA1, 2)* auf ihren gemeinsamen Ursprung hin.

2.3.3 Prozess der Prüfung von Attributszertifikaten

Mit Hilfe von Attributszertifikaten kann die Sicherheit der Zugriffskontrolle prinzipiell erhöht werden: Dank signierter Autorisierungsdaten können bestimmte Manipulationen der Zugriffsentscheidungen verhindert werden. Aus Sicht eines abgesicherten Systems stellt dies den eigentlichen Mehrwert einer zertifikatsbasierten Zugriffskontrolle gegenüber traditionellen Lösungen dar. Um diesen, zunächst lediglich theoretischen, Vorteil in einem System umzusetzen, muss sorgfältig überprüft werden, ob ein vorliegendes Attributszertifikat authentische, integre und möglichst aktuelle Autorisierungsdaten enthält.

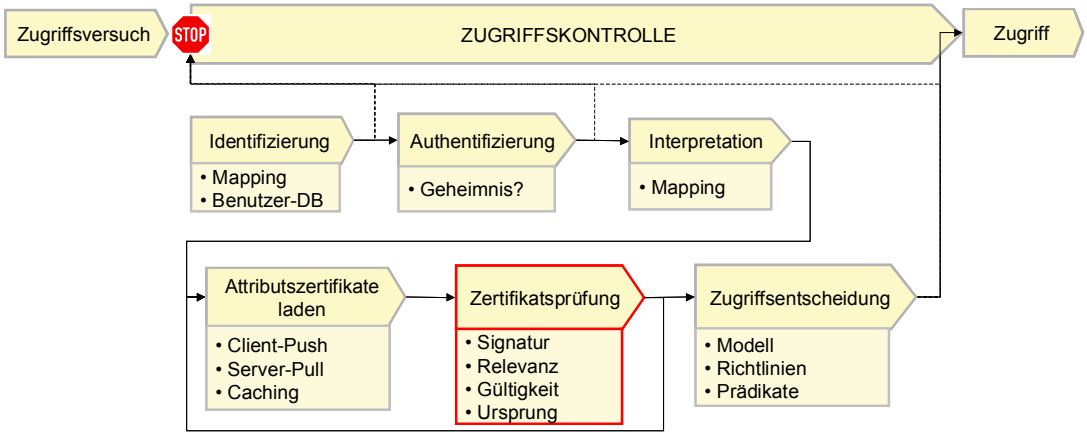


Abb. 2-10: Einordnung der Zertifikatsprüfung in den Zugriffskontrollprozess

In Abb. 2-10 wird die Einbettung dieser Prüfung in den aus Abb. 2-1 bereits bekannten Zugriffskontrollprozess dargestellt.

Wie man sieht, erfolgt dieser Prozessschritt unmittelbar bevor die Autorisierungsdaten des Zugreifenden ausgewertet werden, d.h. bevor entschieden wird, ob der Zugriff erlaubt oder verweigert wird. Eine Voraussetzung dieses Prozessschrittes ist, dass die jeweilig benötigten Attributszertifikate als Ergebnis des Teilprozesses „Attributszertifikate laden“ zur Verfügung stehen.
Um die Zugriffskontrolle zertifikatsbasiert zu realisieren, wird die in Kap. 2.1.2 eingeführte Funktion zur Zugriffskontrollentscheidung (ACDF) um eine Funktionalität zur Prüfung der Attributszertifikate erweitert [ITU01]. Ein solcher in Abb. 2-11 gezeigter erweiterter Verifizierer kann im „Attributszertifikate laden“ genannten Prozessschritt Attributszertifikate von einer Online-Datenbank erhalten.

Diese kann eine zentralisiert verwaltete (verteilte) Datenablage (Server-Pull-Modell) oder auch eine eigene (lokale) Ablage des Zugreifenden (Client-Push-Modell) sein. In beiden Fällen gilt, dass diese Speicherinstanz sowie deren Betreiber primär als nicht vertrauenswürdig eingestuft werden. Ein Verifizierer erkennt nur die SOAs als vertraut an, welche die vom jeweiligen Zugriff betroffenen Ressourcen besitzen [ITU01].
Die hinter einer solchen vertrauten SOA stehenden Systeme sind am operativen Zugriffskontrollprozess nicht beteiligt. Sie sorgen für die gesicherte Ausstellung und Pflege der von dieser Stelle verwalteten Attributszertifikate, und sind aus Sicht der Zugriffskontrolle offline. An die jeweilige Online-Datenbank werden nur die (z.B. periodisch aktualisierten) Daten übermittelt, welche zur Unterstützung der Zugriffskontrolle benötigt werden [ITU01]. Zu solchen Daten gehören beispielsweise die in Kap. 3 behandelten Sperrdaten.

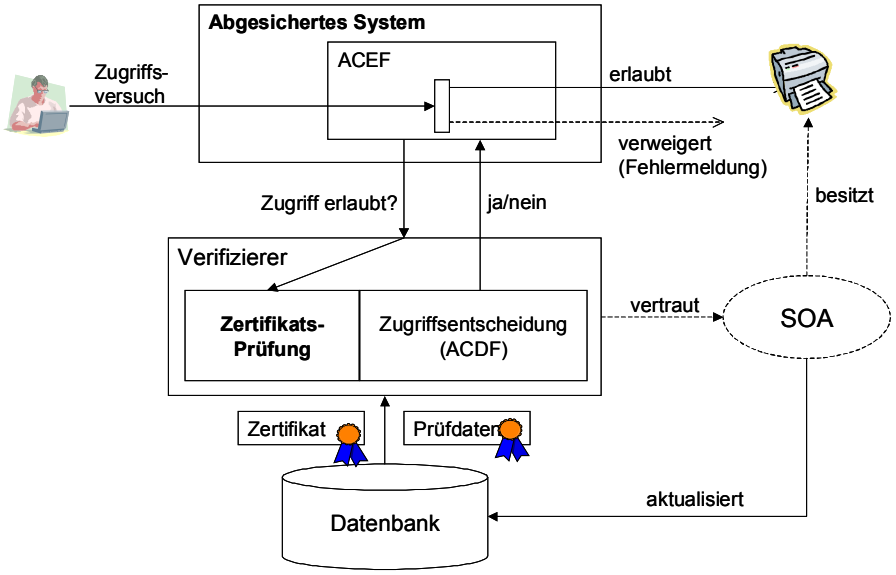


Abb. 2-11: Überprüfung der Attributszertifikate in einem System

Ein Verifizierer muss während der Zugriffskontrolle von vertrauten Offline-Instanzen in der Vergangenheit ausgestellte (evtl. delegierte) Attributszertifikate auf deren Gültigkeit überprüfen, welche ihn über nicht vertrauenswürdige Online-Instanzen erreichen. Zu solchen als nicht vertrauenswürdig eingestuften Instanzen gehören neben Datenablagen auch sämtliche Kommunikationssysteme, wie z.B. das Internet.
Ähnlich zur gesamten Zugriffskontrolle kann der Prozess der Zertifikatsprüfung als eine Folge von Teilprozessen gesehen werden, wie das in Abb. 2-12 gezeigt wird.

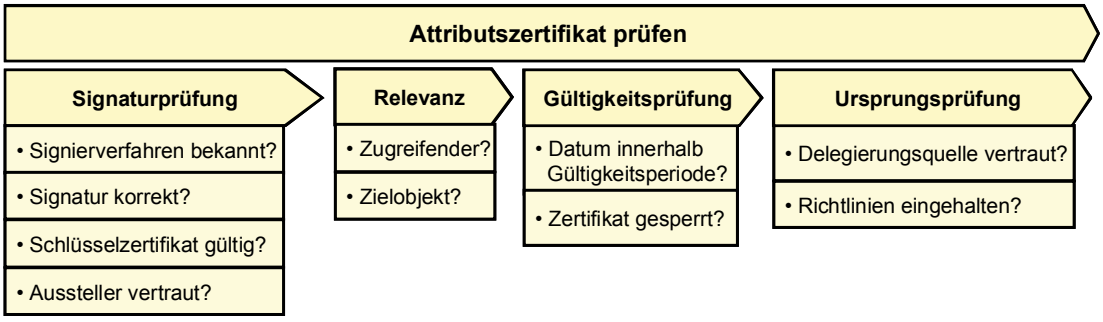


Abb. 2-12: Prozessschritte zur Prüfung eines Attributszertifikats

Dieser Prozess sollte für jedes der Zugriffskontrolle vorliegende Attributszertifikat wiederholt ausgeführt werden. Die Prüfung eines bestimmten Attributszertifikats muss natürlich nicht bei jedem (identischen) Zugriffsversuch in dieser Form erfolgen.

Eine Zwischenspeicherung (engl. *Caching*) bereits geprüfter Attributszertifikate kann einen Verifizierer diesbezüglich entlasten. Ein erneutes Überprüfen beziehungsweise das Aufräumen des Cache wird beispielsweise notwendig, wenn die dort gespeicherten Daten nicht mehr aktuell genug sind. Der Prozess der Überprüfung besteht aus mehreren Phasen, welche in Abb. 2-12 in einer sinnvollen Reihenfolge dargestellt werden.

- **Signaturprüfung:** Einen wichtigen Teil der Auswertung eines Attributszertifikats bilden die adäquate Signaturprüfung sowie die Feststellung, ob das Zertifikat von einer vertrauenswürdigen Stelle (*Source of Authority*, SOA) ausgestellt wurde. Die allgemeinen Berechnungsschritte zur Signaturprüfung wurden in Abb. 2-3 bereits vorgestellt.
  Um diese an einem Attributszertifikat auszuführen, muss das jeweilig verwendete Signierverfahren aus dem (in Kap. 2.2.4 vorgestellten) Eintrag *Signature* entnommen werden. Außerdem wird der passende öffentliche Schlüssel der signierenden Stelle benötigt. Deren systemweit eindeutiger Name kann anhand des Felds *Issuer* ermittelt werden. An dieser Stelle kann ein möglicher Berührungspunkt der Zugriffskontrolle zu einer Public-Key Infrastruktur (PKI) identifiziert werden: Um zu überprüfen, ob der Aussteller (*Issuer*) beim Signieren des Attributszertifikats ein authentisches und gültiges Schlüsselpaar verwendet hat, können die Dienste einer PKI in Anspruch genommen werden.
  Wie bereits in Kap. 2.2.4 erwähnt, kann hierfür ein Schlüsselzertifikat unter dem (optionalen) Eintrag *baseCertificateID* innerhalb des *Issuer*-Felds referenziert werden. Die Überprüfung dieses Schlüsselzertifikats kann beispielsweise mit Hilfe der entsprechend integrierten Verifizierungssoftware der jeweiligen PKI erfolgen [NA+01b]. Erwies sich der öffentliche Schlüssel als authentisch und gültig, so können die jeweilig benötigten Berechnungen mit diesem Schlüssel durchgeführt werden.
  Laufen diese Berechnungen fehlerfrei ab, so muss noch festgestellt werden, ob der jeweilige Aussteller des Attributszertifikats eine als vertrauenswürdig eingestufte Stelle (SOA) ist. Wenn das der Fall ist, kann der Prozess der Zertifikatsprüfung ohne weiteres fortgesetzt werden. Andernfalls muss das Zertifikat abgelehnt werden. Eine Ausnahme hiervon bilden Szenarien, in welchen die Delegierung der Rechte unterstützt wird. Ist das der Fall, so muss der Ursprung der evtl. mehrstufig getätigten Delegierungen festgestellt werden (siehe weiter unten), deren Ergebnis das vorliegende Zertifikat sein kann.

- **Relevanzprüfung:** Liegen einem Verifizierer in einer bestimmten Zugriffssituation mehrere authentische und von vertrauten Stellen stammende Attributszertifikate vor, so können durch eine geschickte Ausfilterung irrelevanter Zertifikate weitere Prüfungsschritte erspart werden. Bei einer solchen Relevanzprüfung können verschiedene Aspekte von Bedeutung sein. Beispielsweise kann von Interesse sein, ob der Name des jeweilig Zugreifenden zum Feld *Holder* des vorliegenden Attributszertifikats passt. Ebenfalls kann untersucht werden, ob der Zugriff auf ein im Zertifikat reflektiertes Zielobjekt zielt.

- **Gültigkeitsprüfung:** Nach diesen Prozessschritten bleibt noch zu entscheiden, ob ein Attributszertifikat, dessen Signatur gültig ist und den Zugreifenden zum Zugriff (zumindest teilweise) auch autorisieren würde, noch gültig ist. Ein Attributszertifikat kann grundsätzlich nur innerhalb seiner im Feld *AttrCertValidityPeriod* vorgesehenen Gültigkeitsperiode verwendet werden. Außerhalb des hier festgelegten Zeitraums (*notBeforeTime, notAfterTime*) kann das Zertifikat für die Zugriffskontrolle nicht benutzt werden [ITU01].
  Deshalb muss festgestellt werden, ob der aktuelle Zeitpunkt der Verifizierung (Datum, Uhrzeit) überhaupt in diese Gültigkeitsperiode fällt. Dies setzt natürlich die Verwendung synchronisierter Uhren sowie einheitliche Zeit-Darstellungsformate im System voraus.
  Nimmt ein Attributszertifikat auch diese Hürde erfolgreich, bleibt noch ein in der Literatur vieldiskutiertes Problem offen: Zum Zeitpunkt der Erzeugung des Attributszertifikats hat dessen Aussteller (*Issuer*) eine beschränkte Gültigkeitsperiode für die Bindung zwischen dem zertifizierten Namen (*Holder*) und den Autorisierungsdaten (*Attributes*) vorgesehen.
  Zum Einen besagt ein solches Attributszertifikat, dass der Zertifizierte unter den am Anfang der Gültigkeitsperiode herrschenden organisatorischen, technischen, rechtlichen usw. Bedingungen würdig *war*, die im Zertifikat festgelegten Berechtigungen zu erhalten. Zum Anderen spiegelt das Zertifikat den Optimismus dessen Ausstellers wieder, dass sich diese Bedingungen bis zum Ende der vorgesehenen Gültigkeitsperiode nicht ändern werden.
  Es können jedoch Probleme entstehen, wenn innerhalb dieses Zeitraums unerwartete, d.h. ungeplante Ereignisse auftreten. Ein Beispiel hierfür: Durch den Wechsel seiner bei der Firma ausgeübten Tätigkeiten müssen die Zugriffsrechte von *Bob* gemindert werden. Die hiervon betroffenen noch nicht abgelaufenen Zertifikate dürfen nicht weiter benutzt werden, was aber nicht heißt, dass *Bob* (oder jemand anders) dies nicht versucht zu tun. Um die hieraus resultierenden fälschlichen Zugriffsentscheidungen zu verhindern, müssen die von den jeweiligen Änderungen betroffenen Zertifikate frühzeitig widerrufen (engl. *revoked*) werden (siehe auch den Lebenszyklus eines Zertifikats in Abb. 2-8).
  Die Überprüfung, ob ein früher ausgestelltes Attributszertifikat immer noch als gültig akzeptiert werden kann, bedarf der Bereitstellung hierfür vorgesehener Daten sowie eines sicheren Mechanismus zur Auswertung derselbigen Daten. Die hierfür aus der Literatur bekannten und vor allem in Public-Key Infrastrukturen eingesetzten Methoden werden im nächsten Kapitel vorgestellt und bewertet.

- **Ursprungsprüfung:** Die Ursprungsprüfung eines Attributszertifikats bedeutet die Bestimmung und Prüfung der Delegierungshistorie des Zertifikats.
  Hierfür muss mit Hilfe der in Kap. 2.3.2 vorgestellten delegierungsbezogenen Erweiterungsfelder die evtl. vertraute Quelle der Delegierungen bestimmt werden.
  Außerdem kann geprüft werden, ob die Delegierungen gemäß der im System geltenden Delegierungsrichtlinien, wie z.B. bezüglich einer maximal erreichbaren Delegierungstiefe, erfolgten.
  Die Rekonstruktion der in der Vergangenheit getätigten Delegierungen kann allerdings ein mehrfach wiederholtes Aufrufen der in Abb. 2-10 dargestellten Prozessschritte „Attributszertifikate laden“ sowie „Zertifikatsprüfung“ erforderlich machen. Haben die auf diese Weise ermittelten Delegierungen einen vertrauten Ursprung (SOA), so kann das gefragte Attributszertifikat akzeptiert werden. Andernfalls muss es abgelehnt und der Prüfungsprozess abgebrochen werden.
  Die mehrfach wiederholten Zertifikatsprüfungen innerhalb der Ursprungsprüfung verursachen einen zusätzlichen Aufwand während der Zugriffskontrolle. Sie können zu inakzeptabel langen Wartezeiten in einem System führen. Bekannte Zertifikatsmanagementsysteme, wie z.B. PERMIS [CO02], unterstützen deshalb keine Delegierung. Sichere Techniken zur Reduktion dieses Aufwands sind Gegenstand von Kap. 6.

3 Stand der Technik

Durch die Verwendung von Attributszertifikaten kann die Sicherheit der Zugriffskontrolle in einem System erhöht werden. Der mit der Prüfung von (evtl. delegierten) Attributszertifikaten verbundene Aufwand sollte dabei möglichst gering ausfallen. Entstehen durch die Zertifikatsprüfung lange Wartezeiten in einem abgesicherten System, führt das seitens des Betreibers sowie der Benutzer des Systems zu Akzeptanzproblemen dieser Technik. Diese Wartezeiten entstehen im Wesentlichen durch berechnungsbeziehungsweise kommunikationsbezogene Verzögerungen (im Weiteren „Kosten“ genannt) während der Zertifikatsprüfung.

Der Fokus der Betrachtungen liegt auf den Mechanismen, die beim Prozessschritt der Gültigkeitsprüfung (siehe Kap. 2.3.3) von Attributszertifikaten eingesetzt werden können. Nachfolgend werden bekannte Verfahren hierfür vorgestellt und bezüglich ihrer Sicherheit und durchschnittlicher Kosten analysiert. Diese, maßgeblich für die Prüfung von Schlüsselzertifikaten entwickelten Verfahren lassen sich hinsichtlich der Art der jeweils verwendeten Daten in drei Kategorien einordnen: Techniken, die sich auf so genannte Sperrdaten verlassen, werden in Kap. 3.1 behandelt. Die Konsequenzen der Aufstellung so genannter Online-Zertifikatsstatus-Dienste zur Unterstützung der Gültigkeitsprüfung werden in 3.2 diskutiert. In Kap. 3.3 werden Ansätze vorgestellt, deren Basis Gültigkeits- statt Sperrdaten sind.

Im Zentrum der Kostenbetrachtungen steht der die Zugriffskontrolle realisierende Verifizierer. Es gilt die auf dessen Seite entstehenden durchschnittlichen Kosten (betrachtet im O-Kalkül [Goo97]) während der Zertifikatsprüfung zu minimieren, um die gesamte Zugriffskontrolle möglichst effizient zu gestalten.

- **Berechnungskosten:** Zum Einen wird die Anzahl der jeweilig benötigten in Kap. 2.2.2 betrachteten Signatur-Prüfungen sowie eventuelle -Erstellungen seitens eines Verifizierers im Auge behalten. Dieser Aspekt gewinnt insbesondere in Systemen an Bedeutung, in welchen häufig mehrere Zertifikate gleichzeitig geprüft werden müssen, die von unterschiedlichen Stellen stammen können. Neben diesem Aspekt spielt natürlich die Performanz des jeweiligen Signierverfahrens, d.h. Hashfunktionen und Public-Key-Verfahren eine Rolle. Diesbezügliche (algorithmenabhängige) Optimierungsmöglichkeiten liegen jedoch außerhalb der Betrachtungen dieser Arbeit. Zum Anderen werden die Kosten der Suche in dem, einem Verifizierer jeweilig vorliegenden, Datenbestand erfasst, um den aktuellen Status eines Zertifikats zu ermitteln.

- **Kommunikationskosten:** Die jeweilig benötigten Daten, um den aktuellen Status eines Zertifikats zu überprüfen, müssen in einer den Verifizierern zugänglichen Online-Datenbank abgelegt werden. Die Auslastung dieser Datenbank sowie der involvierten Kommunikationsnetze hängt unter anderem vom Umfang der Daten ab, welche zur Prüfung eines Zertifikats benötigt werden. Deshalb werden die einzelnen Methoden auch diesbezüglich bewertet.

Neben diesen „operativen“ Kosten hängt die Akzeptanz der diskutierten Methoden auch mit den „administrativen“ Kosten der Pflege und Verteilung der jeweiligen Daten zusammen. Dabei werden ebenfalls Berechnungskosten sowie die Kosten der Kommunikation zwischen der jeweiligen Zertifizierungsstelle und der die Daten speichernden Datenbank betrachtet.

3.1 Gültigkeitsprüfung aufgrund von Sperrdaten

Die Gültigkeitsprüfung von Attributszertifikaten kann durch die Versorgung der Verifizierer mit so genannten Sperrdaten (engl. *Revocation Data*) zu den einzelnen Zertifikaten unterstützt werden. Solche Daten werden nach der Sperrung eines bestimmten Zertifikats erzeugt und gesichert. Folglich wird eine indirekte Beantwortung der Gültigkeitsfrage möglich: Liegt zu einem Zeitpunkt keine Sperrinformation zu einem bestimmten (in der Vergangenheit ausgestellten) Attributszertifikat vor, so gilt dieses Zertifikat als gültig. Bei der Verwendung von Sperrdaten müssen folgende operative und administrative Aspekte berücksichtigt werden:

- **Quelle akzeptierter Sperrdaten:** Es sollten grundsätzlich nur von vertrauenswürdigen Stellen stammenden Sperrdaten akzeptiert werden. Dementsprechend können Sperrdaten durch die Zertifizierungsstellen zu den eigens ausgestellten Attributszertifikaten erzeugt werden [ITU01]. Alternativ dazu kann diese Aufgabe durch eine zentrale, von allen Verifizierern als vertrauenswürdig eingestufte, so genannte Sperrautorität (engl. *Certificate Revocation Authority,* CRA) übernommen werden [Koc98]. In diesem Fall spricht man von indirekter Sperrung (engl. *Indirect Revocation*) [AL99]. Bei der Etablierung einer solchen Instanz muss berücksichtigt werden, dass diese beliebige Zertifikate für ungültig erklären kann.

- **Beantragung einer Sperrung:** Die zur Zertifikatssperrung berechtigten Stellen müssen entsprechend gesicherte elektronische oder „out-of-band“ Schnittstellen zur Beantragung von Zertifikatssperrungen etwa durch die Zertifikatsbesitzer zur Verfügung stellen. Gelingt es einem Angreifer beispielsweise einen über das Internet gesendeten ungeschützten Sperrantrag zu fälschen oder zu blockieren, so wird das zu sperrende Attributszertifikat weiterhin für gültig gehalten.

- **Schutz gegen Fälschungen:** Sperrdaten müssen gegen nachträgliche Manipulationen geschützt werden. Gelingt es einem Angreifer etwa, die vorhandene Sperrinformation zu einem widerrufenen Zertifikat unbemerkt zu entfernen, so wird dieses Zertifikat weiterhin als gültig akzeptiert. Um ihre Authentizität und Integrität zu schützen, müssen Sperrdaten mit einer Signatur der jeweilig zuständigen Stelle versehen werden. Dadurch erhöhen sich allerdings die Berechnungskosten während der Zertifikatsprüfung: Neben den Signaturen an Attributszertifikaten müssen auch die an den zugehörigen Sperrdaten geprüft werden.

- **Speicherung:** Sperrdaten müssen die Verifizierer möglichst schnell erreichen. Dies motiviert die Aufstellung einer so genannten Sperrdatenbank. Eine solche Sperrdatenbank wäre selbst dann notwendig, wenn Attributszertifikate im Besitz der Zugreifenden sind und nicht etwa in einer Zertifikatsdatenbank liegen. Das ist im Client-Push-Modell (siehe Kap. 2.1.2) der Fall. Der eine allgemeine Vorteil dieses Modells, nämlich die Möglichkeit zum Verzicht auf Datenbankanfragen während der Zugriffskontrolle, geht dadurch verloren.

Um einen schnellen Zugriff (Suche) auf die jeweilig gesuchten Sperrinformationen zu ermöglichen, sollten die Sperrdaten in der Datenbank entsprechend indiziert vorliegen. Außerdem kann durch Strukturierung der Umfang der Sperrinformation zu einem bestimmten Attributszertifikat niedrig gehalten werden. Dadurch kann eine Reduktion der Kommunikations- und Berechnungskosten während der Zugriffskontrolle erreicht werden. Die Robustheit und Ausfallsicherheit der Sperrdatenbank sicherzustellen ist ebenfalls wichtig: Sind die aktuellen Sperrdaten unerreichbar, so könnten Verifizierer deren Aktualisierung „verpassen“ und dadurch zwischenzeitlich ungültig gewordene Zertifikate akzeptieren.

- **Aktualisierung:** Grundsätzlich gilt, dass die jeweilige Sperrtechnik der Änderungsdynamik der Autorisierungsdaten möglichst gut folgen muss. Sperrdaten müssen deshalb regelmäßig aktualisiert und verteilt werden. Die Häufigkeit der Aktualisierungen hat einen wesentlichen Einfluss auf die Sicherheit der Zertifikatsprüfungen: Die frühzeitige Sperrung eines Attributszertifikats bleibt den Verifizierern bis zur nächsten Aktualisierung der Daten unbekannt.
  Sperrdaten werden in kommerziellen Public-Key Infrastrukturen (PKI) heute üblicherweise in einem wöchentlichen Rhythmus aktualisiert, während die reflektierten Schlüsselzertifikate typischerweise für einige Jahre ausgestellt werden. Im Fall von Attributszertifikaten mit ähnlich langer Gültigkeitsperiode reicht eine wöchentliche Aktualisierung erwartungsgemäß nicht aus, da sich Autorisierungsdaten dynamischer verhalten als Schlüsselpaare. Verifizierer sollten die Gültigkeitsprüfung natürlich immer anhand der aktuellsten Sperrdaten durchführen.

- **Archivierung:** Beispielsweise für die Zwecke einer Nachweisführung kann die langfristige Archivierung von Sperrdaten erforderlich sein. Die Beantwortung der Frage, ob ein bestimmtes Zertifikat zu einem bestimmten früheren Zeitpunkt gültig war, bedarf der damals aktuellen Sperrdaten.

Diese Aspekte beziehungsweise Anforderungen implizieren, dass ein Zertifikatsmanagementsystem eigentlich zwei Systeme kapseln muss: Eins für die Verwaltung der gültigen Attributszertifikate und eins für die Verwaltung von Sperrdaten zu bereits ungültig gewordenen, d.h. aus Sicht der Zugriffskontrolle wertlosen, Attributszertifikaten. Diese Sperrdaten müssen (über ihren gesamten Lebenszyklus hinweg) ähnlichen Sicherheitskriterien entsprechen wie die Zertifikate selbst, wobei sie häufiger aktualisiert und verteilt werden müssen.

3.1.1 Listenbasierte Sperrtechniken

Eine einfach implementierbare Lösung, den Sperrstatus von Zertifikaten zu prüfen, bietet die Verwendung von so genannten Sperrlisten (engl. *Certificate Revocation List*, CRL). Eine Sperrliste ist eine digital signierte Liste, die Informationen zu frühzeitig widerrufenen Zertifikaten sowie zur Prüfung der Liste enthält. In [ITU01] wurde ein Sperrlistenformat namens „CRLv2“ definiert, welches eine unsichere erste Version „CRLv1“ ablöste [Cho96]. Aus „CRLv2“ wurde durch die bereits erwähnte PKIX-Gruppe ein eigenes Sperrlistenformat abgeleitet [HFP+99] [PHB02] [HP+02]. Folglich kommen in den meisten heutigen PKI-Produkten diese Art Sperrlisten zum Einsatz [AL99].

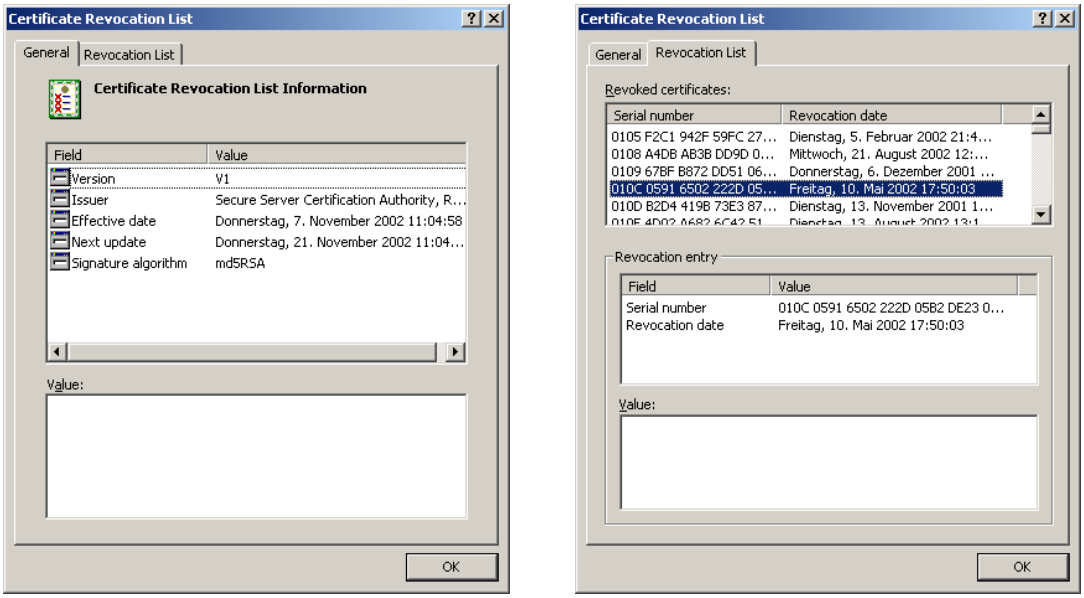


Abb. 3-1: Sperrliste eines kommerziellen PKI-Anbieters

In Abb. 3-1 wird die Sperrliste eines kommerziellen PKI-Anbieters gezeigt. Ein jedes Sperrlistenelement enthält einen eindeutigen Identifikator eines gesperrten Zertifikats sowie Angaben zu dessen Sperrung. Im Fall von X.509-Attributszertifikaten ist das Feld *SerialNumber* ein geeigneter Identifikator [ITU01]. Neben diesem können in jedem Sperrlistenelement beispielsweise Zeitpunkt und Ursache der Sperrung des jeweiligen Zertifikats angegeben werden (siehe Abb. 3-1 rechts) [AL99]. Um die signierte Sperrliste zu überprüfen, können der Signaturalgorithmus, die Seriennummer der Liste, der Name des Listenausstellers, das Datum der Veröffentlichung und die nächste geplante Aktualisierung eingefügt werden (siehe Abb. 3-1 links).
Um den Sperrstatus eines Attributszertifikats zu prüfen, muss zunächst die passende Sperrliste gefunden werden. Diese kann durch einen im Zertifikat reflektierten Speicherort, den so genannten *CRL Distribution Point* (CDP) beispielsweise in Form eines URL erreicht werden. Nach Erhalt der Liste kann sie nach dem Identifikator des gefragten Zertifikats durchsucht werden.

Ein Problem mit Sperrlisten ist ihre (wachsende) Länge *r*. Laut Schätzungen werden im Schnitt 10% der Schlüsselzertifikate frühzeitig gesperrt [NIST94]. Ähnlich fundierte Schätzungen speziell für Attributszertifikate sind dem Autor leider nicht bekannt. Angenommen, Attributszertifikate hätten ein mit dem der Schlüsselzertifikate vergleichbar langes Gültigkeitsintervall (d.h. einige Jahre), dürfte dieser Anteil höher ausfallen. Als Grund kann die erwartete höhere Dynamik der Autorisierungsdaten in einem System genannt werden.
Neben dem Anteil der durchschnittlich zu sperrenden Attributszertifikate wird die erwartete Länge einer Sperrliste zusätzlich durch die Dauer der Zeitperiode beeinflusst, während der ein gesperrtes Attributszertifikat in der Liste reflektiert werden muss. Die Gültigkeitsprüfung eines Attributszertifikats wird genau dann nicht gefährdet, wenn der entsprechende Eintrag zu jedem bereits gesperrten Zertifikat mindestens bis zu dessen Ablaufdatum (*notAfterTime*) in der Liste bleibt.
Erst nach diesem Zeitpunkt kann die Sperrinformation zum abgelaufenen (aber schon lange nicht mehr gültigen) Attributszertifikat aus der Sperrliste entfernt werden. Das impliziert, dass die durchschnittliche Länge *r* einer Sperrliste proportional zur Anzahl *n* der mit Hilfe dieser Sperrliste geprüften Attributszertifikate ist.

Wächst also die Anzahl der gültigen Attributszertifikate innerhalb eines Systems, wird die Länge der Sperrlisten dementsprechend zunehmen. Dies wird erwartungsgemäß der Fall in neuen Systemen sein. Wenn aber etwa gleich viele neue Attributszertifikate ausgestellt werden wie ablaufen, wird sich die Länge *r* der jeweiligen Sperrliste bei einem zu *n* proportionalen Wert stabilisieren. Die Sperrliste in Abb. 3-1 hat beispielsweise mehrere Tausend Einträge und einen Umfang von 650 Kilobyte. Die Kosten der Gültigkeitsprüfung mit Hilfe einer Sperrliste während einer Zertifikatsprüfung sind die folgenden:

- **Kommunikationskosten:** Sollten Zertifikate einer Zertifizierungsstelle geprüft werden, so muss die passende Sperrliste von der im jeweiligen CDP angegebenen Sperrdatenbank heruntergeladen werden. Dies entspricht einem Aufwand von *O(n)*. Diese Kosten fallen natürlich weg, wenn die jeweilig notwendige (noch aktuelle) Sperrliste dem Verifizierer bereits vorliegt.
  In den meisten Systemen (allen voran Web-Anwendungen) wird heute hauptsächlich wegen der hohen Kommunikationskosten auf eine Sperrstatusprüfung gänzlich verzichtet, und es werden lieber die hiermit verbundenen Risiken in Kauf genommen.

- **Berechnungskosten:** Diese setzen sich im Wesentlichen aus den Kosten der Suche innerhalb der Liste nach einem bestimmten Zertifikat (anhand von *SerialNumber*) und der Signaturprüfung der Liste zusammen.
  Ist die Sperrliste unsortiert, fallen die Suchkosten im Bereich *O(n)*, andernfalls kann die Suche in $O(\log_2 n)$ Schritten ausgeführt werden.

Ebenfalls lassen sich die Verwaltungskosten der Liste identifizieren: Zum Einfügen (Löschen) eines Listenelements in eine sortierte Sperrliste bedarf es $O(\log_2 n)$ Suchschritte. Ist die Sperrliste unsortiert, kann ein neues Element in *O(1)* eingefügt werden, wobei das Löschen mit einem Aufwand von *O(n)* verbunden sein wird. Hinzu kommen die Berechnungskosten, um die Liste zu signieren.

- **Partitionierung der Sperrdaten:**

Eine Möglichkeit zur Senkung der Kommunikationskosten ist die Limitierung der maximalen Länge von Sperrlisten [AL99] [AZ98]. Dies kann durch eine Unterteilung der Sperrdaten in mehrere Sperrlisten (siehe Abb. 3-2 Teil *a*) erfolgen.
Hierbei stellt eine Zertifizierungsstelle mit insgesamt *n* verwalteten Attributszertifikaten *p* Sperrlisten für maximal jeweils $r=\lceil n/p \rceil$ Attributszertifikate auf. Dabei sollte ein Attributszertifikat mit der Seriennummer *id* den Speicherort (CDP) der zugehörigen Sperrliste CRL_i $(0<i\leq p)$ enthalten. Für diese Sperrliste gilt, dass sie Sperrinformationen zu Attributszertifikaten mit $(i-1)*\lceil n/p \rceil \leq id < i*\lceil n/p \rceil$ enthält. Durch eine geeignete Wahl von *p* kann ein Kostenoptimum für die jeweilige Betriebsumgebung gefunden werden. In einem Extremfall könnte natürlich *n=p* beziehungsweise *r=1* gewählt werden, so dass die Gültigkeit eines Attributszertifikats mit Hilfe einer exklusiv für dieses ausgestellten Sperrliste zu prüfen wäre.
Dieser Ansatz kann die durchschnittlichen Kommunikationskosten während der Gültigkeitsprüfung eines einzigen Attributszertifikats auf $O(r=\lceil n/p \rceil)$ reduzieren, da die Sperrlisten die fixe Länge $r=\lceil n/p \rceil$ nicht überschreiten können.

Sollten aber gleichzeitig $k>1$ durch dieselbe Autorität ausgestellte Attributszertifikate geprüft werden, welche nun in verschiedenen Sperrlisten reflektiert werden können, erhöht sich der Kommunikationsaufwand auf $\lceil k/2 \rceil * O(r=\lceil n/p \rceil)$. Die Berechnungskosten steigen ebenso, weil ein Verifizierer nun statt eine, im Durchschnitt $1+k/2$ Sperrlisten-Signaturen prüfen muss. Im Vergleich zu den nicht partitionierten Sperrlisten ist hier auch mit p-fach erhöhten administrativen Kosten zu rechnen.

- **Dynamische Partitionierung von Sperrdaten:**

Wird der oben eingeführte Partitionierungsparameter p verändert, um etwa das erwähnte Optimum im laufenden Betrieb zu finden, so führt das sowohl zu einer Neuausstellung der Sperrlisten als auch der betroffenen Attributszertifikate (evtl. alle). Ein Attributszertifikat, das nach der Neu-Partitionierung eine andere Sperrliste als zuvor (durch seinen CDP-Eintrag) referenzieren muss, muss nämlich neu ausgestellt werden.
Um dies zu vermeiden, wurden in [AZ98] die so genannten *Redirect CRLs* (RCRL, siehe Teil *b* in Abb. 3-2) vorgestellt.

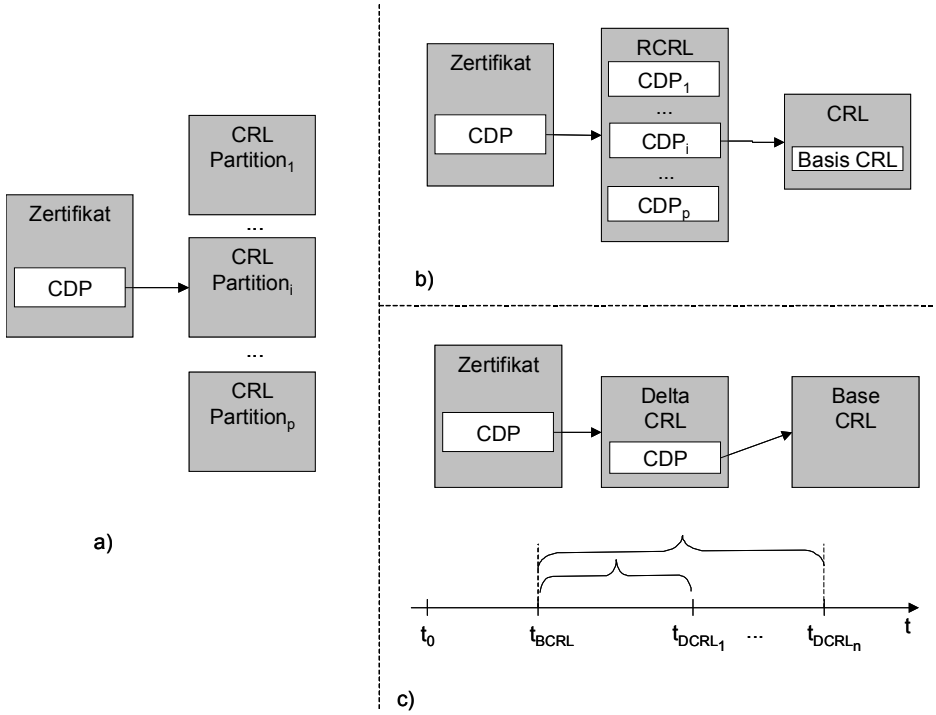


Abb. 3-2 Verschiedene CRL-Techniken

Eine RCRL ist eine signierte „Vermittlersperrliste“ zwischen den Zertifikaten und den nun dynamisch (re-)partinionierbaren eigentlichen Sperrlisten. In den Attributszertifikaten gibt jeweils ein CDP-Eintrag die fixe Adresse einer RCRL an. Diese RCRL beinhaltet wiederum eine Liste von CDP-Einträgen auf p Sperrlisten mit einer maximalen Länge von $\lceil n/p \rceil$.
Erfolgt durch die Änderung des Parameters p eine Neu-Partitionierung dieser Sperrlisten, so werden diese sowie die RCRL entsprechend geändert und signiert. Die neue RCRL bleibt aber weiterhin unter der alten Adresse für die Verifizierer erreichbar. Folglich müssen die CDP-Einträge in den Attributszertifikaten nicht geändert werden, was die Neuausstellung der Attributszertifikate unnötig macht.

Die Kommunikationskosten während einer Gültigkeitsprüfung mit dieser Technik fallen in den Bereich $O(p)+O(r=\lceil n/p \rceil)$.
Zum Auffinden der jeweils benötigten aktuellen Sperrliste muss nämlich auch die aktuelle Vermittlerliste mit p Einträgen heruntergeladen werden. An die jeweilige Sperrdatenbank müssen dabei mindestens zwei Anfragen gestellt werden, da die Adresse der zugehörigen Sperrliste nicht im Attributszertifikat enthalten ist. Nachteilig erscheint weiterhin, dass zur Prüfung eines einzigen Attributszertifikats nun mindestens zwei Sperrlisten samt ihrer Signaturen überprüft werden müssen.

- **Inkrementierung von Sperrdaten:**

Eine weitere Alternative, den Kommunikationsaufwand während der Sperrlistenprüfung zu reduzieren, ist die Inkrementierung der Sperrdaten mithilfe so genannter Basis-Sperrlisten (engl. *Base CRL*, BCRL) und Delta-Sperrlisten (engl. *Delta CRL,* DCRL) [ITU01] [AL99]. Hierbei gibt eine Delta-Sperrliste der Länge *dr* zum Zeitpunkt t_{DCRL} sämtliche seit dem Zeitpunkt der Veröffentlichung der Basis-Sperrliste t_{BCRL} gesperrte Zertifikate an. Dies verdeutlicht Abb. 3-2 Teil c. Die Basis-Sperrliste hat stets die fixe Länge $br=r-dr$, wobei r die Gesamtanzahl der aktuell gesperrten Zertifikate bezeichnet.
Ein Verifizierer, welcher die Basis-Sperrliste bereits heruntergeladen hat, muss zu jedem späteren Zeitpunkt die (wahrscheinlich) kürzere aktuellste Delta-Sperrliste von der Sperrdatenbank herunterladen und prüfen. Der jeweilige Kommunikationsaufwand liegt hier also bei $O(dr)$. Nach der Prüfung der aktuellen Delta-Sperrliste kann ein Verifizierer die gesamte aktuelle Sperrliste aus dieser sowie der Basis-Sperrliste rekonstruieren. Bemerkenswert ist, dass nach Erhalt der aktuellen Delta-Sperrliste $DCRL_i$ die vorher erhaltene Delta-Sperrliste $DCRL_{i-1}$ gelöscht werden kann, vorausgesetzt die Basis-Sperrliste der beiden Listen ist identisch.
Da die Einträge in $DCRL_i$ zumindest teilweise mit denen in $DCRL_{i-1}$ übereinstimmen, könnte eine jede Delta-Sperrliste eigentlich gleich als Basisliste für die unmittelbare Nachfolger-Liste betrachtet werden. Von dieser sparsamen Vorgehensweise wird allerdings in [ITU01] aus Rückwärtskompatibilitätsgründen abgeraten.

- **Kombinierte Strategien:**

Die bisher dargestellten Techniken zur Kostensenkung bei der Sperrlistenprüfung können auch kombiniert genutzt werden. Sinnvoll erscheint z.B. die Kombination von inkrementierten und dynamisch partitionierten Listen. Dabei kann in jedem Attributszertifikat eine *Redirect CRL* referenziert werden, die wiederum auf partitionierte Delta-Sperrlisten verweisen kann. Solche Delta-Sperrlisten referenzieren ihre jeweilige Basis-Sperrliste.

3.1.2 Sperrbäume

Eine der ersten Anwendungen der in Kap. 2.2.3 behandelten Authentifizierungsbäume für das Zertifikatsmanagement waren so genannte Sperrbäume (engl. *Certification Revocation Tree, CRT*) [Koc98]. Sperrbäume sind eine vielversprechende Alternative zu den oben vorgestellten Sperrlistenvarianten, was die Senkung der Kosten angeht. Das Funktionsprinzip von Sperrbäumen ist einfach. Eine Stelle, welche Sperrdaten zu eigenen oder von anderen ausgestellten Zertifikaten veröffentlicht, erzeugt und pflegt einen Authentifizierungsbaum, der Sperrdaten signiert. Um den Sperrstatus eines Zertifikats zu ermitteln, muss ein Authentifizierungspfad generiert und geprüft werden.

Dem Sperrbaum wurde in [Koc98] ein binärer Baum zugrunde gelegt (siehe Abb. 3-3). Die Blätter des Baums enthalten jeweils einen Hashwert eines Sperrdatums vom Typ *(Authority_Name, (Revoked_ID, Next_Revoked_ID))*.
Revoked_ID und *Next_Revoked_ID* bezeichnen jeweils bereits gesperrte Zertifikate, welche von der durch *Authority_Name* angegebenen Stelle stammen. Als eindeutiger Bezeichner kann die Seriennummer (*SerialNumber)* der Zertifikate verwendet werden. Solche Sperrdaten haben folgende Bedeutung: Ein von Zertifizierungsstelle *A* ausgestelltes Zertifikat Z_{Aid} mit der Seriennummer *id* gilt als gesperrt, wenn es das Sperrdatum (*A, Revoked_ID=id≤Next_Revoked_ID*) gibt.
Gibt es kein solches Blatt, ist Z_{Aid} gültig. Dies impliziert, dass alle Zertifikate mit Seriennummer v_i für welche *Revoked_ID*$<v_i<$*Next_Revoked_ID* gilt, gültig sind. Um ein Zertifikat zu sperren, muss ein neues Sperrdatum und folglich ein neues Blatt in den Baum eingefügt werden. Zusätzlich müssen die linken und rechten Nachbarn dieses neuen Blatts angepasst werden, wenn es diese gibt.

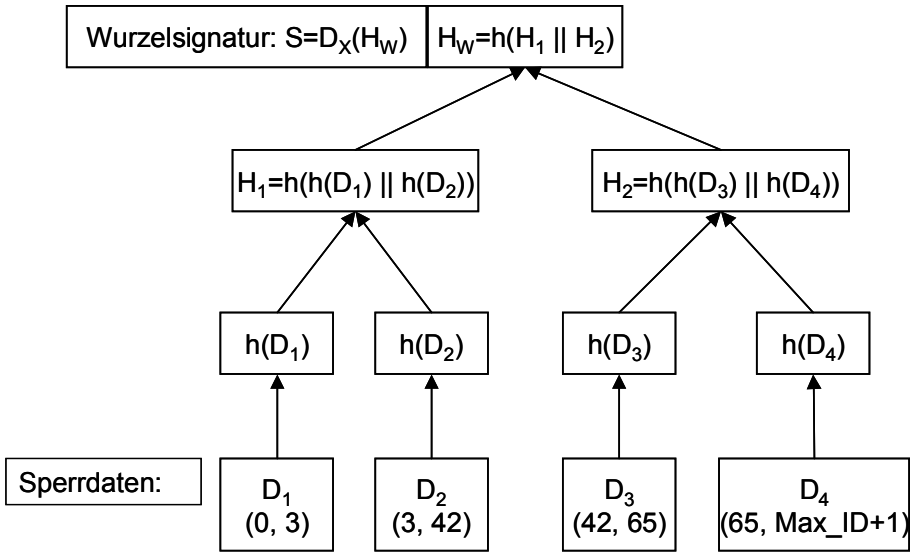


Abb. 3-3 Sperrbaum mit Sperrdaten einer Zertifizierungsstelle

In Abb. 3-3 wird ein Sperrbaum gezeigt, der Sperrdaten einer einzigen Zertifizierungsstelle beinhaltet. Die Sperrdaten D_1, D_2, D_3, und D_4 enthalten deshalb keinen Ausstellernamen und haben das Format *(Revoked_ID, Next_Revoked_ID*). Wie man sieht, wurden hier die Zertifikate mit *id={0, 3, 42, 65}* gesperrt.
Um den Status eines gesperrten Zertifikats mit *id=x* zu bestimmen, wird das Sperrdatum D_i mit *Revoked_ID=x≤Next_Revoked_ID* sowie der zugehörige Authentifizierungspfad $AP(D_i)$ benötigt.
Beispielsweise um das (gesperrte) Zertifikat mit *id=3* zu prüfen, wird D_2 und $AP(D_2)=\{D_X(H_W) \,||\, h(D_1) \,||\, H_2=h(h(D_3)||\, h(D_4))\,\}$ benötigt. Die Prüfung von $AP(D_2)$ erfolgt wie das schon am diesbezüglich identischen Beispiel in Kap. 2.2.3 gezeigt wurde.
Falls $AP(D_2)$ ein gültiger Authentifizierungspfad ist, wird geprüft, ob für das erhaltene Datum D_2 *Revoked_ID=x≤Next_Revoked_ID* oder *Revoked_ID*$<x<$*Next_Revoked_ID* gilt. Im ersten Fall (wie in diesem Beispiel) wird das Zertifikat als gesperrt und im zweiten als immer noch gültig betrachtet. Wurde ein Zertifikat hingegen mit *id=x* noch nicht gesperrt, bekommt ein Verifizierer das passende Sperrdatum D_i mit *Revoked_ID*$<x<$*Next_Revoked_ID* sowie den Authentifizierungspfad $AP(D_i)$. Beispielsweise erhält er zum gültigen Zertifikat mit *id=40* wieder D_2 und $AP(D_2)$.

Die Kosten dieser Konstruktion sind wie folgt zu beziffern:

- **Kommunikationskosten:** Die Übertragungskosten eines Authentifizierungspfads entsprechen $O(\log_2 r)$. Dabei steht r für die Anzahl der aktuell gesperrten Zertifikate (Blätter des Sperrbaums). Da Authentifizierungspfade häufig wiederverwendbar sind, ist ihre sowohl datenbank- als auch verifiziererseitige Zwischenspeicherung bis zur nächsten Aktualisierung des Sperrbaums sinnvoll. Dadurch können überflüssige Downloads und Signaturprüfungen vermieden werden. (Beispielsweise kann der oben behandelte $AP(D_2)$ über den Sperrstatus von 40 Zertifikaten Auskunft geben).

- **Berechnungskosten:** Was die Rechenkosten betrifft, müssen durchschnittlich $O(\log_2 r)$ Hashwertberechnungen durchgeführt werden, um einen Authentifizierungspfad und damit ein Zertifikat zu verifizieren. Hinzu kommen die Kosten der Signaturprüfung an der errechneten Wurzel. Hier ist also keine Beschleunigung gegenüber der Prüfung einer sortierten Sperrliste zu erkennen.

Insgesamt kann festgestellt werden, dass Sperrbäume eine Reduzierung der Kommunikationskosten gegenüber einfachen Sperrlisten ermöglichen, wobei die Berechnungskosten näherungsweise gleich ausfallen. Ähnlich gute Kommunikationskosten könnten aber beispielsweise auch mit in Kap. 3.1.1 vorgestellten partitionierten Sperrlisten erreicht werden, falls die Länge l der Sperrlisten-Partitionen entsprechend kurz, nämlich $l \leq \log_2 r$ gewählt wird.
Bei der Verwaltung eines Sperrbaums erscheint es nachteilig, dass das Einfügen (Löschen) eines Blatts zu einer Neuberechnung vieler (evtl. aller) Hashwerte innerhalb der Knoten des Baums führt. Ein Verbesserungsvorschlag hierfür wurde in [NN00] veröffentlicht: Durch die Nutzung von mehrfach verzweigten Bäumen fällt die durchschnittliche Anzahl der notwendigen Hashwertberechnungen niedriger aus.

3.1.3 Certificate Revocation System

In [Mic96] wurde das patentgeschützte so genannte *Certificate Revocation System* (CRS) veröffentlicht. Die Grundidee besteht darin, den Status sowohl bereits gesperrter als auch aktuell gültiger Zertifikate eindeutig prüfen zu können. CRS funktioniert folgendermaßen:
Beim Ausstellen eines Zertifikats Z werden diesem zwei ganze Zahlen Y (für *„Yes“*) und N (für *„No“*) zugeordnet. N und Y werden beide Bestandteil von Z. (Sie könnten z.B. als neuer Eintrag des Felds *Extensions* in X.509-Attributszertifikate eingefügt werden.) Um Y zu berechnen, wählt die jeweilige Zertifizierungsstelle eine geheime Zufallszahl Y_0 und berechnet $Y=Y_t=h_t(Y_0)=h(h(\ldots h(Y_0)))$, wobei h eine kollisionsfreie Hashfunktion sein muss, und t die diskrete Gültigkeitsperiode (beispielsweise gemessen in Tageseinheiten) von Z angibt. Die Zahl N wird durch $N=h(N_0)$ berechnet, wobei N_0 eine, bis zur etwaigen Sperrung von Z geheimzuhaltende, Zufallszahl sein muss. Durch die Zufallszahlen soll erreicht werden, dass die Werte Y und N für jedes Zertifikat unterschiedlich sind.
Jede Zertifizierungsstelle pflegt eine signierte Liste L, die periodisch (bspw. täglich) an eine den Verifizierern zugängliche Datenbank übermittelt wird. Falls ein Zertifikat Z_x mit dem eindeutigen Bezeichner (z.B. *SerialNumber*) x gesperrt wird, enthält L an der Stelle x die passende bis dahin geheime Zufallszahl N_0 sowie Informationen, wie z.B. den Sperrgrund.

Ein Verifizierer, der auf seine Anfrage nach dem Status von Z_x als Antwort N_0 erhält, testet ob $N=h(N_0)$. Ist das der Fall, wird Z_x als gesperrt erkannt, da die geheime N_0 nur von der Zertifizierungsstelle kommen kann (unter der Annahme, dass die Hashfunktion sicher ist).
Wurde hingegen ein Zertifikat Z_z mit $id=z$ seit dem letzten (i-ten) Update der Liste L nicht gesperrt, so enthält L an der Stelle z den Wert $V=Y_{t-i}=h_{t-i}(Y_0)$. So einen Wert muss also die Zertifizierungsstelle für jedes noch gültige Zertifikat zu jedem Aktualisierungszeitpunkt (z.B. täglich) durch eine (t-i)-fache iterierte Verwendung von h berechnen. Da nur diese Stelle die jeweilige geheime Zahl Y_0 kennt, kann nur sie diese Berechnung ausführen.
Ein Verifizierer, der einen solchen Gültigkeitseintrag V zu einem Zertifikat prüft, welches Y enthält, muss eine i-fache Iteration durchführen und $Y'=Y_t=h_i(V)$ berechnen. Ist $Y'=Y$, so ist das Zertifikat aktuell gültig, andernfalls erfolgte eine Manipulation von V.
CRS zeichnet sich vor allem durch niedrige durchschnittliche Kosten aus:

- **Kommunikationskosten:** Zur Verifizierung eines Zertifikats Z werden $O(1)$ Daten benötigt: Die Datenbank sendet nämlich zu jeder Statusanfrage lediglich eine Zahl N_0 (Z gesperrt) oder V (Z gültig). Im ersten Fall kommen noch die Angaben zu der Sperrung dazu.

- **Berechnungskosten:** Handelt es sich um die Prüfung eines gültigen Zertifikats, so entsprechen die Kosten $O(t)$, wobei t Hashwertberechnungen notwendig sind. Wurde ein gefragtes Zertifikat hingegen gesperrt, so liegen die Kosten im Bereich $O(1)$. Ein weiterer Vorteil ist, dass die Datenbank keinerlei Berechnungen durchführen muss, um ihre Antworten zu generieren. Sie kommt bei der Suche nach Informationen zu einem bestimmten Zertifikat mit einer einfachen Binärsuche in L in $O(\log_2 n)$ Schritten aus. Dabei bezeichnet n die maximal mögliche Anzahl der Zertifikate im System.

Neben den erwähnten Vorteilen erscheint nachteilig, dass es nicht überprüfbar ist, ob die Datenbank tatsächlich die dort abgelegten zum gefragten Zertifikat passenden Daten zurücksendet. Ihre (möglicherweise falsche) Antworten sind mangels einer Signatur der jeweiligen Zertifizierungsstelle nicht eindeutig auf diese zurückzuführen. Des Weiteren ist die Aktualisierung der Liste L aufwändig. Dabei müssen zu jedem bis dato ausgestellten Zertifikat t-i Hashwertberechnungen durchgeführt werden. Die Kosten hierfür sowie für die Übermittlung der Liste an die Datenbank entsprechen $O(n)$. Außerdem muss das System nach der t-ten Aktualisierung der Datenbank neu initialisiert werden. Das heißt, es müssen neue Y_0, Y und evtl. auch N_0, N Werte für jedes Zertifikat generiert werden. Danach müssen die erneuerten Zertifikate signiert und verteilt werden.
In [Mic97] wurde eine Lösung dieser Probleme diskutiert. Dabei wird ein Authentifizierungsbaum verwendet. Zu jedem der n möglichen Zertifikatsbezeichner im System wird jeweils ein 2 Bit langer Wert S_{id} zugeordnet. Dieser zeigt an, ob das Zertifikat Z_{id} mit der Kennung id ausgestellt wurde (1. Bit) und ob es gültig ist (2. Bit). Der Authentifizierungsbaum signiert Datensätze, welche jeweils k (z.B. $k=64$) solche Bitpaare enthalten. Die n/k Blätter des Baums enthalten dementsprechend Hashwerte, wie z.B. $h(S_1, S_2, \ldots S_k)$. Die Sperrkosten können somit auf etwa $O(\log_2 (n/k))$ geschätzt werden [NN00], da hier das Einfügen eines entsprechend geänderten Blattes in den Baum benötigt wird.

Um das Zertifikat Z_{id} zu prüfen, bekommt nun der Verifizierer den signierten Authentifizierungspfad zum passenden Datensatz, welcher S_{id} enthält, sowie die anderen mit diesem zusammen gehashten *k-1* Werte. Die Überprüfung von S_{id} erfolgt, indem der Authentifizierungspfad geprüft wird. Da nun die Datenbank Authentifizierungspfade dynamisch generieren muss, entstehen hier Kosten im Bereich $O(log_2(n/k))$. Ebenfalls müssen nun etwa $O(log_2 (n/k))$ statt $O(1)$ Hashwertberechnungen ausgeführt werden, um ein Zertifikat zu prüfen. Hinzu kommen natürlich die Kosten der Prüfung der Wurzelsignatur.
In [ALO98] wurde ebenfalls für die Nutzung eines Authentifizierungsbaums plädiert, mit ähnlichen kostenbezogenen Konsequenzen wie der dargestellte Ansatz in [Mic97].

3.2 Einsatz von Online-Statusdiensten

Eine denkbare Alternative zur Bereitstellung von Sperrdaten für die Verifizierer ist die (teilweise) Auslagerung der Zertifikatsprüfung (siehe Kap. 2.3.3) an hierfür spezialisierte Instanzen. Durch die Einführung eines so genannten Online Zertifikatsstatus-Dienstes (engl. *Online Certificate Status Service*) können Verifizierer in einem System entlastet werden: Die zur Zertifikatsprüfung benötigten Daten (beispielsweise Sperrlisten) müssen nämlich nicht regelmäßig heruntergeladen und geprüft werden. Das allgemeine Funktionsprinzip eines solchen Dienstes wird in Abb. 3-4 schematisch dargestellt. (In dieser Abbildung wurden auch die weiter unten diskutierten Techniken eingeordnet.)

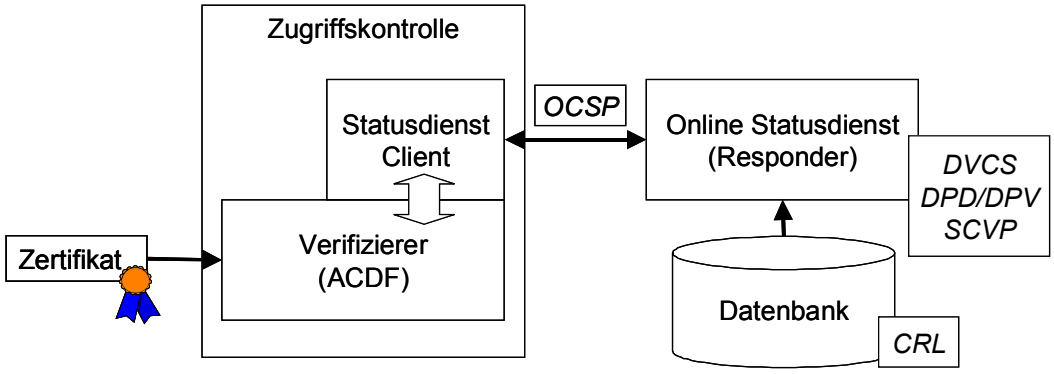


Abb. 3-4 Funktionsprinzip der Statusdienste

Angenommen einem Verifizierer liegt ein zu prüfendes Attributszertifikat vor. Wie man in Abb. 3-4 sieht, handelt es sich hier um einen Verifizierer, der nun um einen Statusdienst-Client ergänzt wurde. Dieser Client wickelt im Wesentlichen die Kommunikation mit dem Statusdienst beziehungsweise mit dessen häufig als „Responder“ bezeichneter Komponente ab. Der Statusdienst kann hierbei auf Anfrage des Verifizierers bestimmte Teilschritte innerhalb des Zertifikatsprüfungs-Prozesses (evtl. alle) übernehmen. Hierfür werden in einem Hintergrundsystem abgelegte Daten, wie z.B. Sperrlisten verwendet. Diese Daten müssen grundsätzlich von vertrauenswürdigen Stellen stammen. Nach den notwendigen Bearbeitungsschritten wird die Anfrage beantwortet. Nach der Auswertung der Antwort wird entschieden, ob das vorliegende Attributszertifikat akzeptiert wird.
Nachfolgend werden existierende und zum Teil bereits standardisierte Verfahren vorgestellt, deren Kombination die Realisierung eines Statusdienstes zur Unterstützung der Zugriffskontrolle ermöglicht.

3.2.1 Online Certificate Status Protocol (OCSP)

OCSP ist ein standardisiertes Protokoll (aktuell ist Version 2) zur Abwicklung der notwendigen Kommunikation während einer ausgelagerten Zertifikatsprüfung [MAM+99] [MMP02]. OCSP ermöglicht es einem OCSP-Client, die Gültigkeit von gleichzeitig mehreren Zertifikaten bei einem so genannten OCSP-Responder abzufragen. OCSP-Nachrichten werden standardgemäß über HTTP transportiert. Hierbei wird eine OCSP-Request-Nachricht, welche eine Liste eindeutiger Zertifikats-Identifikatoren (*certID*) enthält, an den OCSP-Responder gesendet. Zu jedem gefragten Zertifikat enthält die OCSP-Response-Nachricht jeweils eine der Antworten *„good“, „revoked“* oder *„unknown“*. In den Spezifikationen [MAM+99] und [MMP02] wird davon ausgegangen, dass die Antworten eines OCSP-Responders anhand von Sperrlisten generiert werden. Die Antwort *„revoked“* (*„good“*) bedeutet daher, dass das gefragte Zertifikat in der Sperrliste (nicht) auftaucht. Die Antwort *„unknown“* wird in Fehlerfällen generiert, beispielsweise wenn keine zum Zertifikat passende Sperrliste vorliegt. Es wird leider nicht behandelt, wie ein Verifizierer auf diese Meldung reagieren soll. Es kann deshalb vorkommen, dass Zertifikate ungeprüft akzeptiert werden.
Bezüglich der Korrektheit seiner Status-Antworten muss ein OCSP-Responder als vertrauenswürdig eingestuft werden. Dies steht jedoch nicht im Einklang mit dem X.509-Standard [X.509], in welchem Verifizierern vorgeschrieben wird, sich nur auf Informationen zu verlassen, die von den vertrauten Zertifizierungsstellen (SOAs) stammen (siehe auch Kap. 2.3.3). Eine potenzielle Gefahrenquelle stellt die Möglichkeit zur Fälschung der gesendeten OCSP-Nachrichten dar. Ein Angreifer mit Zugriff auf den benutzten Kommunikationskanal (z.B. Internet) kann OCSP-Nachrichten unbemerkt verändern. Dadurch können gültige (ungültige) Attributszertifikate fälschlicherweise für ungültig (gültig) erklärt werden. Dies führt wiederum zu gefälschten Zugriffskontrollentscheidungen im System. Deswegen müssen OCSP-Nachrichten geschützt werden, wofür in [MAM+99] und [MMP02] die Verwendung von digitalen Signaturen vorgesehen wird. Dabei ist das Signieren von OCSP-Request-Nachrichten optional. Dieser signaturbasierte Ansatz zur Sicherung der Kommunikation schränkt die Nutzungsmöglichkeiten von OCSP für eine performante Zugriffskontrolle ein. Die grobe Kostenbilanz (Wartezeiten) der Gültigkeitsprüfung eines einzigen Attributszertifikats mit *certID* sieht nämlich wie folgt aus:

- **Verifizierer**: Erstellung einer (optional) signierten OCSP-Request-Nachricht, Versenden dieser Nachricht an den OCSP-Responder, Warten auf dessen Antwort, Signaturprüfung und Auswertung der erhaltenen OCSP-Response-Nachricht.

- **OCSP-Responder:** (Optionale) Signaturprüfung und Auswertung der erhaltenen OCSP-Request-Nachricht, Suche nach *certID* in der passenden Sperrliste (falls diese vorhanden ist), Erstellen und Signieren einer OCSP-Response-Nachricht, Versenden der Nachricht an den Verifizierer.

Es müssen also insgesamt mindestens zwei und maximal vier signaturbezogene Berechnungen ausgeführt werden, um die aktuelle Gültigkeit eines einzigen Attributszertifikats zu prüfen. Im schlimmsten Fall wird nach solch einem Aufwand festgestellt, dass der Status des Zertifikats unbekannt (*unknown*) ist. Die pro Zertifikat anfallenden Kosten können jedoch sinken, da in einer OCSP-Request-Nachricht gleichzeitig nach dem Status mehrerer Attributszertifikate gefragt werden kann.

3.2.2 Data Validation and Certification Server (DVCS)

DVCS stellt einen Online-Statusdienst für signierte Daten beliebiger Art, d.h. prinzipiell auch für Attributszertifikate dar [ASZ+01]. Sein Funktionsprinzip ist Folgendes: Eine Serverkomponente welche als vertrauenswürdig eingestuft werden muss, stellt auf Anfragen des Clients signierte so genannte Validierungszertifikate (engl. *Data Validation Certificate,* DVC) aus. Ein DVC ist eine Bestätigung, dass ein gefragtes signiertes Dokument bestimmte Sicherheitsmerkmale vorweist. Eines der in [ASZ+01] vorgesehenen Merkmale ist, ob die Signatur am Dokument mit einem zu einem bestimmten Zeitpunkt gültigen Schlüssel erzeugt wurde. Das bedeutet, eine DVCS-Komponente könnte im Prinzip die Signaturprüfung innerhalb der in Kap. 2.3.3 behandelten Zertifikatsprüfung übernehmen (siehe auch Abb. 2-12). Das prinzipielle Problem damit ist, dass zur Signaturprüfung eines Attributszertifikats nun ein anderes Zertifikat, nämlich das passende Validierungszertifikat (DVC), angefordert, ausgestellt und überprüft werden muss. Zumindest was die Kosten angeht, ist dies keine echte Unterstützung für die Zugriffskontrolle.
Eine sinnvollere Anwendung für DVCS kann im Bereich der Nachweisführung gefunden werden. Hierbei können bereits abgelaufene archivierte Attributszertifikate auf ihre Gültigkeit zu einem früheren Zeitpunkt geprüft werden. Dadurch kann beispielsweise in einem Streitfall entschieden werden, ob ein Benutzer zu diesem vergangenen Zeitpunkt bestimmte Berechtigungen im System besaß.

3.2.3 Simple Certificate Validation Protocol (SCVP)

In der Spezifikation von SCVP [MHF03], welcher die Anforderungssammlung [PH02] zugrunde liegt, werden folgende Aspekte der Zertifikatsprüfung betrachtet:

- **Suche nach Zertifikatsketten:** Die Funktionalität mit der Bezeichnung *Delegated Path Discovery* (DPD) dient der Ermittlung so genannter Zertifikatsketten (engl. *Certificate Chain*) zu einem bestimmten Schlüsselzertifikat. Solche Ketten entstehen durch die so genannte Kreuzzertifizierung (engl. *Cross Certification*). Dabei zertifizieren sich Zertifizierungsstellen gegenseitig und ermöglichen damit ihren Kunden das Finden einer gemeinsamen (indirekt) vertrauten Stelle [AL99] [ITU01] [HP01]. Bei der Suche nach möglichst allen Zertifikatsketten, deren Mitglied das gefragte Zertifikat ist, kann sich ein DPD-Dienst auf externe Zertifikatsdatenbanken stützen. Die Überprüfung, ob eine von einem DPD-Dienst gelieferte Zertifikatskette auch tatsächlich akzeptabel ist, erfolgt seitens der abgesicherten Anwendung (Verifizierer). Eine mögliche Gefahr beim Einsatz eines solchen Dienstes ist, dass dieser bestimmte (evtl. alle) gefundene Zertifikatsketten zurückhalten und damit beispielsweise die Etablierung sicherer Kommunikationsbeziehungen verhindern kann.

- **Gültigkeitsprüfung:** Neben dieser Suchfunktionalität kann ein SCVP-basierter Dienst auch die Gültigkeitsprüfung von Zertifikaten unterstützen und somit die Funktionalität eines in Kap. 3.2.1 behandelten OCSP-Responders erbringen. Dabei gelten die oben beim OCSP behandelten Einschränkungen.

- **Übernahme der kompletten Zertifikatsprüfung:** Das Konzept der so genannten *Delegated Path Validation* (DPV) sieht die ausgelagerte Abwicklung des kompletten Prozesses der Zertifikatsprüfung (siehe Abb. 2-12) vor.

Eine DPV-Anwendung, also in unserem Fall ein Verifizierer, leitet zuvor erhaltene Zertifikate ungeprüft an den Dienst weiter. Daraufhin erhält die DPV-Anwendung Auskunft darüber, ob diese Zertifikate akzeptiert oder angelehnt werden sollen. Durch diese Vorgehensweise bekommt der DPV-Dienst eine große Macht über die Geschehnisse innerhalb der Zugriffskontrolle. Diesen Problemen wird in der Spezifikation [MHF03] versucht entgegenzuwirken.
Hierbei sollen DPV-Anwendungen die Richtlinien abfragen können, aufgrund welcher die jeweiligen Entscheidungen des DPV-Dienstes gefällt werden. In diesem Fall muss ein DPV-Client die ihm akzeptablen Richtlinien nicht nur im Voraus kennen sondern auch *glauben*, dass der DPV-Dienst tatsächlich gemäß dieser arbeitet.

Die in diesem Kapitel vorgestellten Mechanismen für Online Zertifikatsstatus-Dienste können - vor allem im Vergleich zu einfachen Sperrlisten – zur Reduktion der Kommunikationskosten innerhalb der zertifikatsbasierten Zugriffskontrolle führen.
Diesem Gewinn stehen aber in vielen Fällen erhöhte Berechnungskosten seitens der abgesicherten Systeme gegenüber. Des Weiteren können durch die teilweise oder komplette Auslagerung der Zertifikatsprüfung an ein Online-System verschiedene vertrauens- bzw. sicherheitstechnische Probleme entstehen, die den möglichen Einsatzbereich dieser Technik auf weniger kritische Anwendungen einschränken.

3.3 Gültigkeitsprüfung anhand von Gültigkeitsinformationen

Die bisher behandelten Techniken zur Zertifikatsprüfung ermöglichen eine indirekte Beantwortung der Frage, ob ein früher ausgestelltes Attributszertifikat immer noch als gültig angenommen werden kann. Die dabei verwendeten Daten, wie z.B. Sperrlisten oder Sperrbäume enthalten nämlich Informationen zu bereits ungültig gewordenen Zertifikaten. Aus diesem Grund werden diese Datenstrukturen häufig als *„Blacklists“* bezeichnet (obwohl es sich hier nicht zwangsläufig um Listen handelt).

3.3.1 „Whitelist“-Techniken

Eine alternative Strategie zur Unterstützung der Gültigkeitsprüfung ist die Bereitstellung von Daten, so genannten *„Whitelists“,* die Informationen zu aktuell gültigen anstatt bereits ungültig gewordenen Attributszertifikaten tragen (dieser Strategie wird auch in CRS teilweise gefolgt, siehe Kap. 3.1.3). Nach Erhalt solcher Gültigkeitsdaten kann die Gültigkeitsfrage etwas expliziter beantwortet werden: Ein in der Vergangenheit ausgestelltes Zertifikat gilt nämlich genau dann als gültig, wenn zu diesem aktuelle und authentische Gültigkeitsinformationen vorhanden sind.
Die in Kap. 3.1 vorgestellten verschiedenen Sperrtechniken können auf die Bedürfnisse dieser Vorgehensweise angepasst werden. Somit sind signierte „Positivlisten“ statt Sperrlisten denkbar [Woh01] [BGG+99], die sich (analog den Sperrlisten) statisch oder dynamisch partitionieren lassen. Ein Zertifikat gilt demnach genau dann als gültig, wenn dessen eindeutiger Identifikator, wie z.B. *SerialNumber* in der signierten Positivliste enthalten ist.
Analog können „Gültigkeitsbäume“ statt Sperrbäume konstruiert werden, deren Blätter Kennungen von gültigen statt frühzeitig widerrufenen Zertifikaten enthalten (dies ist nicht zu verwechseln mit den in Kap. 4.1 behandelten *Certification Verification Trees*).
Die Bereitstellung von Gültigkeits- anstatt von Sperrdaten allein kann die (systemweiten) Probleme mit der frühzeitigen Sperrung der Attributszertifikate natürlich nicht lösen. Sperrungen werden ja durch Änderungen der Autorisierungsdaten ausgelöst, die weiterhin zu erwarten sind.

Sollte ein Attributszertifikat gesperrt werden, so muss der passende Eintrag nun aus der jeweiligen Gültigkeits-Datenstruktur entfernt werden. Auch hier gilt, dass die Verifizierer über eventuelle Status-Änderungen möglichst schnell und effizient informiert werden sollten. Deshalb müssen neben den Attributszertifikaten (Nutzdaten) nun Gültigkeitsdaten anstatt Sperrdaten verwaltet, regelmäßig aktualisiert, zur Verfügung gestellt und von den Verifizierern ausgewertet werden (siehe auch die diesbezügliche Diskussion in Kap. 3.1).
Ein grundsätzlicher Nachteil von Whitelist-Ansätzen ist die durchschnittlich erwartete größere Datenmenge. Schenkt man beispielsweise der bereits erwähnten Studie [NIST94] Glauben und werden 90% der ausgestellten Zertifikate *nicht* frühzeitig gesperrt, so muss diese 90% der Zertifikate in den aktuellen Gültigkeitsdaten stets reflektiert werden. Das heißt, beim Herunterladen der Gültigkeitsdaten muss grundsätzlich mit einem (etwa neunfach) erhöhten Kommunikationsaufwand gerechnet werden.

3.3.2 Technik der „Suicide Bureaus“

In [Riv98] wurde die Frage aufgeworfen, ob es zwingend notwendig sei, die frühzeitige Sperrung von Zertifikaten mit periodisch veröffentlichten Daten (betrachtet wurden Sperrlisten) zu unterstützen. Es wurde dabei für die Etablierung eines Netzwerks bestehend aus Online-Zertifizierungsstellen, die so genannten „*Suicide Bureaus*“ (SB), plädiert. Eine solche Stelle stellt (wie gewohnt) Zertifikate mit eingeschränkter Gültigkeitsperiode aus. Während der Prüfung eines Zertifikats kann sich ein Verifizierer entscheiden, ob ihm das Zertifikat „aktuell genug“ ist. Bei Bedarf kann er den zugreifenden Benutzer auffordern, sich an eines der SBs zu wenden und eine aktualisierte Version des vorgelegten Zertifikats mitzubringen. Ein SB kann dabei ein (evtl. schon abgelaufenes) Zertifikat verlängern und es dem Benutzer aushändigen. Ebenfalls kann ein SB auf die Anfrage des Benutzers („suicide“) ein Zertifikat (frühzeitig) für ungültig erklären, d.h. sperren. In diesem Fall legen die Benutzer die signierte „Selbstmordnachricht“ dem Verifizierer vor, sodass dieser sich nicht um die Beschaffung und Auswertung von (zusätzlichen) Sperrinformationen kümmern muss.
Durch diese Vorgehensweise konnte also nachgewiesen werden, dass die Abschaffung von periodisch ausgestellten den Verifizierern zugänglich gemachten Sperrdaten prinzipiell möglich ist. Der Beitrag hat eine rege Diskussion in der Fachwelt ausgelöst und viele zum Weiterdenken inspiriert [MR00] [WLM00] [GGM00].
Da bei dieser Vorgehensweise die Aktualität eines früher ausgestellten Attributszertifikats nur bei Bedarf durch zusätzliche Daten bestätigt und geprüft wird, können vor allem die kommunikationsbezogenen Kosten der Zertifikatsprüfung reduziert werden. Ein Verifizierer, der ein vorliegendes Attributszertifikat für „frisch genug“ hält, muss nämlich keine zusätzliche Gültigkeitsprüfung anhand von evtl. großen Datenmengen durchführen.
Die Frage ist nur, wann ein Attributszertifikat noch als „frisch genug“ eingestuft werden kann. Dies hängt theoretisch nur von den Sicherheitsanforderungen des jeweiligen abgesicherten Systems ab (z.B. „nicht älter als 3 Tage“ oder „nicht älter als 5 Sekunden“). Anwendungen, auf welche häufig viele Benutzer mit evtl. jeweils mehreren Attributszertifikaten zugreifen, werden jedoch ihre Erwartungen diesbezüglich zurückstellen müssen. Die nun online gegangenen Zertifizierungsstellen müssen nämlich unter Umständen sehr häufig viele neue Attributszertifikate oder Selbstmordnachrichten ausstellen und diese signieren, was leicht zu langen Wartezeiten führen kann.

4 Häufige Aktualisierung von Attributszertifikaten

Die in Kap.3 behandelten Verfahren unterstützen die Gültigkeitsprüfung von Zertifikaten. Diese Verfahren basieren auf der Annahme, dass die Zertifikate zu einem gegebenen Zeitpunkt möglicherweise Daten, wie z.B. Zugriffsrechte, beinhalten, die den zum Zeitpunkt der Zertifizierung herrschenden Bedingungen nicht mehr entsprechen. Deshalb wird anhand von aktuelleren Prüfdaten wie z.B. Sperrlisten oder Sperrbäumen entschieden, ob ein vorliegendes Zertifikat immer noch als gültig akzeptiert werden kann. Hierfür müssen diese Prüfdaten mit der jeweilig gewünschten Häufigkeit aktualisiert und unter den Verifizierern verteilt werden. Da solche Daten ebenso schützenswert sind wie die Zertifikate selbst, müssen sie durch vertrauenswürdige Stellen signiert werden. Bei der Überprüfung und Auswertung dieser Daten während der Zugriffskontrolle entstehen folglich je nach Konstruktion unterschiedlich lange zusätzliche Wartezeiten.

In diesem Kapitel wird eine andere, effizientere Vorgehensweise entworfen. Eine eigentlich naheliegende Idee zur Aufwandsreduzierung der Zertifikatsprüfung ist die häufige Aktualisierung der Attributszertifikate. Im Grunde genommen bedeutet dies eine (evtl. einheitliche) Verkürzung der Gültigkeitsperiode der Zertifikate in einem System. Nach Erreichen des vorgesehenen Ablaufsdatums kann entschieden werden, ob ein Zertifikat „verlängert“, d.h. aktualisiert und neu signiert wird. Werden in einem System sämtliche Attributszertifikate häufig genug (beispielsweise im Minutentakt) aktualisiert, so kann bei deren Prüfung auf die Nutzung von zusätzlichen Daten verzichtet werden. Bei einer hinreichend kurzen Gültigkeitsperiode t ist nämlich das Risiko gering, dass ein inzwischen ungültig gewordenes Attributszertifikat innerhalb der bis zur nächsten Aktualisierung verbleibenden Zeitspanne (im Durchschnitt $t/2$) fälschlicherweise verwendet wird. Diese Vorgehensweise hat folgende erkennbare Auswirkungen auf den Betrieb eines Systems:

- **Beschleunigung der Zugriffskontrolle:** Der Prozess der Zugriffskontrolle wird infolge der nun vereinfachten Gültigkeitsprüfung (siehe Abb. 4-1) beschleunigt. Während der Gültigkeitsprüfung muss nämlich nur entschieden werden, ob sich ein Attributszertifikat noch innerhalb der vorgesehenen (kurzen) Gültigkeitsdauer befindet.

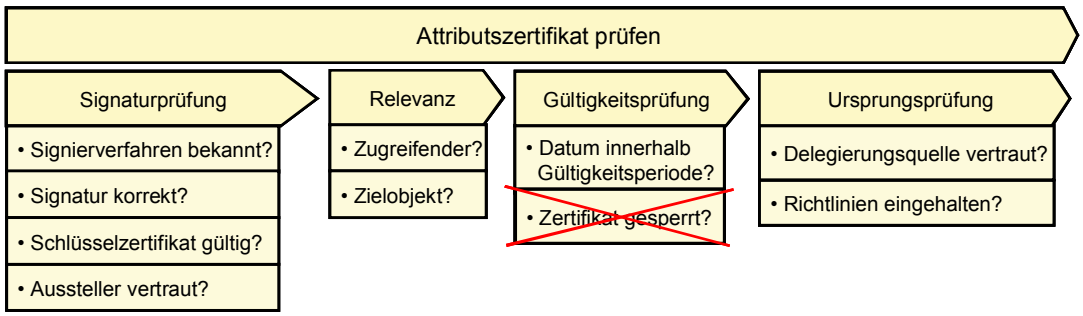


Abb. 4-1: Durch häufige Aktualisierungen vereinfachte Zertifikatsprüfung

- **Reduzierung des Verwaltungsaufwands**: Da die Zertifikate nun genug Informationen zu deren sicheren Verifizierung enthalten, wird die Aufrechterhaltung der in Kap. 3 behandelten verschiedenen Hintergrundsysteme, wie z.B. einer Sperrdatenbank oder eines OCSP-Responders überflüssig.

- **Standardkonformität:** Der X.509-Standard [ITU01] unterstützt diese Strategie zumindest implizit. Es ist nämlich erlaubt, während der Zugriffskontrolle auf die Prüfung von Sperrdaten zu verzichten. Genau dies wird hier angestrebt, wobei keine zusätzlichen Risiken entstehen, wie das bei Attributszertifikaten mit einer langen Gültigkeitsperiode der Fall ist.

Eine Voraussetzung dieser Vorgehensweise ist, dass sämtliche *n* Attributszertifikate einer Zertifizierungsstelle auch tatsächlich mit der jeweilig gewünschten Häufigkeit aktualisiert sowie den Verifizierern zugänglich gemacht, d.h. verteilt werden können. Ob dies gelingt, hängt im Wesentlichen von den bei den Aktualisierungen entstehenden Kosten ab:

- **Berechnungskosten:** Im Zuge einer Aktualisierung des Zertifikatsbestands muss die neue Gültigkeitsperiode (*AttrCertValidityPeriod*) der jeweilig zu verlängernden Attributszertifikate festgesetzt werden. Anschließend müssen diese Zertifikate signiert werden. Die Berechnungskosten der Aktualisierung werden maßgeblich von der Anzahl *n* der jeweils notwendigen Signaturberechnungen beeinflusst, und liegen im Bereich *O(n)*. Die Zeit, die damit verbracht wird, setzt eine obere Schranke für die maximal erreichbare Häufigkeit der Aktualisierungen.
  Eigenen Messungen zufolge nimmt das Signieren eines 4 Kilobytes großen Attributszertifikats durchschnittlich 31 Millisekunden in Anspruch. Dieser Wert wurde anhand von 1000 unabhängigen Messungen ermittelt, die auf einem PC-System mit einem 1,8 GHz Prozessor durchgeführt wurden. Dabei wurde eine ANSI C-Implementierung (genauer die Bibliotheken der Microsoft Windows Cryptographic API) der beiden in Kap. 2.2.1 respektive Kap. 2.2.2 erwähnten Verfahren SHA-1 und RSA mit einer Schlüssellänge von 2048 Bit verwendet.

- **Kommunikationskosten:** Eine weitere Einschränkung der Möglichkeiten ist der ebenfalls im Bereich von *O(n)* liegende Aufwand bei der Verteilung der aktualisierten Attributszertifikate (siehe die Server-Pull- und Client-Push-Modelle in Kap. 2.1.3). Erreichen die Zertifikate die Zugriffskontrolle über eine Online-Zertifikatsdatenbank (Server-Pull), so müssen in der Datenbank stets die aktuellsten Zertifikate vorliegen. Die häufigen Aktualisierungen können das jeweilige Datenbanksystem stark belasten, so dass Anfragen der Verifizierer evtl. nicht mit einer adäquaten Geschwindigkeit bedient werden können. Die Situation wird noch etwas komplizierter, wenn Benutzer des abgesicherten Systems ihre Attributszertifikate in ihrer lokalen Umgebung aufbewahren und diese bei Bedarf an die Zugriffskontrolle schicken (Client-Pull). Hier kann nämlich nicht damit gerechnet werden, dass sämtliche entsprechenden Systeme stets erreichbar sind.

4.1 Certification Verification Tree

Eine in [GGM00] veröffentlichte Idee kann zur Reduktion der oben genannten Kosten beitragen. Ein so genannter *Certification Verification Tree* (CVT) ist eine weitere Anwendung für Merklesche Authentifizierungsbäume (siehe Kap. 2.2.3) [Mer90]. Hierbei werden die durch eine Zertifizierungsstelle ausgestellten Zertifikate mit Hilfe eines Authentifizierungsbaums signiert. Folglich kann die Signatur eines einzigen Zertifikats unter Zuhilfenahme des entsprechenden Authentifizierungspfads geprüft werden (siehe auch [AFS01]). Solche Authentifizierungspfade wurden in [GGM00] sinngemäß als Zertifizierungspfade (ZP) (engl. *Certification Path*) bezeichnet. Die in die Hashwertberechnung der Blätter einbezogenen und dadurch zertifizierten Daten sind die so genannten „*Cert-Statements*".

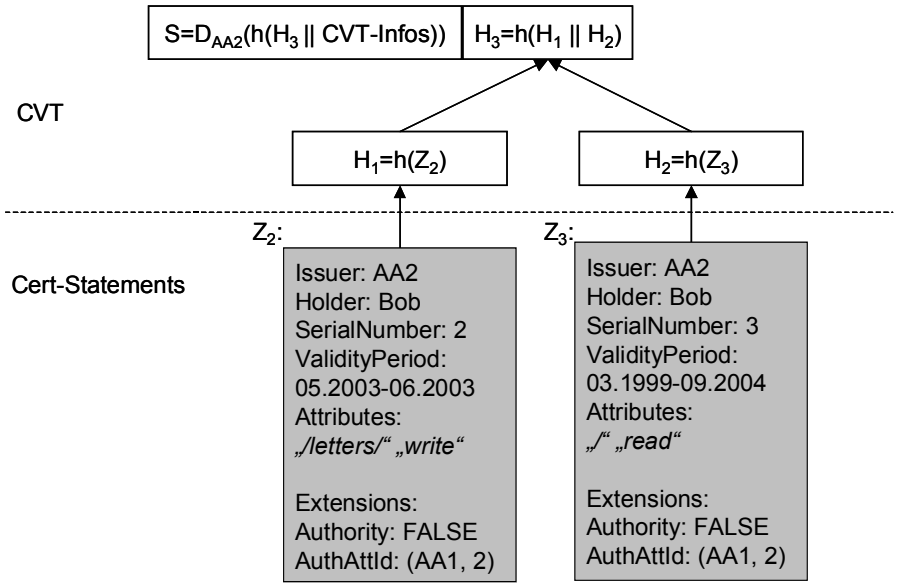


Abb. 4-2: Binärer Certification Verification Tree (CVT)

Ein auf dieser Grundlage konstruierter binärer CVT wird in Abb. 4-2 gezeigt. Wie man sieht, werden hier die bereits aus Abb. 2-9 bekannten Attributszertifikate Z_2 und Z_3 von *Bob* mit den Seriennummern (*SerialNumber*) *2* respektive *3* durch die Wurzelsignatur *S* des Baums erfasst. Die Wurzelsignatur $S=D_{AA2}(h(H_3 \,||\, CVT\text{-}Infos))$ wurde mit Hilfe des privaten Schlüssels der Zertifizierungsstelle (*Issuer*) namens *AA2* erzeugt.
Die in die Wurzelsignatur einfließenden, als *CVT-Infos* bezeichneten Daten können zur Überprüfung der Wurzelsignatur verwendet werden, indem hier beispielsweise der jeweilig verwendete Signaturalgorithmus angegeben wird (siehe noch Kap. 4.2).
Um zu prüfen, ob z.B. das vorliegende Zertifikat („Cert-Statement") Z_2 durch den Baum signiert wurde, wird der Zertifizierungspfad $ZP(Z_2)$ benötigt:

$$ZP(Z_2)= [\ S=D_{AA2}(\ h(H_3 \,||\, CVT\text{-}Infos\ )) \,||\, H_2 \,||\, CVT\text{-}Infos\ ]$$

Dieser beinhaltet genug Information, um die Wurzelsignatur und damit die Authentizität und Integrität von Z_2 zu überprüfen. Berechnet wird hierfür der Wurzelhashwert

$$H_W'=h(\ h(h(Z_2) \,||\, H_2) \,||\, CVT\text{-}Infos).$$

Anschließend muss (unter der Verwendung des passenden öffentlichen Schlüssels von *AA2*) der tatsächlich signierte Wurzelhashwert durch

$$H_W=E_{AA2}(S)$$

ermittelt werden. Ist

$$H_W'=H_W,$$

so ist Z_2 tatsächlich durch den Baum signiert worden. Andernfalls handelt es sich hier um ein durch den Baum nicht erfasstes (gefälschtes) Zertifikat.

4.2 Einsatz von CVTs für die zertifikatsbasierte Zugriffskontrolle

Durch die Verwendung von CVTs kann eine Beschleunigung der Zertifikatsaktualisierungen erreicht werden. Die insgesamt *n* durch eine Zertifizierungsstelle verwalteten Zertifikate können nämlich praktisch beliebig häufig neu signiert werden.
Hierfür muss bei jeder Aktualisierung des gesamten Zertifikatsbestands genau eine, d.h. *O(1)* Signaturberechnung erfolgen. Die Anzahl der für die Aktualisierung des Baums jeweilig benötigten Hashwertberechnungen kann nun eine Grenze der erreichbaren Aktualisierungshäufigkeit setzen. Angenommen, zwischenzeitlich werden k ($0<k\leq n$) Zertifikate geändert, so müssen im Schnitt *O(k)* Hashwerte berechnet werden, um den neuen Wurzelhashwert des Baums zu ermitteln.
Dabei gilt zu beachten, dass bekannte Hashfunktionen (siehe Kap. 2.2.1) etwa 10.000-fach schneller sind als die bei weitem komplizierteren Public-Key-Verfahren [Sch96]. Dies bestätigen auch eigene Messungen auf der weiter oben bereits erwähnten PC-Plattform.
Die Gesamtkosten zur Aktualisierung eines Zertifikatsbestands durch einen CVT fallen also auf jeden Fall niedriger aus, als wenn ein jedes zu verlängerndes Zertifikat einzeln signiert werden müsste. In diesem zweiten Fall sind nämlich *O(n)* Hashwertberechnungen und *O(n)* mit dem privaten Schlüssel der Zertifizierungsstelle getätigten Berechnungen notwendig.
Um die Möglichkeit zu häufigen Zertifikatsaktualisierungen für die Zugriffskontrolle basierend auf X.509-Attributszertifikaten auszunutzen, sind die folgenden Aspekte von Bedeutung:

- **Redundanzsenkung zertifizierter Daten:** Das in Kap. 2.2.4 behandelte X.509-Zertifikatsformat *AttributeCertificateInfo* beinhaltet die Felder *Version, Holder, Issuer, Signature, SerialNumber, AttrCertValidityPeriod, Attributes, IssuerUniqueID* und *Extensions* (siehe Abb. 2-5). Da nun mehrere Attributszertifikate durch einen CVT signiert werden können, erscheinen einige dieser Felder redundant:
  Die Felder *Version* (muss den Wert *2* haben), *Issuer*, *IssuerUniqueID* (beide bezeichnen den Signierenden eines Zertifikats) und *Signature* (bezeichnet die Signiermethode) fallen nämlich für sämtliche durch einen CVT erfasste, d.h. aktuell gültige, Zertifikate identisch aus. Folglich können diese Daten Bestandteile der oben als *CVT-Infos* bezeichneten, im Wurzelknoten des CVT abgelegten Daten werden, und müssen nicht in den einzelnen durch den Baum signierten Zertifikaten enthalten sein.
  In Abb. 4-3 werden die auf diese Weise zwischen *AttributeCertificateInfo* und *CVT-Infos* aufgeteilten Daten dargestellt (vgl. Abb. 2-5). Durch diese Aufteilung der Daten kann der systemweite Speicheraufwand sowie der Berechnungsaufwand während der Aktualisierungen reduziert werden.

Sie hat gleichzeitig keinen negativen Einfluss auf die Sicherheit der Zertifikatsprüfungen: Einem Verifizierer stehen sämtliche im X.509-Standard spezifizierten Daten in einer gesicherten Form zur Verfügung, nämlich entweder als Bestandteile des vorliegenden Zertifikats *AttributeCertificateInfo* oder als Bestandteile der in die Berechnung der Wurzelsignatur ebenfalls einfließenden *CVT-Infos* (siehe auch Abb. 4-2).

```
AttributeCertificateInfo ::= SEQUENCE {
    issuer AttCertIssuer
    holder Holder,
    serialNumber CertificateSerialNumber,
    attrCertValidityPeriod AttCertValidityPeriod,
    attributes SEQUENCE OF Attribute,
    extensions Extensions OPTIONAL          }

CVT-Infos ::= SEQUENCE {
    version AttCertVersion,
    issuer AttCertIssuer,
    signature AlgorithmIdentifier,
    attrCertValidityPeriod AttCertValidityPeriod,
    issuerUniqueID UniqueIdentifier OPTIONAL }
```

Abb. 4-3: Auslagerung redundanter Daten in die CVT-Wurzel (CVT-Infos)

- **Festlegung der Gültigkeitsperiode von Attributszertifikaten:** Da in einem System nun zu jedem Zeitpunkt nur die durch den aktuellsten CVT erfassten Zertifikate als gültig gelten, kann auf die mehrfache Speicherung der Gültigkeitsperiode dieser Zertifikate (*AttrCertValidityPeriod)* ebenfalls verzichtet werden.
  Statt dessen kann unter *CVT-Infos* eine, für alle durch den Baum aktuell signierte Attributszertifikate, einheitliche Gültigkeitsperiode festgelegt werden. Ein unter *CVT-Infos* abgelegtes *AttrCertValidityPeriod*–Feld gibt somit den Zeitpunkt der letzten (in *NotBeforeTime*) sowie der nächsten geplanten (in *NotAfterTime*) Aktualisierung des Baums und somit aller durch diesen signierter Zertifikate. Durch diese Vereinheitlichung der Gültigkeitsperiode der Zertifikate kann der *O(n)* entsprechende Editier-Aufwand reduziert werden, der ansonsten bei jeder Aktualisierung eines Zertifikatsbestands auftreten und die maximal erreichbare Aktualisierungshäufigkeit stark limitieren würde.
  Neben *CVT-Infos* kann jedoch weiterhin ein jedes Attributszertifikat jeweils ein „eigenes“ *AttrCertValidityPeriod*–Feld enthalten. In diesem Feld kann die für dieses Zertifikat bei dessen Erstausstellung vorgesehene (lange) Gültigkeitsperiode festgehalten werden. Diese Gültigkeitsperiode, die also nicht bei jeder Aktualisierung des CVT geändert werden muss, kann eine administrative Funktion erfüllen: Nach Erreichen des Ablaufsdatums (*notAfterTime*) kann durch die Zertifizierungsstelle entschieden werden, ob das jeweilige Zertifikat weiterhin durch den Baum reflektiert werden soll.

- **Lebenszyklusmanagement der Zertifikate**: Die durchgängige Pflege der Attributszertifikate (*AttributeCertificateInfo*) über ihren gesamten Lebenszyklus hinweg (siehe Kap. 2.3.1 und Abb. 2-8) wird nun mit der Pflege des diese signierenden CVT verzahnt:

    - Beim Ausstellen eines (neuen) Zertifikats muss es (genauer sein Hashwert) in den Baum eingefügt werden. Im Fall eines binären Baums bedeutet dies im Allgemeinen das Hinzufügen eines neuen CVT-Blatts.

    - Für die Speicherung des CVT muss eine für die Verifizierer stets erreichbare Online-Datenbank aufgestellt werden. Diese Datenbank generiert für die anfragenden Verifizierer die jeweils gewünschten Zertifizierungspfade zu den Attributszertifikaten.

    - Ein Zertifikat kann frühzeitig, d.h. vor dem Ablauf der im Zertifikat abgelegten (langen) Gültigkeitsperiode, widerrufen werden. In diesem Fall muss es aus dem Baum entfernt werden und es darf nach der nächsten Aktualisierung nicht mehr in der Wurzelsignatur reflektiert werden.

    - Bei der Archivierung eines Zertifikats muss nun auch der passende Zertifizierungspfad (evtl. der komplette CVT) archiviert werden.

- **Einsatz in Client-Push-Systemen:** In einem System, welches dem Client-Push-Modell (siehe Kap. 2.1.3) folgt, können die Zertifikatsbesitzer die scheinbar unsignierten Attributszertifikate (*AttributeCertificateInfo*) erhalten und diese aufbewahren. Beim Zugriff auf das abgesicherte System senden sie die vom Verifizierer jeweilig gewünschten Zertifikate diesem zu. Um ein auf diese Weise erhaltenes Zertifikat zu prüfen, muss sich der Verifizierer an die Datenbank, die den jeweilig relevanten CVT der Zertifizierungsstelle (*Issuer*) speichert, wenden und dort den passenden Zertifizierungspfad anfordern. Aufgrund der Antwort der Datenbank (Zertifizierungspfad) wird entschieden, ob ein vorgezeigtes Zertifikat akzeptiert oder abgelehnt wird.

- **Einsatz in Server-Pull-Systemen:** Im Unterschied zum obigen Fall ist es hier sinnvoll, eine Online-Datenbank aufzusetzen, die sowohl die Zertifikate (*AttributeCertificateInfo*) als auch den diese signierenden CVT speichert. Auf Anfragen der Verifizierer hin kann eine solche Datenbank neben dem angeforderten Zertifikat auch gleich den zugehörigen Zertifizierungspfad aushändigen. Dies ermöglicht beispielsweise einen schnelleren Aufbau eines aus evtl. vielen Zertifikaten bestehenden Benutzerprofils.

- **Prozess der Zertifikatsprüfung:** Durch die Verwendung von CVTs wird der in Abb. 4-1 dargestellte Prozess der Zertifikatsprüfung verändert. Um ein vorliegendes Attributszertifikat zu überprüfen, muss nämlich die Signatur an der CVT-Wurzel mit Hilfe des Zertifizierungspfads rekonstruiert und anschließend verifiziert werden (siehe Abb. 4-4 links).

Die Phase der Gültigkeitsprüfung kann sich auf die Prüfung der Aktualität der Wurzelsignatur, d.h. auf den Eintrag *AttrCertValidityPeriod* innerhalb von *CVT-Infos,* beschränken. Ist die Wurzelsignatur abgelaufen, so muss bei der Datenbank ein neuer, aktualisierter Zertifizierungspfad angefordert werden.
Enthält das Attributszertifikat ebenfalls einen *AttrCertValidityPeriod*-Eintrag, welcher bereits überschritten (*NotAfterTime*) wurde oder noch gar nicht begonnen (*NotBeforeTime*) hat, kann das Zertifikat als ein irrtümlich im Baum reflektiertes Datum behandelt werden. In diesem Fall kann entschieden werden, ob es trotzdem akzeptiert oder abgelehnt wird.

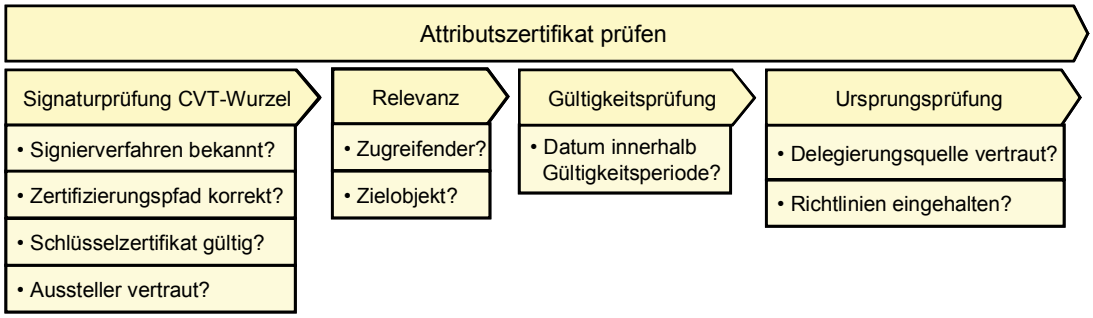


Abb. 4-4: Ablauf der Zertifikatsprüfung mithilfe eines CVT

- **Durchschnittliche Kosten der Zertifikatsprüfung:** Wie das auch aus dem in Kap. 4.1 behandelten Beispiel hervorgeht, bedarf die Prüfung eines einzigen Zertifikats anhand von einem (binären) CVT etwa $O(\log_2 n)$ Hashwertberechnungen sowie genau einer, d.h. $O(1)$, Signaturprüfung. Dabei steht n für die Anzahl der durch den jeweiligen CVT signierten Zertifikate. Die Kommunikationskosten werden maßgeblich durch die Länge der Zertifizierungspfade beeinflusst. Sie liegen im Bereich $O(\log_2 n)$, da hier entsprechend viele Hashwerte zu übermitteln sind.

4.3 Probleme mit CVTs und Lösungswege

Obwohl CVTs grundsätzlich geeignet sind, einen Zertifikatsbestand häufig zu aktualisieren sowie einzelne Attributszertifikate effizient zu prüfen [Nes00], bereitet deren Verwendung einige Probleme. Im Folgenden werden implementierungs- sowie sicherheitstechnische Probleme mit CVTs diskutiert und Lösungswege für diese Probleme vorgeschlagen.

4.3.1 Effizienz der Prüfung von Zertifizierungspfaden

Ein Problem macht sich während der Verifizierung der Zertifizierungspfade bemerkbar. In Kap. 4.1 wurde ein binärer Beispielbaum (siehe Abb. 4-2) behandelt. Es wurde dort exemplarisch gezeigt, auf welche Art und Weise ein Zertifizierungspfad geprüft werden kann. Ein wichtiger Schritt innerhalb dieser Prüfung ist die Rekonstruktion des Wurzelhashwerts. Im behandelten Beispiel wurde angenommen, dass ein Verifizierer irgendwie „wissen“ kann, wie die Hashfunktion anhand der in einem Zertifizierungspfad vorliegenden Daten parametrisiert werden muss.

Anhand des in der linken Hälfte von Abb. 4-5 gezeigten CVT lässt sich zum Zertifikat Z_2 mit der Seriennummer (*SerialNumber*) *2* der bereits in Kap. 4.1 behandelte Zertifizierungspfad

$$ZP(Z_2)=[\ S \,||\, H_2 \,||\, CVT\text{-}Infos\ ]$$

generieren.

Dieser wird verifiziert, indem der Wurzelhashwert mit

$$H_W'=h(h(h(Z_2) \,||\, H_2) \,||\, CVT\text{-}Infos)$$

rekonstruiert und anschließend durch eine Signaturprüfung überprüft wird (siehe Kap. 4.1).

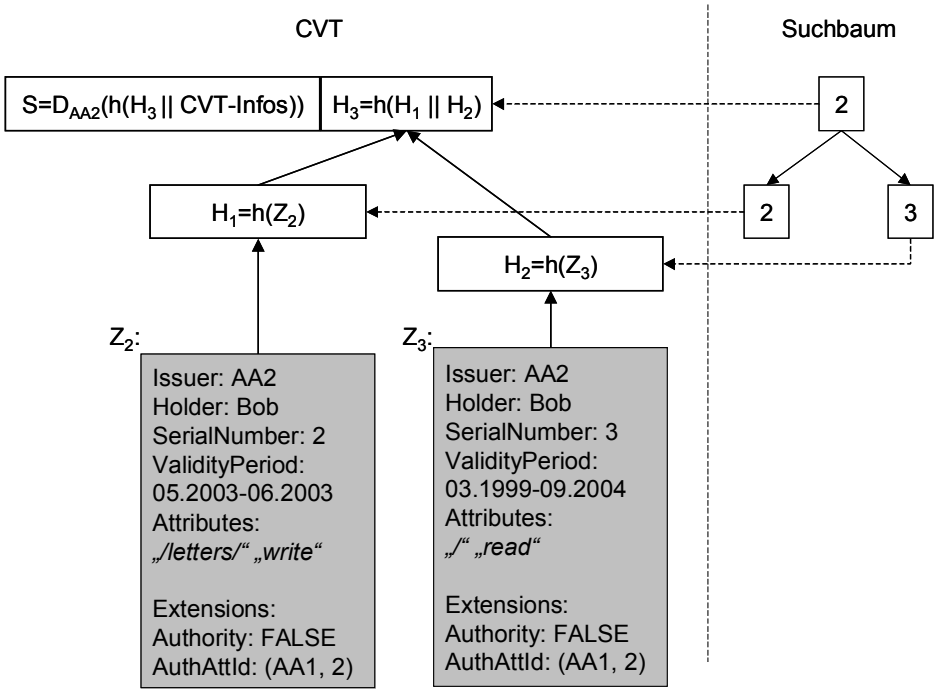


Abb. 4-5: Suchbaum indiziert die Knoten eines binären CVT

Die Frage, die sich hier stellt ist, woher ein Verifizierer „weiss", in welcher Reihenfolge $h(Z_2)$ und H_2 in die obige Berechnung von $H_W'=h(h(h(Z_2)||H_2)||CVT\text{-}Infos)$ einfließen müssen. Die korrekte Reihenfolge der beiden Argumente ist nämlich eine Voraussetzung der Berechnung des korrekten Wurzelhashwerts.
Ist die Reihenfolge der beiden Argumente falsch, so schlägt die Prüfung (trotz des an sich korrekten Zertifizierungspfads) fehl. Die Wahrscheinlichkeit, dass ein Verifizierer diese beiden Argumente der Hashfunktion h in der korrekten Reihenfolge übergibt, beträgt *1/2*. Hat ein binärer CVT k Ebenen, so liegt die Wahrscheinlichkeit für die Ermittlung des korrekten Wurzelhashwerts bei $1/2^{k-1}$.
Um die Verifizierung der Zertifizierungspfade effizienter als das Durchprobieren von im Schnitt 2^{k-1} möglichen Kombinationen zu gestalten, müssen die Zertifizierungspfade entsprechende „Navigationsinformationen" enthalten, welche die richtige Einordnung der Hashwerte ermöglichen. Hierfür wird nun zum Einen die (aufsteigende) Sortierung der durch den Baum signierten Zertifikate und zum Anderen die Verwendung eines Suchbaums zur Indizierung der einzelnen CVT-Knoten vorgeschlagen (siehe Abb. 4-5).

- **Wahl eines Sortierschlüssels:** Als Sortierschlüssel kann das Feld *SerialNumber* der Zertifikate genommen werden. Dieses Feld ist ein geeigneter Sortierschlüssel, denn es muss laut des X.509-Standards [ITU01] die durch eine Zertifizierungsstelle ausgestellten Zertifikate eindeutig voneinander unterscheiden (siehe auch Kap. 2.2.4). Außerdem nimmt *SerialNumber* standardgemäß Werte aus dem geschlossenen Intervall *[0; Max_Int]* der ganzen Zahlen auf [ITU01], was die Verwendung gängiger Sortieralgorithmen [Knu98] erleichtert.
- **Referenzierung der Knoten durch einen Suchbaum:** Einem auf diese Weise erzeugten (binären) CVT wird ein (binärer) Suchbaum [Knu98] zugeordnet. Als Sortierschlüssel für diesen Suchbaum wird ebenfalls das Feld *SerialNumber* der Zertifikate gewählt. Die internen Knoten (Blätter) des verwendeten Suchbaums enthalten jeweils einen Zeiger auf genau einen internen Knoten (ein Blatt) des CVT. Dabei muss ein Blatt des Suchbaums genau den Sortierschlüssel (*SerialNumber*) enthalten wie das Zertifikat, dessen Hashwert das referenzierte CVT-Blatt enthält. Der Vater von *k* Knoten (Blättern) des Suchbaums enthält einen Zeiger auf den Vaterknoten der entsprechenden *k* CVT-Knoten (CVT-Blätter), die durch diese Suchbaumknoten (-blätter) referenziert werden.

Eine in diesem Sinne entstandene Konstellation stellt Abb. 4-5 dar. Wie man sieht, wurden die beiden Zertifikate Z_2 und Z_3 von *Bob* in den CVT gemäß der über deren Seriennummern (*SerialNumber*) erfolgte aufsteigende Sortierung aufgenommen. Die Knoten des in der rechten Hälfte von Abb. 4-5 abgebildeten binären Suchbaums [Knu98] referenzieren jeweils einen CVT-Knoten (siehe die gestrichelten Pfeile).

Eine Datenbank, welche sowohl den CVT als auch den zugehörigen Suchbaum speichert, kann nun Zertifizierungspfade aushändigen, die neben den Hashwerten aus dem CVT auch die zugehörigen Sortierschlüssel aus dem Suchbaum enthalten. Der Zertifizierungspfad $ZP(Z_2)$ zum Zertifikat Z_2 mit der Seriennummer *2* kann demnach die folgende Form haben:

$$ZP(Z_2)=[\ S=D_{AA2}(\ h(H_3\ ||CVT\text{-}Infos\ ))\ ||\ \{3,\ H_2\}\ ||\ CVT\text{-}Infos\ ]$$

Dieser $ZP(Z_2)$ enthält neben den anderen Daten nun ein {*SerialNumber, Hashwert*}-Paar, nämlich $\{3, H_2\}$. Anhand von der Seriennummer *2* des vorliegenden Zertifikats Z_2 sowie anhand vom Hashwert H_2 gehörenden aus dem Suchbaum stammenden Sortierschlüssel *3* kann festgestellt werden, dass Z_2 im linken Teilbaum der CVT-Wurzel liegt (da $2<3$).
Folglich kann die anvisierte Berechnung $H_W'=h(h(h(Z_2)\ ||\ H_2)\ ||\ CVT\text{-}Infos)$ zur Rekonstruktion des Wurzelhashwerts korrekt durchgeführt werden. Dabei werden der Hashwert $h(Z_2)$ des vorliegenden Zertifikats in die linke und der Hashwert H_2 in die rechte Position bei der Berechnung von $h(h(Z_2)\ ||\ H_2)$ gesetzt.
Zu beachten ist, dass diese Konstruktion weder auf die bei der Erzeugung des CVT noch auf die bei der Prüfung eines Zertifizierungspfads getätigten Berechnungsschritte einen Einfluss nimmt. Diese Schritte werden in einer unveränderten Form durchgeführt. Das Konzept ist aber hilfreich, um die Effizienz der Berechnungen während einer Zertifikatsprüfung zu erhöhen. Außerdem kann der zur Hilfe gestellte Suchbaum auch seitens der Datenbank genutzt werden, um die Zertifizierungspfade zu generieren. Hierbei kann die Suche nach den jeweils auszuliefernden Hashwerte aus dem CVT mit Hilfe dieses Suchbaums erfolgen.

4.3.2 Behandlung sicherheitskritischer Probleme

Die durch einen CVT signierten Zertifikate können nach ihrer Erstellung nicht mehr unentdeckt verändert werden. Hierfür sorgen die Wurzelsignatur sowie die verwendete kollisionsresistente Hashfunktion. Sind diese hinreichend sicher, werden während der Verifizierung jegliche nachträgliche Manipulationen der zertifizierten Daten entdeckt.

Deshalb können sämtliche durch einen aktuellen CVT signierten Attributszertifikate als gültig gelten. Dies impliziert aber auch, dass alle anderen Zertifikate, welche also nicht durch den aktuellen Baum signiert werden, ungültig sind.
Eine Online-Datenbank, die einen CVT speichert, muss primär Anfragen der Verifizierer nach Zertifizierungspfaden beantworten. Da es sich hier um ein Online-System mit entsprechenden Kommunikationsschnittstellen nach außen hin handelt, muss hier mit verschiedenen Angriffen gerechnet werden. Beispielsweise können eingeschleuste Fremdprogramme (so genannte Trojanische Pferde) die Steuerung der Datenbank inklusive der Beantwortung der Anfragen übernehmen. Außerdem kann nicht immer garantiert werden, dass der womöglich externe Betreiber der Datenbank stets im Sinne der Zertifizierungsstellen handelt [NEA02]. Folglich kann und muss diese Datenbank als eine nicht vertrauenswürdige Komponente des Systems eingestuft werden. Dies steht auch im Einklang mit den diesbezüglichen Empfehlungen des X.509-Standards [ITU01].
Angenommen, die Datenbank wird während der Zugriffskontrolle nach dem Zertifizierungspfad zu einem dem Verifizierer vorliegenden Attributszertifikat gefragt. Wenn sich zum angefragten Zertifikat anhand des jeweiligen CVT ein Zertifizierungspfad generieren lässt, wird dieser im Normalfall ausgehändigt. Nach der Prüfung des Zertifizierungspfads kann das Zertifikat akzeptiert oder abgelehnt werden.
Ist hingegen das gefragte Zertifikat im CVT nicht reflektiert, kann eine entsprechende Fehlermeldung, wie z.B. „*Certificate not found*“ generiert werden. Das Problem mit diesem (in heutigen Datenbankmanagementsystemen üblichen) Vorgehen ist, dass die kompromittierte oder maliziöse Datenbank die obige Fehlermeldung jederzeit schicken kann, auch wenn das gefragte Zertifikat in Wirklichkeit durch den Baum signiert wurde. Durch solche falschen Antworten kann die Datenbank ein gültiges Attributszertifikat für ungültig erklären. Die Folgen aus Sicht einer effektiven zertifikatsbasierten Zugriffskontrolle sind fatal:

- Beinhaltet ein zu prüfendes Attributszertifikat (im Feld *Attributes*) tolerierte Zugriffswege (siehe Kap. 2.1), so kann das Zurückhalten des passenden Zertifizierungspfads die Zugriffskontrolle zur Verweigerung ansonsten erlaubter Zugriffe bewegen. Dies kann beispielsweise in Systemen mit konkurrierenden Benutzern von Bedeutung sein, wie z.B. in einem Auktionsportal. Wird hier der Betreiber der Datenbank bestochen, um Zugriffsrechte bestimmter Benutzer auf diese Weise zu blockieren, so können Gebotsversuche dieser Benutzer von der Zugriffskontrolle abgelehnt werden. Mangels Konkurrenz können die unehrlichen Kunden günstigere Kaufpreise erreichen.

- Noch kritischer erscheint, wenn ein zurückgehaltener Zertifizierungspfad die Gültigkeit eines Attributszertifikats mit unerlaubten Zugriffswegen (siehe Kap. 2.1) bestätigen würde. Solche Zugriffswege gelten für die Zugriffskontrolle als „Verbotsregeln“.

Werden solche verbotenen Zugriffswege als Folge mangelnder Zertifikatspfade nicht betrachtet, so kann dem jeweiligen Zugreifenden ein eigentlich unerlaubter Zugriff fälschlicherweise genehmigt werden.

Durch die Möglichkeit diese Art Angriffe auszuführen, muss der eigentlich nicht vertrauenswürdigen Online-Datenbank bezüglich der Korrektheit ihrer Antworten vertraut werden. Einen Ausweg aus dieser Situation bieten die nachfolgend vorgestellten Mechanismen an. Dank dieser wird es möglich, die Antworten der Datenbank stets auf den vertrauenswürdigen Signierenden des jeweiligen CVT zurückzuführen.

4.3.2.1 Einfügen von Fehleinträgen in den CVT

Die Kernidee hinter dem nachfolgend diskutierten Konzept ist die folgende: Die Datenbank, welche einen Zertifizierungspfad zu einem bestimmten Zertifikat mit der Seriennummer *y* nicht aushändigen kann, muss einen Beweis generieren, dass dieses Zertifikat im jeweiligen CVT tatsächlich nicht reflektiert wird. Statt der Fehlermeldung „*Certificate not found*“ muss dieser Beweis gesendet und anschließend durch den Verifizierer überprüft werden. Hierfür werden durch den Erzeuger so genannte Fehleinträge in den Baum eingefügt, wie das in Abb. 4-6 veranschaulicht wird. (Auf die Darstellung des den CVT indizierenden Suchbaums wurde in Abb. 4-6 aus Platzgründen verzichtet.)

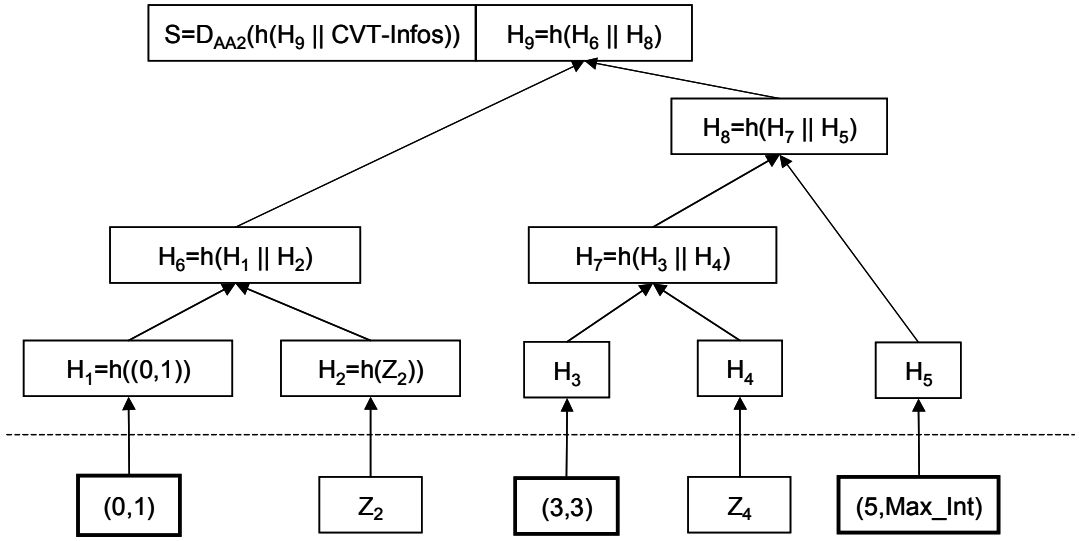


Abb. 4-6: CVT mit Fehleinträgen zu nicht signierten Daten

Angenommen, durch einen CVT werden insgesamt $k>0$ Zertifikate signiert. Diese werden (dem Entwurf in Kap. 4.3.1 entsprechend) nach ihren Seriennummern (*SerialNumber*) sortiert in den Baum eingefügt. Sei $n>k$ die Anzahl der möglichen Seriennummern, die ihre Werte aus dem geschlossenen Intervall *[0; Max_Int]* der ganzen Zahlen aufnehmen können [ITU01]. Das Fehlen der in den Baum noch (in Zukunft) hinzuzufügenden beziehungsweise aus dem Baum bereits entfernten insgesamt $n-k$ Zertifikate wird durch mithilfe des Baums signierte Fehleinträge bestätigt (siehe die fettgedruckten Kästen in Abb. 4-6). Ein solcher Eintrag bestätigt das Fehlen von insgesamt $0<r\leq n$ (gesperrten oder noch nicht ausgestellten) Zertifikaten mit den Seriennummern aus dem Intervall *[First-Non-Signed; Last-Non-Signed]*.
Dabei gilt, dass die Zertifikate mit den Seriennummern *First-Non-Signed-1* und/oder *Last-Non-Signed+1* durch den Baum signierte Zertifikate sind. Eine Ausnahme hiervon bildet ein (initialer) CVT, welcher (noch) kein einziges Zertifikat signiert.

Ein solcher CVT enthält einen einzigen Fehleintrag mit *First-Non-Signed=0* und *Last-Non-Signed=Max_Int*.
Wird nun nach dem Zertifizierungspfad eines der *k* durch den CVT aktuell signierten Zertifikate gefragt, so wird dieser auf die in Kap. 4.3.1 exemplarisch behandelte Art und Weise erzeugt und geprüft.
Wird hingegen nach dem Status eines durch den Baum nicht signierten Zertifikats mit der Seriennummer *y* gefragt, muss der passende Fehleintrag *F* mit

$$F_{First\text{-}Non\text{-}Signed} \leq y \leq F_{Last\text{-}Non\text{-}Signed}$$

sowie der Authentifizierungspfad *AP(F)* zu *F* ausgehändigt werden. (In diesem Fall kann von keinem Zertifizierungspfad die Rede sein.) *F* sowie *AP(F)* werden anschließend auf ihre Korrektheit überprüft.
Wird beispielsweise nach dem durch den Baum in Abb. 4-6 nicht signierten Zertifikat mit der Seriennummer (*SerialNumber*) *3* gefragt, so werden der Fehleintrag *{3, 3}* sowie der passende Authentifizierungspfad

$$AP(\{3, 3\}) = [S \,||\, \{H_6 \,||\, H_5 \,||\, H_4\} \,||\, CVT\text{-}Infos]$$

ausgehändigt. Die Überprüfung von *AP({3, 3})* erfolgt analog zu den bereits behandelten Beispielen. Erweist sich *S* als eine gültige Signatur für den Fehleintrag, so kann ein Verifizierer davon ausgehen, dass das gefragte Zertifikat tatsächlich nicht im CVT reflektiert wurde.
Da die Datenbank nun gezwungen wird, auf jede Anfrage einen auf die Zertifizierungsstelle zurückzuführenden signierten Nachweis zu erzeugen, sind ihre Manipulationsmöglichkeiten extrem eingeschränkt. Sie kann nun höchstens die Beantwortung der Anfragen ganz ablehnen, d.h. nichts „sagen“. Nach Ablauf einer bestimmten Zeitperiode kann dies zu ihrer sofortigen Ablösung beispielsweise durch eine andere Datenbankinstanz mit replizierten Daten führen. Dieses Konzept hat einige leicht erkennbare Auswirkungen auf die Leistungsfähigkeit des Systems:

- **Redundanz der Information:** Im CVT werden neben den signierten auch frühzeitig gesperrte beziehungsweise nicht weiter verlängerte aber auch noch gar nicht ausgestellte Zertifikate durch die Fehleinträge repräsentiert. Dank dieser Daten kann sichergestellt werden, welche Zertifikate im Baum *nicht* reflektiert wurden. Sie stellen aber im Grundsatz redundante Informationen dar.

- **Erhöhter administrativer Aufwand:** Die konsequente Pflege eines CVT wird in dieser Konstruktion komplizierter. Beim Einfügen oder Löschen eines Zertifikats müssen nämlich auch die hiervon betroffenen Fehleinträge entsprechend modifiziert beziehungsweise gelöscht oder neue Fehleinträge hinzugefügt werden.

- **Erhöhte Verifizierungskosten:** Ein CVT mit *k* signierten Zertifikaten wird im schlimmsten Fall *2k+1* statt *k* Blätter und entsprechend mehr interne Knoten haben. Beispielsweise hat der Baum in Abb. 4-6 mit insgesamt zwei signierten Zertifikaten fünf Blätter. Hieraus folgt, dass die durchschnittlich erwartete Baumhöhe größer wird. Dies führt wiederum zu einer erhöhten durchschnittlichen Länge der Zertifizierungspfade (im Fall eines binären CVT um maximal zwei Einträge).

Dadurch werden während der Prüfung eines Zertifizierungspfads einige zusätzliche Hashwertberechnungen benötigt, um den Wurzelhashwert zu rekonstruieren.

Diese zum Teil als nachteilig erscheinenden Punkte motivieren die Suche nach einer anderen Lösung des Problems, welche ohne das Einfügen zusätzlicher Fehleinträge in den Baum auskommt.

4.3.2.2 Zertifizierungspfade mit gesicherten Suchschlüsseln

Das nachfolgend diskutierte Konzept verwendet Informationen aus dem in Kap. 4.3.1 zur Referenzierung der CVT-Knoten eingeführten Suchbaum. Anhand der im Suchbaum enthaltenen Suchschlüssel kann nämlich ebenfalls bestätigt werden, wenn ein gefragtes Zertifikat nicht durch den referenzierten CVT signiert wird. Diese Vorgehensweise wird nun anhand eines Beispiels erläutert.
Der in der oberen Hälfte von Abb. 4-7 gezeigte CVT ist identisch mit dem aus Abb. 4-5 bereits bekannten Baum. Die entsprechenden Sortierschlüssel aus dem zugehörigen Suchbaum sind nun neben den jeweilig referenzierten CVT-Knoten dargestellt (siehe die kleinen Kästen).

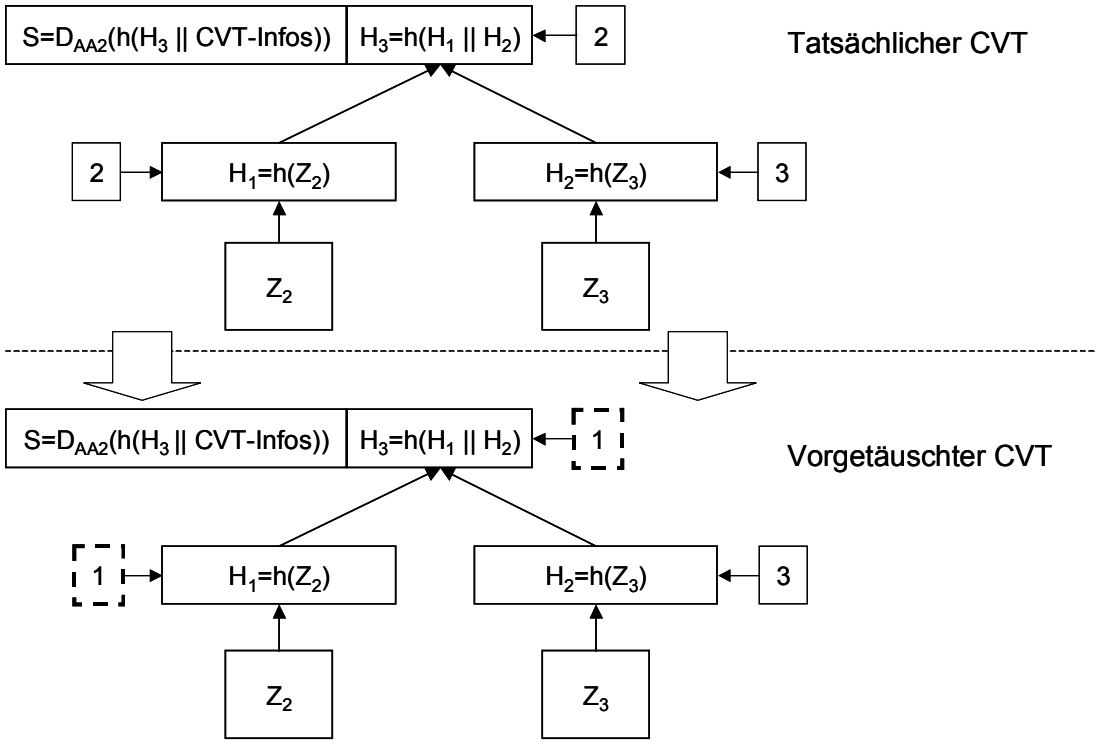


Abb. 4-7 Unbemerkte Manipulation der Zertifizierungspfade

Wird nun nach dem durch diesen Baum nicht signierten Attributszertifikat mit der Seriennummer *42* gefragt, so kann die Datenbank mit dem (redundanten) Zertifizierungspfad

$$ZP(42)=[D_{AA2}(\, h(H_3 \,||\, \text{CVT-Infos}\,)) \,||\, \{3, H_2\} \,||\, \{2, H_1\} \,||\, \{2, H_3\} \,||\, \text{CVT-Infos}\,]$$

der Wahrheit entsprechend belegen, dass das Zertifikat im CVT nicht reflektiert wird. Aufgrund der in *ZP(42)* enthaltenen Seriennummern kann nämlich festgestellt werden, dass das gesuchte Zertifikat im rechten Teilbaum der Wurzel reflektiert sein müsste: Dem Hashwert H_3 in der Wurzel wurde nämlich die Seriennummer *2* zugeordnet.

Da $2<42$ gilt, müsste das gefragte Zertifikat im rechten Teilbaum der Wurzel reflektiert werden.
Wie man aber in der oberen Hälfte von Abb. 4-7 sieht, enthält der rechte Kindknoten der Wurzel, welcher ein Blatt des Baums ist, den Hashwert H_2. Diesem H_2 wurde aber die ebenfalls ausgehändigte Seriennummer *3* und nicht etwa *42* zugeordnet.

Durch die Rekonstruktion des Wurzelhashwerts H_W' durch

$$H_W' = h(h(H_1 \,||\, H_2) \,||\, CVT\text{-}Infos),$$

sowie durch die anschließende Signaturprüfung kann die Authentizität von $ZP(42)$ sichergestellt werden. Somit kann also eine negative Antwort seitens der Datenbank auf die von der Zertifizierungsstelle stammende signierte Datenstruktur zurückgeführt werden.
Diese Vorgehensweise ist effizienter als das weiter oben diskutierte Einfügen von Fehleinträgen, sie ist aber leider angreifbar. Die Datenbank kann nämlich Verifizierern, welche den in der Datenbank abgelegten CVT nicht kennen (können), durch falsche Navigationsangaben eine von der tatsächlichen abweichende Baumstruktur vortäuschen. Hierfür muss sie keine signierten Baumteile verändern. Diese Art Manipulation würde nämlich spätestens bei der Prüfung der Wurzelsignatur entdeckt, d.h. ein solcher Angriff ist unwirksam. Die mögliche Täuschung wird mit Hilfe der unteren Hälfte von Abb. 4-7 demonstriert.
Auf die Anfrage eines Verifizierers nach dem vorhandenen Zertifikat Z_2 muss die Datenbank im Normalfall den bereits behandelten Zertifizierungspfad

$$ZP(Z_2)=[\ S=D_{AA2}(\ h(H_3 \,||\, CVT\text{-}Infos\ ))\ ||\ \{3,\ H_2\}\ ||\ CVT\text{-}Infos\ ]$$

aushändigen. Die Überprüfung von $ZP(Z_2)$ wurde bereits in Kap. 4.3.1 behandelt.
Will nun die Datenbank den Verifizierer betrügen, so kann sie beispielsweise den der Wahrheit nicht entsprechenden Zertifizierungspfad

$$ZP(Z_2)'=[\ D_{AA2}(\ h(H_3 \,||\, CVT\text{-}Infos\ ))\ ||\ \{3,\ H_2\}\ ||\ \{1,\ H_1\}\ ||\ \{1,\ H_3\}\ ||\ CVT\text{-}Infos\ ]$$

aushändigen. Die dabei fälschlicherweise angegebenen Suchschlüssel sieht man in Abb. 4-7 in den fett und gestrichelt gedruckten Kästchen. Durch die Rekonstruktion des Wurzelhashwerts erweist sich dieser Pfad definitionsgemäß als gültig. Er veranlasst gleichzeitig den Verifizierer zu glauben, dass gesuchtes Z_2 sich im rechten (statt im linken) Teilbaum der Wurzel befinden muss. Hierfür wurde nämlich in $ZP(Z_2)'$ neben dem Hashwert H_3 der vorgetäuschte Suchschlüssel *1* eingefügt ($2>1$).
Der Eintrag $\{3,\ H_2\}$ ist aber ein eindeutiger Hinweis darauf, dass der rechte Kindknoten der CVT-Wurzel den Suchschlüssel *3* ($3>2$) hat. Dadurch wird der Verifizierer überzeugt, dass gefragtes Z_2 nicht durch den CVT signiert wurde. In seinem Glauben kann ihn die ebenfalls gefälschte Angabe zum linken Kindknoten der Wurzel $\{1,\ H_1\}$ bestätigen. Wie man sieht, ist H_1 der Hashwert des gefragten Z_2. Wegen der erwarteten Einwegfunktionalität der Hashfunktion *h* lässt sich dies natürlich nicht ohne Weiteres herausfinden.
Aus diesen Überlegungen folgt: Obwohl der CVT korrekt erzeugt und durch einen Suchbaum korrekt indiziert wurde, kann eine Datenbank, welche den Baum speichert, die ausgehändigten Zertifizierungspfade unentdeckt manipulieren.

Dadurch kann sie die Existenz eines durch den Baum signierten Attributszertifikats unentdeckt abstreiten. Ihre falschen Antworten kann sie sogar auf die authentische Wurzelsignatur zurückführen.
Im Folgenden wird eine Lösung dieses Problems diskutiert. Der gewählte Ansatz ist die Absicherung des zur Indizierung der CVT-Knoten verwendeten Suchbaums, welcher die im obigen Angriff fälschlicherweise ausgehändigten Suchschlüssel (Navigationsinformation) enthält.
Eine adäquate Absicherung von Suchbäumen gegen nachträgliche Manipulationen ermöglicht die Technik mit der Bezeichnung *Undeniable Authenticated Search Tree Attester* (UASTA). Diese Technik wurde zunächst in [BLL00] sowie in einer verbesserten Form in [BLL02] veröffentlicht (siehe auch [Sär00]).
Betrachtet wurden dabei binäre Suchbäume [Knu98], welche insgesamt n Daten über ihre eindeutige Suchschlüssel $s_1, s_2, ..., s_n$ indizieren. Die Kernidee besteht darin, die jeweilige Sortierung der durch den Baum indizierten Daten zu schützen. Dies geschieht durch eine an der Wurzel des Suchbaums getätigte digitale Signatur. Die signierte Wurzel enthält dabei einen Wurzelhashwert, in dessen Berechnung sämtliche im jeweiligen Suchbaum enthaltenen Suchschlüssel einfließen.
Ein jeder Knoten *knoten* des (binären) Suchbaums einschließlich der Wurzel mit $k=\{0, 1, 2\}$ Kindern enthält nämlich neben dem jeweiligen Suchschlüssel s_i ($0<i\leq n$) auch einen Hashwert $knoten.wert=h((knoten.kind_{links}).wert \,||\, s_i \,||\, (knoten.kind_{rechts}).wert)$.
Durch diese Vorgehensweise entsteht also ein signierter Suchbaum, welcher für die Sicherung der jeweiligen Sortierung der referenzierten Daten geeignet ist.

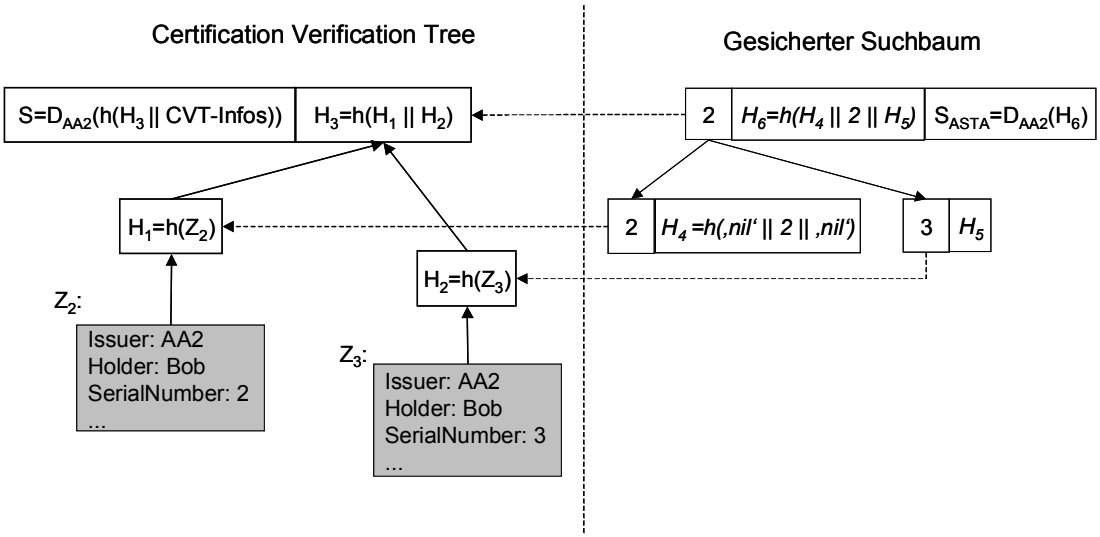


Abb. 4-8: CVT-Struktur wird durch einen signierten Suchbaum gesichert

Eine auf diese Art und Weise abgesicherte CVT-Suchbaum-Konstellation wird in Abb. 4-8 dargestellt (vgl. Abb. 4-5). Wie man sieht, wird der Suchbaum, der die auf der linken Seite abgebildeten CVT-Knoten referenziert, durch eine Signatur S_{ASTA} an dessen Wurzel geschützt. Diese Signatur muss von der gleichen Stelle (hier *AA2*) erzeugt werden, die auch den jeweiligen CVT signiert hat.
Die Datenbank, welche zu einem gefragten Zertifikat einen Zertifizierungspfad anhand des CVT aushändigt, muss nun die zur Prüfung des Zertifizierungspfads benötigten Navigationsinformationen ausgehend von dem signierten Suchbaum generieren.

Das bedeutet aber auch, sie muss nun den passenden Suchpfad (engl. *Search Path*, SP) sowie die signierte Wurzel des Suchbaums ausliefern. Diese Vorgehensweise wird nun an einem Beispiel verdeutlicht: Der bereits in Kap. 4.3.1 behandelte Zertifizierungspfad $ZP(Z_2)$ zum Zertifikat Z_2

$$ZP(Z_2)=[\ S=D_{AA2}(\ h(H_3 \mid\mid CVT\text{-}Infos\ ))\ \mid\mid \{3,\ H_2\} \mid\mid CVT\text{-}Infos\ ]$$

wird nun ergänzt um den gesicherten Suchpfad

$$SP(Z_2)=[\ S_{ASTA}=D_{AA2}(H_6) \mid\mid \{2,\ H_5\} \mid\mid \{,nil',\ 2,\ ,nil'\}\ ].$$

Nach Rekonstruktion und Überprüfung des Wurzelhashwerts des CVT anhand von $ZP(Z_2)$ (siehe Kap. 4.3.1) muss nun überprüft werden, ob der ausgelieferte Suchpfad $SP(Z_2)$ korrekt ist. Dies erfolgt, indem nun der Wurzelhashwert des Suchbaums rekonstruiert und überprüft wird. Hierfür berechnet der Verifizierer zunächst

$$H_6'=h(\ h(,nil',\ 2,\ ,nil') \mid\mid 2 \mid\mid H_5).$$

Die anschließende Entschlüsselung der Wurzelsignatur durch

$$H_6=E_{AA2}(S_{ASTA})$$

hat als Ergebnis den signierten Wurzelhashwert H_6 des Suchbaums. Ist

$$H_6'=H_6,$$

sind die Signatur S_{ASTA} und somit die zu $ZP(Z_2)$ ausgehändigten Navigationsinformationen korrekt, andernfalls handelt es sich um einen manipulierten und daher abzulehnenden Suchpfad.

Durch diese Vorgehensweise können Manipulationsversuche der Datenbank nun sicher ausgeschlossen werden. Versucht etwa die Datenbank, eine falsche Baumstruktur vorzuspielen (siehe die untere Hälfte von Abb. 4-7) und dadurch den Verifizierer bezüglich der Existenz eines Zertifikats zu täuschen, so wird dies spätestens bei der Signaturprüfung des Suchpfads entdeckt. Dieses Konzept kommt ohne Änderungen an der in [GGM00] beziehungsweise in [Mer90] vorgesehenen CVT-Struktur sowie ohne zusätzlich eingefügte Fehleinträge (siehe weiter oben) aus. Der Preis hierfür ist jedoch, dass nun zwei signierte Baumstrukturen verwaltet werden müssen, um die etwaigen Angriffe der Speicherinstanz zu verhindern.
Dies erhöht zum Einen den administrativen Aufwand seitens einer Zertifizierungsstelle. Diese muss stets für die Konsistenzhaltung der beiden Bäume sorgen. Wird ein Zertifikat in einen CVT eingefügt oder aus dem CVT entfernt, führt dies zwangsläufig zu Änderungen des signierten Suchbaums. Folglich verändern sich die Wurzelhashwerte der beiden Bäume, die deshalb (etwa gleichzeitig) neu signiert und der Datenbank zugesendet werden müssen.
Zum Anderen muss eine Datenbank nun zu jedem gespeicherten CVT einen zugehörigen, etwa gleich großen signierten Suchbaum speichern. Dies erhöht also den Speicherbedarf an dieser Stelle.

Aus performanztechnischer Sicht erscheint des Weiteren nachteilig, dass die Datenbank nun zu jedem Zertifizierungspfad auch einen zusätzlichen gesicherten Suchpfad anhand des passenden Suchbaums erzeugen und aushändigen muss.
Dementsprechend muss mit einer im Durchschnitt verlangsamten Zertifikatsprüfung gerechnet werden. Ein Verifizierer muss ja nun stets zwei Signaturprüfungen durchführen, um ein einziges vorliegendes Zertifikat auf seine aktuelle Gültigkeit hin zu überprüfen. Hinzu kommt, dass nun zwei Wurzelhashwerte rekonstruiert werden müssen. Außerdem werden sich die durchschnittlichen Kosten der notwendigen Kommunikation zwischen Verifizierern und der Datenbank etwa verdoppeln.
Um vor allem diesen betriebsbedingten Problemen entgegenzuwirken, wird im nächsten Kapitel die Konstruktion so genannter *Improved Certification Verification Trees* diskutiert.

5 Improved Certification Verification Tree

Die betriebsbedingten bzw. kostentechnischen Nachteile der in Kap. 4.3 vorgestellten Konzepte motivieren die Modifikation Merklescher Authentifizierungsbäume [Mer90]. In Kap. 5.1 wird dazu eine neue Struktur für solche Bäume vorgeschlagen. Dabei wird auf den Einbezug von Fehleinträgen (siehe Kap. 4.3.2.1) und auf die Verwendung einer externen Zusatz-Datenstruktur (siehe Kap. 4.3.2.2) verzichtet.
Ausgehend von diesem modifizierten (binären) Authentifizierungsbaum können Authentifizierungspfade generiert werden, die als Existenz- beziehungsweise Nicht-Existenz-Beweise gelten: Es wird anhand von Beispielen gezeigt, wie mit Hilfe der vorgeschlagenen Baumstruktur die Authentizität und Integrität einzelner durch einen Baum erfasster, d.h. existierender, Datensätze (insbesondere Attributszertifikate) geprüft werden kann. Außerdem wird eine Vorgehensweise vorgestellt, die sicherstellt, dass ein bestimmter durch den Baum nicht erfasster Datensatz nicht existent ist.
Anschließend wird in Kap. 5.2 eine Methodik gezeigt, wie die gewählte Datenstruktur verwendet werden kann, um so genannte Vollständigkeitsbeweise zu erstellen und zu prüfen. Diese Art von Beweisen ermöglichen die Prüfung, ob eine Datenbank die an sie gerichteten Datenbankanfragen nach gleichzeitig mehreren, in der Datenbank (vermutlich) vorhandenen Datensätzen korrekt beantwortet.
Auf diesen Grundlagen wird in Kap. 5.3 die Konstruktion so genannter *Improved Certification Verification Trees* (I-CVT) diskutiert, welchen B^+-Bäume zugrunde gelegt wurden. Die Algorithmen zur Erstellung und Prüfung der verschiedenen Beweisarten werden anhand von einfachen Beispielszenarien vorgestellt.

5.1 Sichere Existenz- und Nicht-Existenz-Beweise

Zunächst werden die beiden Begriffe Existenz- und Nicht-Existenz-Beweis präzisiert, um festzulegen, welche Änderungen an den Authentifizierungsbäumen einzuführen sind.

Def. 5-1: Existenz-Beweis

Ein Existenz-Beweis ist ein auf eine vertraute Instanz zurückführbarer Nachweis. Er kann von einer nicht vertrauenswürdigen Instanz anhand von einer durch die vertraute Instanz angelegten Datenstruktur generiert werden. Er stellt sicher, ob ein gefragter, das jeweilige Suchkriterium erfüllender Datensatz unverändert und authentisch vorliegt.

Es ist beispielsweise leicht einzusehen, dass eine durch *Bob* signierte (sortierte) Liste $L=\{E_1, E_2, ..., E_n\}$ zusammen mit dem Signaturwert $S=D_{Bob}(h(L))$ ein Existenz-Beweis für alle E_i Elemente (Bitfolgen) der Liste ist: Angenommen L sowie S werden in derselben nicht vertrauenswürdigen Datenbank abgelegt.
Ein Verifizierer, der *Bob* vertraut, kann auf seine Anfrage, ob ein bestimmtes Datum x in der Datenbank vorliegt, L und S erhalten. Ist $x=E_i$ $(1 \leq i \leq n)$, so wird dies durch die erfolgreiche Signaturprüfung der Liste $L'=\{E_1, ..., x, ..., E_n\}$, welche in der i-ten Position x enthält, eindeutig bestätigt. Nach dem Signieren von L werden nämlich Manipulationsversuche, wie z.B. das Einfügen eines ursprünglich nicht vorhandenen Listenelements in L, während der Prüfung von S entdeckt.

Dies gilt unter der Annahme, dass die verwendete Hashfunktion h und die Signierfunktion D_{Bob} sicher sind. Es ist jedoch nachteilig, dass zur Prüfung, ob ein bestimmtes Element E_i tatsächlich ein Element der Liste ist, sämtliche n Listenelemente vorliegen müssen.
Im Fall eines Authentifizierungsbaums B, der Datensätze (Bitfolgen) aus der Menge M sichert, gilt es nun Folgendes zu erreichen: Es soll zu einem bestimmten Datum $x \in M$ ein Authentifizierungspfad $AP(x)$ erzeugt werden können, um durch die Verifizierung von $AP(x)$ die Integrität und Authentizität des Datums x zu bestätigen. Dies impliziert, dass die mit Hilfe von $AP(x)$ überprüfbare Wurzelsignatur S von B gültig sein muss. Außerdem müssen Versuche, die Existenz eines durch den Baum nicht signierten Datums $y \notin S$ etwa mit Hilfe eines manipulierten Authentifizierungspfads vorzuspielen, zuverlässig entdeckt werden.
Das heißt, Manipulationsversuche, um die Existenz des in Wirklichkeit nicht vorhandenen Blatts von B vorzutäuschen, welches den Hashwert $h(y)$ von y enthält, müssen entdeckt werden. Die entsprechend ausgestellten Authentifizierungspfade gelten als Existenz-Beweise für die durch B erfassten Daten.

Analog zu Def. 5-1 lassen sich Nicht-Existenz-Beweise folgendermaßen definieren:

Def. 5-2: Nicht-Existenz-Beweis

Ein Nicht-Existenz-Beweis ist ein auf eine vertraute Instanz zurückführbarer Nachweis. Er kann bei einer Anfrage von einer nicht vertrauenswürdigen Instanz anhand von einer durch die vertraute Instanz angelegten Datenstruktur generiert werden. Er stellt sicher, dass keiner der durch die Datenstruktur erfassten Datensätze das jeweilige Suchkriterium erfüllt.

Es ist leicht einzusehen, dass eine von *Bob* signierte (sortierte) Liste $L=\{E_1, E_2, ..., E_n\}$ zusammen mit dem Signaturwert $S=D_{Bob}(h(L))$ einen gemeinsamen Nicht-Existenz-Beweis für genau die durch S nicht erfassten Daten (Bitfolgen) darstellt. Nach dem Aushändigen von L und S an einen Verifizierer kann dieser mit Hilfe der Signaturprüfung entscheiden, ob ein vorliegendes Datum x in der ursprünglich signierten Liste tatsächlich nicht vorkommt. Hierfür muss er entscheiden können, welche Position i das Datum x innerhalb der korrekt sortierten L annehmen würde. Schlägt die Signaturprüfung der Liste $L'=\{E_1, ..., x, ..., E_n\}$ fehl, so beweist dies, dass L' und L nicht identisch sein können. Nach dem Signieren von L werden Versuche, um etwa vorzuspielen, dass ein bestimmtes Datum $E_i \in L$ nicht in der vorliegenden Liste enthalten ist, während der Prüfung von S entdeckt. Auch dies gilt natürlich unter der Annahme, dass die verwendete Hashfunktion h und die Signierfunktion D_{Bob} sicher sind. Es ist jedoch nachteilig, dass für diese Art Prüfung ebenfalls sämtliche n Listenelemente vorliegen müssen.

Die Erwartungen an Nicht-Existenz-Beweise, die ggf. anhand von einem Authentifizierungsbaum ausgestellt werden müssen, sind die folgenden: Werden durch die Wurzelsignatur S eines Authentifizierungsbaums B Daten (Bitfolgen) aus der Menge M gesichert, so gilt zu erreichen:
Es soll zu einem bestimmten durch den Baum nicht erfassten Datum $y \notin M$ ein Authentifizierungspfad $AP(y)$ erzeugt werden können, um eindeutig zu bestätigen, dass y durch die gültige Wurzelsignatur S von B nicht erfasst wurde. Dies impliziert, dass die mit Hilfe von $AP(y)$ überprüfbare Wurzelsignatur S definitiv keine gültige Signatur des Datums y sein darf.

Das heißt aber auch, dass Versuche, die Existenz eines durch den Baum signierten Datums $x \in M$ mit Hilfe eines fälschlich ausgestellten Authentifizierungspfads abzustreiten, zuverlässig entdeckt werden. Das bedeutet, dass das Abstreiten der Existenz des Blatts von *B,* welches den Hashwert *h(x)* von *x* enthält, entdeckt wird. Solche Authentifizierungspfade gelten als Nicht-Existenz-Beweise.

Der zweifellos einfachste Weg, sichere Existenz- bzw. Nicht-Existenz-Beweise im Sinne von Def. 5-1 respektive Def. 5-2 anhand von einem Merkleschen Authentifizierungsbaum (siehe Def. 2-5) zu erzeugen, ist die Aushändigung des kompletten in der Datenbank abgelegten Baums an die Verifizierer.
Ein Verifizierer kann in diesem Fall leicht entscheiden, ob ein (vorliegendes) Datum *D* durch die Wurzelsignatur *S* erfasst wurde. Hierfür muss er den Hashwert *h(D)* von *D* berechnen und prüfen, ob *h(D)* mit einem der in den Blättern des Baums gespeicherten Hashwerte identisch ist. Anschließend muss er den Wurzelhashwert H_W rekonstruieren und prüfen, ob die Wurzelsignatur *S* eine gültige Signatur von H_W ist.
Diese Vorgehensweise ist vor allem aus Kostengründen unpraktikabel. Um nämlich einen von *n* durch den (binären) Baum signierten Datensätzen auf diese Weise zu prüfen, müssen dem Verifizierer im schlimmsten Fall *2n-1* Knoten (Hashwerte) aus dem Baum übertragen werden. Die gleiche Nutzfunktionalität kann auch mit Hilfe einer signierten Liste mit *n* Hashwerten zu *n* Datensätzen erbracht werden, wobei nur *n* statt *2n-1* Hashwerte übertragen werden müssen.

In Def. 5-3 wird eine modifizierte Struktur von Authentifizierungsbäumen angegeben (vgl. Def. 2-5). Anhand von einem gemäß dieser Struktur erzeugten Baum können Authentifizierungspfade erzeugt werden, die je nach Situation als sichere Existenz- bzw. Nicht-Existenz-Beweise verwendet, d.h. geprüft werden können.

Def. 5-3: Modifizierter (binärer) Authentifizierungsbaum

knoten=„NL“ || s || knoten.wert
knoten.wert=h(„NL“ || s || (knoten.kind$_{links}$*).wert || (knoten.kind*$_{rechts}$*).wert)*
blatt=„L“ || s_{datum} *|| h(datum) || blatt.wert*
blatt.wert=h(„L“ || s_{datum} *|| h(datum))*

Diese Struktur kann als eine Art Verschmelzung eines hohlen (binären) Blattsuchbaums [Knu98] und des ursprünglichen binären Authentifizierungsbaums [Mer90] angesehen werden (vgl. [DG+01]). Durch den nicht leeren Baum können Daten, d.h. Bitfolgen, beliebiger Natur (semantischer Bedeutung) signiert werden.
Hierfür muss zunächst jedem durch den Baum erfassten Datum *datum* ein Suchschlüssel s_{datum} aus der linear geordneten endlichen Gesamtmenge *K* der Suchschlüssel eineindeutig zugeordnet werden. Dies impliziert, dass durch den Baum maximal $|K|$ unterschiedliche Datensätze signiert werden können. Für jeden internen Knoten *knoten* des Baums gilt, dass alle Suchschlüssel im linken (rechten) Teilbaum von *knoten* kleiner oder gleich (größer) als der im *knoten* abgelegte Schlüssel *s* sind. Von dem Baum wird im Allgemeinen keine Balanciertheit erwartet, wobei dies zur Stabilisierung der durchschnittlichen Suchkosten (Pfadlängen) von Vorteil wäre.
In einem jeden Blatt *blatt* des Baums werden ein Typbezeichner *„L“* (engl. *Leaf*, L), der Hashwert *h(datum)* eines bestimmten durch den Baum signierten Datums *datum* sowie der *datum* zugeordnete Suchschlüssel s_{datum} gespeichert. Außerdem wird im Blatt *blatt* der Hashwert *blatt.wert=h(„L“||*s_{datum}*||h(datum))* abgelegt.

Von der Hashfunktion *h* wird im Allgemeinen erwartet, dass diese eine kollisionsresistente Hashfunktion (siehe Def. 2-4) ist.
Ein jeder interne Knoten (einschließlich die Wurzel) *knoten* des Baums enthält hingegen den Typbezeichner *„NL“ (engl. Non-Leaf,* NL*),* einen Suchschlüssel *s* sowie einen Hashwert *knoten.wert*. In die Berechnung von *knoten.wert* fließen neben den in den beiden Kindknoten $knoten_{links}$ und $knoten_{rechts}$ abgelegten Hashwerten sowohl der im *knoten* abgelegte Suchschlüssel *s* als auch dessen Typbezeichner *„NL“* ein.

Die Suchschlüssel innerhalb der internen Knoten müssen eine eindeutige Suche nach dem Blatt mit den Angaben zu einem bestimmten Datum *datum* ermöglichen. Dies ist nämlich eine Voraussetzung, um anhand des Baums einen überprüfbaren Authentifizierungspfad *AP(datum)* zu *datum* zu generieren. Während einer von der Wurzel des Baums ausgehenden Suche *Find(datum)* bzw. $Find(s_{datum})$ muss das Blatt gefunden werden, welches den Suchschlüssel s_{datum} enthält, falls ein solches Blatt existiert.
Hierfür muss ausgehend von einem in einem Knoten *knoten* (Wurzel) abgelegten Suchschlüssel *s*, der jeweilig nächste, d.h. linke oder rechte Kindknoten (Blatt) von *knoten* eindeutig gefunden werden. Die Richtung des nächsten Abstiegsschrittes wird gemäß der jeweiligen über der Menge *S* der Suchschlüssel definierten Ordnungsrelation, d.h. durch die Vergleiche $s_{datum} \leq s$ respektive $s < s_{datum}$, bestimmt. Damit im Baum auch tatsächlich erfolgreich nach den erfassten Daten gesucht werden kann, muss dessen Erzeuger (Verwalter) stets gemäß der jeweilig geltenden Sortierregeln vorgehen. Das heißt, beim Einfügen eines neuen oder Löschen eines vorhandenen Blatts muss die durch die Suchschlüssel bestimmte Ordnung der Knoten erhalten bleiben.

Die Wurzelsignatur *S* des Baums kann der Baumerzeuger *Bob* (analog zu [Mer90]) über den Hashwert *knoten.wert* des Wurzelknotens durch $S=D_{Bob}(knoten.wert)$ berechnen. Im Einklang mit den Vorschlägen in Kap. 4.2 können in die Wurzelsignatur weitere Informationen *Infos*, wie z.B. der Name des verwendeten Signierverfahrens oder die Gültigkeitsperiode von *S*, einfließen (siehe auch Abb. 4-3*)*.
Somit wird die Wurzelsignatur im Allgemeinen durch $S=D_{Bob}(h(knoten.wert, Infos))$ berechnet, wobei *knoten.wert* der in der Wurzel des Baums abgelegte Hashwert ist. Da in die rekursive Berechnung des Wurzelhashwerts sämtliche durch den Baum erfassten Daten sowie die in den Knoten abgelegten Typbezeichner, Suchschlüssel und Hashwerte einfließen, werden diese Daten durch die Wurzelsignatur *S* des Baums geschützt. Das heißt, nachträgliche Manipulationen an diesen Angaben können während der Verifizierung von Authentifizierungspfaden entdeckt werden.

In Abb. 5-1 wird ein auf dieser Grundlage aufgebauter einfacher Authentifizierungsbaum gezeigt, der die Funktion eines binären CVT erfüllt (vgl. Abb. 4-2). Wie man sieht, werden hier die bereits aus Abb. 2-9 bekannten Attributszertifikate Z_2 und Z_3 von *Bob* mit den Seriennummern (*SerialNumber*) *2* respektive *3* durch den Baum erfasst. Hierbei wurde das Zertifikatsfeld *SerialNumber* als Sortierschlüssel für den Baum verwendet. Die in den einzelnen Knoten des Baums abgelegten Typbezeichner, Suchschlüssel und Hashwerte sind jeweils in einem separaten Kasten dargestellt.

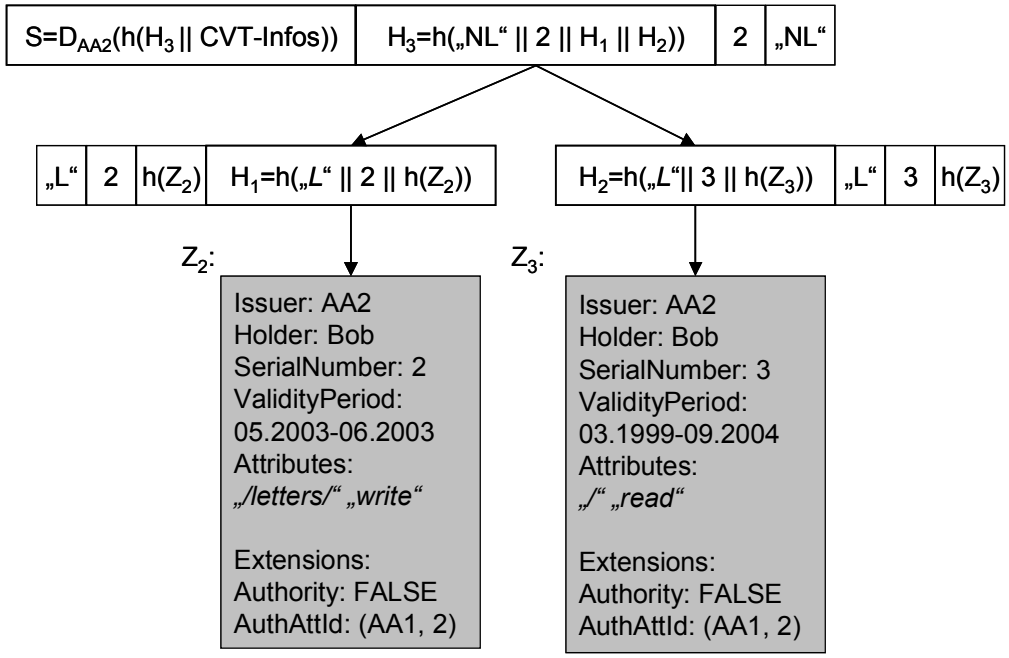


Abb. 5-1: CVT zur Erzeugung von Existenz- und Nicht-Existenz-Beweisen

Die Wurzelsignatur $S=D_{AA2}(h(H_3||CVT\text{-}Infos))$ dieses Baums wurde mit Hilfe des privaten Schlüssels der Zertifizierungsstelle (*Issuer*) AA_2 erzeugt. Wie man sieht, fließen in die Berechnung von S der Wurzelhashwert H_3 sowie die als *CVT-Infos* bezeichneten Daten (siehe auch Abb. 4-3) ein.
Angenommen ein Verifizierer hält AA_2 für vertrauenswürdig. Das heißt, er geht davon aus, dass AA_2 den Baum korrekt sortiert aufgebaut und stets korrekte Daten in die Berechnung der H_i Hashwerte sowie in die Berechnung der Wurzelsignatur S einbezogen hat. Ebenfalls nimmt er an, dass AA_2 im exklusiven Besitz seines privaten Schlüssels ist, mit welchem er S erzeugt hat. Ansonsten kann nämlich die Wurzelsignatur nicht sicher auf AA_2 zurückgeführt werden.
Dieser Baum kann nach seiner Erzeugung in einer Datenbank abgelegt werden, der Verifizierer (aus den in Kap. 4.3.2 genannten Gründen) nicht vertrauen. Diese Datenbank kann und muss nämlich ihre Antwort auf die Anfrage nach einem bestimmten Zertifikat auf die durch AA_2 erzeugte Wurzelsignatur zurückführen. Gelingt ihr dies, so wird ihre Antwort durch den Verifizierer akzeptiert, ansonsten wird sie abgelehnt.
Zunächst wird anhand des obigen Beispielsbaums die allgemeine Vorgehensweise zur Erzeugung und Prüfung von Existenz-Beweisen diskutiert.

5.1.1 Generierung und Prüfung von Existenz-Beweisen

Liegt einem Verifizierer ein bestimmtes Zertifikat Z_s mit der Seriennummer (*Serial-Number*) s bereits vor, so stellt er eine Anfrage an die Datenbank, ob Z_s durch den aktuellen Baum signiert wurde. In diesem Fall gilt Z_s als aktuell gültiges Zertifikat.
Wurden in der Datenbank mehrere von verschiedenen Stellen stammende CVTs abgelegt, so kann der Verifizierer neben s auch die jeweilige Stelle (hier AA_2) nennen, welche das vorliegende Zertifikat ausgestellt haben soll. Deren Namen kann er beispielsweise aus dem Zertifikatsfeld *Issuer* (siehe Kap. 2.2.4 und Abb. 2-5) entnehmen.
Nach Erhalt der Anfrage des Verifizierers muss die Datenbank nach dem passenden CVT anhand der diesbezüglichen Angaben in *CVT-Infos* (siehe Abb. 4-3 in Kap. 4.2) suchen. Falls es im Datenbestand keinen durch AA_2 signierten CVT gibt, wird eine entsprechende Fehlermeldung, wie z.B. *„CVT not found“*, generiert.

Welche Aktionen diese Meldung seitens des Verifizierers (bzw. der Zugriffskontrolle) erforderlich macht, hängt vom jeweiligen Anwendungskontext ab. Eine mögliche Strategie ist, dass sich der Verifizierer an eine andere Datenbankinstanz wendet, welche einen durch AA_2 ausgestellten aktuellen Baum speichert.

- **Generierung von Existenz-Beweisen:**

Wurde im Datenbestand der gewünschte CVT gefunden, so führt die Datenbank von der Wurzel ausgehend eine Suche *Find(s)* nach dem Blatt des Baums, welches den Suchschlüssel $s_{datum}=s$ sowie den möglicherweise korrekten Hashwert $h(Z_s)$ von Z_s enthält. Diese Suche wird gemäß der im Baum jeweilig geltenden Ordnung der Schlüssel (durch entsprechende Wertevergleiche) durchgeführt. Wird dabei das gesuchte Blatt tatsächlich gefunden, so kann die Datenbank einen Zertifizierungspfad $ZP(Z_s)$ zu Z_s ausstellen. $ZP(Z_s)$ ist ein Existenz-Beweis im Sinne von Def. 5-1. Ist die Suche ohne einen Treffer ausgegangen, so wird ein entsprechender Nicht-Existenz-Beweis (siehe dazu Kap. 5.1.2) generiert.
Die während der erfolgreichen Suche besuchten k ($k>0$) Knoten des Baums bilden einen Suchpfad. In Kenntnis des Suchpfads kann der Zertifizierungspfad $ZP(Z_s)$ zusammengestellt werden. Der Zertifizierungspfad $ZP(Z_s)$ enthält die Wurzelsignatur S (im Beispiel $S=D_{AA2}(h(H_3||CVT\text{-}Infos))$, den Hashpfad $HP(Z_s)$ und die im Wurzelknoten abgelegten Daten *CVT-Infos*.
Der Hashpfad $HP(Z_s)$ muss die Rekonstruktion des korrekten Wurzelhashwerts zwecks Signaturprüfung ermöglichen. Außerdem muss er die Prüfung unterstützen, ob der Suchpfad und damit $ZP(Z_s)$ durch eine korrekte Suche zustande kam. Hierfür enthält der Hashpfad $HP(Z_s)$ die Suchschlüssel und Typbezeichner aus den während der Suche besuchten insgesamt k Knoten. Außerdem enthält er die Hashwerte, welche in den (nicht besuchten) Geschwisterknoten der besuchten Knoten abgelegt wurden. Auf der Blattebene kann auf die Aushändigung des Hashwerts $h(Z_s)$ des gefragten Zertifikats Z_s verzichtet werden, wenn sich dieser während der Verifizierung (siehe weiter unten) aus dem vorliegenden Z_s berechnen lässt.
Wird die Datenbank beispielsweise nach der Existenz des von AA_2 (*Issuer*) ausgestellten Zertifikats Z_3 von *Bob* (*Holder*) gefragt, so kann sie anhand vom in Abb. 5-1 gezeigten CVT den Zertifizierungspfad $ZP(Z_3)$ als Existenzbeweis aushändigen:

$$ZP(Z_3)=[\ D_{AA2}(h(H_3||CVT\text{-}Infos)) \ ||\ CVT\text{-}Infos \ ||\ \{,,NL"||2||H_1\} \ ||\ \{,,L"||3||h(Z_3)\}]$$

Dieser Zertifizierungspfad enthält die im Wurzelknoten abgelegte Wurzelsignatur $S=D_{AA2}(h(H_3||CVT\text{-}Infos))$ sowie *CVT-Infos*. Außerdem enthält er den Hashpfad $HP(Z_3)=[\ \{,,NL"||2||H_1\} \ ||\ \{,,L"||3||h(Z_3)\}\ ]$. Die Generierung des Hashpfads $HP(Z_3)$ ist wie folgt: Wie man in Abb. 5-1 sieht, endet die Suche nach dem Zertifikat mit Suchschlüssel (*SerialNumber*) $s=3$ beim (rechten) Blatt des Baums, welches den Typbezeichner *„L“*, den Suchschlüssel *3*, den Hashwert $h(Z_3)$ des Zertifikats Z_3 und den Hashwert $h(,,L"||3||h(Z_3))$ speichert. Dem entsprechend wurde in den Hashpfad das Tripel $\{,,L"||3||h(Z_3)\}$ aufgenommen. Während der Suche nach diesem Blatt wurde die Wurzel des Baums besucht, die ein Vaterknoten des erreichten Blattes ist. Neben dem in der Wurzel gespeicherten Typbezeichner *„NL“* und Suchschlüssel *2* enthält das zweite Tripel $\{,,NL"||2||H_1\}$ in $HP(Z_3)$ auch den Hashwert H_1. Dieser ist genau der Hashwert H_1, welcher im nicht besuchten Geschwisterknoten (linkes Blatt) des schließlich gefundenen Blatts gespeichert wurde.

- **Verifizierung von Existenz-Beweisen:**

Bei der Verifizierung eines Existenz-Beweises bzw. Zertifizierungspfads wird geprüft, ob die im Beweis enthaltene Wurzelsignatur eine gültige Signatur des noch zu rekonstruierenden Wurzelhashwerts ist. Ist dies der Fall, so wird der Zertifizierungspfad als ein Beweis der Existenz des vorliegenden Zertifikats akzeptiert, andernfalls wird er abgelehnt. Das grundsätzliche Vorgehen während der Verifizierung wird anhand des oben behandelten Zertifizierungspfads $ZP(Z_3)$ exemplarisch demonstriert.
Nach Erhalt von $ZP(Z_3)$ wird von $HP(Z_3)=[\ \{„NL“||2||H_1\}\ ||\ \{„L“||3||h(Z_3)\}\ ]$ ausgehend der vermutlich korrekte Wurzelhashwert H_W' des Baums rekonstruiert. Dies erfolgt durch die (geschachtelte) Hashwertberechnung

$$H_W'=h(„NL“||2||H_1||\ h(„L“||3||h(Z_3))\ ).$$

Während der Berechnung von H_W' kann der Verifizierer die jeweilige Position der eingesetzten Hashwerte mit Hilfe der in $HP(Z_3)$ enthaltenen Suchschlüssel bestimmen. In unserem konkreten Beispielfall wird ausgehend vom Tripel $\{„NL“||2||H_1\}$ festgestellt, dass das gesuchte Zertifikat Z_3 mit dem Suchschlüssel $s=3$ im rechten Teilbaum der Wurzel liegen muss, da die Wurzel den Suchschlüssel *2* enthält. (Dies gilt natürlich unter der Annahme, dass AA_2 den Baum korrekt sortiert aufgebaut hat.) Deswegen kann der anhand des anderen Tripels berechnete Hashwert $h(„L“||3||h(Z_3))$ in die rechte Position bei der obigen Berechnung von H_W' eingesetzt werden.
Anschließend kann die in $ZP(Z_3)$ enthaltene Wurzelsignatur $S=D_{AA2}(h(H_3||CVT\text{-}Infos))$ überprüft werden. Hierfür wird zunächst mit Hilfe des öffentlichen Schlüssels von AA_2 durch

$$H_S=E_{AA2}(S)$$

der Hashwert H_S bestimmt, der von AA_2 ursprünglich signiert wurde. Um diese Berechnung durchzuführen, muss der öffentliche Schlüssel von AA_2 vorliegen und evtl. auf dessen Gültigkeit geprüft werden. Dabei können ein Schlüsselzertifikat beziehungsweise die Dienste einer PKI in Anspruch genommen werden.
Danach wird bestimmt, ob sich der ursprünglich signierte Hashwert H_S aus dem rekonstruierten Wurzelhashwert H_W' und aus den dem Verifizierer ebenfalls vorliegenden *CVT-Infos* berechnen lässt. Ist also

$$H_S=h(H_W',\ CVT\text{-}Infos),$$

so wird von der Korrektheit des rekonstruierten Wurzelhashwerts, d.h. im obigen Beispielfall von $H_W'=H_3$ ausgegangen. Da der Beispiel-Zertifizierungspfad $ZP(Z_3)$ den Hashwert $h(Z_3)$ enthält, muss noch überprüft werden, ob dieser mit dem Hashwert des vorliegenden Zertifikats Z_3 identisch ist. In diesem Fall kann der Zertifizierungspfad $ZP(Z_3)$ als ein gültiger Existenz-Beweis für das Zertifikat Z_3 akzeptiert werden.
Ein Existenz-Beweis muss laut Def. 5-1 sicherstellen, dass anhand eines manipulierten Baums die Existenz eines in Wirklichkeit nicht vorhandenen Datensatzes (Zertifikats) nicht vorgetäuscht werden kann.
Wird die Datenbank beispielsweise nach einem Existenz-Beweis zum Zertifikat Z_4 gefragt, welches durch den CVT in Abb. 5-1 nicht erfasst wurde, so muss ausgeschlossen bleiben, dass die Datenbank einem Verifizierer die Existenz dieses Zertifikats unentdeckt vortäuschen kann.

Ist sie in der Lage, einen gefälschten Zertifizierungspfad $ZP(Z_4)=\{$Wurzelsignatur S || *CVT-Infos* || Hashpfad $HP(Z_4)\}$ auszuhändigen, so kann sie einen Verifizierer in die Irre führen und dadurch das ungültige Zertifikat Z_4 für gültig anerkennen lassen.
Ein solch erfolgreicher Manipulationsversuch setzt aber voraus, dass entweder das jeweilig verwendete Signierverfahren oder/und die Hashfunktion gebrochen werden. Die Datenbank muss nämlich hierfür einen passend gefälschten Hashwert $H_S' \neq H_S$, mit einer vom Verifizierer später als gültig akzeptierten, d.h. scheinbar durch AA_2 getätigten Signatur versehen. Dies kann sie in Besitz des (gestohlenen oder gebrochenen) privaten Schlüssels von AA_2 erreichen.
Alternativ muss sie eine entsprechende Kollision der Hashfunktion h finden, sodass der während der Verifizierung rekonstruierte falsche Wurzelhashwert H_W'' zusammen mit den ausgehändigten (ggf. passend manipulierten) *CVT-Infos* genau den ursprünglich durch AA_2 signierten Hashwert, d.h. $H_S=h(H_W'', \textit{CVT-Infos})$ ergibt.

5.1.2 Generierung und Prüfung von Nicht-Existenz-Beweisen

Ausgehend von einem gemäß Def. 5-3 aufgebauten Authentifizierungsbaum (CVT) können Beweise generiert werden, um die Existenz eines bestimmten Datensatzes (Zertifikats) zu überprüfen, d.h. ob dieser durch den Baum erfasst wurde. Eine noch offene Frage ist wie das Fehlen, d.h. die Nicht-Existenz, eines im Baum (vergebens) gesuchten Datums zufrieden stellend nachgewiesen werden kann. Die grundsätzliche Vorgehensweise in diesem Fall wird mit Hilfe des Beispielbaums in Abb. 5-2 erklärt.

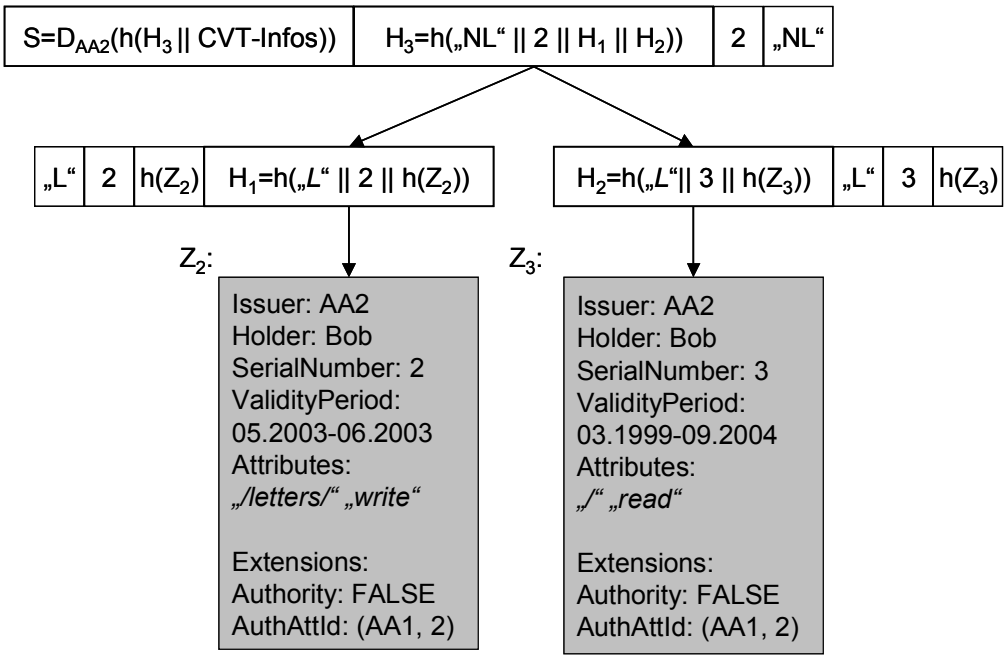


Abb. 5-2: CVT zur Erzeugung von Existenz- und Nicht-Existenz-Beweisen

- **Generierung von Nicht-Existenz-Beweisen:**

Angenommen die Datenbank wird nach dem Zertifizierungspfad des Zertifikats Z_s gefragt, welches (vermutlich) durch die Zertifizierungsstelle (*Issuer*) AA_2 ausgestellt wurde. Die Datenbank startet zuerst eine Suche *Find(s)* im durch AA_2 angelegten CVT. Hierbei stellt sie fest, dass das gefragte Zertifikat durch den CVT nicht erfasst wurde.

Genauer bedeutet dies, dass der durch AA_2 angelegte CVT kein Blatt enthält, welches den gefragten Suchschlüssel (*SerialNumber*) s bzw. den Hashwert $h(Z_s)$ von Z_s enthält. Unter der Annahme, dass AA_2 den eigens signierten Baum richtig sortiert aufgebaut hat, kann die erfolglose Suche nach dem Zertifikat mit Hilfe eines entsprechend erzeugten Zertifizierungspfads nachgewiesen werden.
Hierfür muss die Datenbank einen Zertifizierungspfad $ZP(Z_s)=ZP(Z_x)$ ($s \neq x$) aushändigen. Dieser wird anhand der Knoten des Baums ausgestellt, welche während der erfolglosen Suche *Find(s)* nach s besucht wurden. Diese Suche endet bei dem Blatt des Baums, welches statt des gefragten Suchschlüssels s den Suchschlüssel x und den Hashwert $h(Z_x)$ des durch den Baum erfassten Zertifikats $Z_x \neq Z_{id}$ enthält.
Die während dieser Suche besuchten k ($k>0$) Knoten des Baums bilden einen Suchpfad. In Kenntnis des Suchpfads kann der Nicht-Existenz-Beweis in Form des Zertifizierungspfads $ZP(Z_x)=ZP(Z_s)$ zusammengestellt werden. Dieser enthält die Wurzelsignatur S, den Hashpfad $HP(Z_x)$ sowie die in der Wurzel abgelegten Daten *CVT-Infos*.

Der Hashpfad $HP(Z_x)$ muss zum Einen die Rekonstruktion des korrekten Wurzelhashwerts zwecks Signaturprüfung ermöglichen. Zum Anderen muss er die Prüfung unterstützen, ob der Suchpfad und damit der Zertifizierungspfad durch eine korrekt ausgeführte und wirklich erfolglos geendete Suche nach dem gefragten Z_s zustande kam.
Hierfür enthält der Hashpfad $HP(Z_x)$ die Suchschlüssel und Typbezeichner aus den während der Suche besuchten k Knoten. Außerdem enthält er die Hashwerte, welche in den (nicht besuchten) Geschwisterknoten der jeweilig besuchten Knoten abgelegt wurden. Auf der Blattebene kann hier auf die Aushändigung des Hashwerts $h(Z_x)$ des Zertifikats Z_x normalerweise nicht verzichtet werden, da dieses dem Verifizierer nicht vorliegt.
Wird die Datenbank nach dem Zertifikat Z_4 mit dem Suchschlüssel (*SerialNumber*) $s=4$ gefragt, welches im Beispiel-CVT (siehe Abb. 5-2) nicht reflektiert wird, generiert sie den Zertifizierungspfad $ZP(Z_4)=ZP(Z_3)$:

$$ZP(Z_4)=[D_{AA2}(h(H_3||\text{CVT-Infos})) \,||\, \text{CVT-Infos} \,||\, \{,,NL"||2||H_1\} \,||\, \{,,L"||3||h(Z_3)\}]$$

Die gemäß der im Baum herrschenden Sortierregeln ausgeführte Suche *Find(s=4)* endete im rechten Blatt des Baums, welches den Typbezeichner *„L“,* den Suchschlüssel *3,* den Hashwert $h(Z_3)$ des Zertifikats Z_3 sowie den Hashwert $h(,,L"||3||h(Z_3))$ enthält. Dementsprechend enthält der Hashpfad $HP(Z_4)$ das Tripel $\{,,L"||3||h(Z_3)\}$.
Während der Suche wurde außer diesem Blatt nur die Wurzel des Baums besucht, welcher der Vaterknoten des schließlich erreichten Blattes ist. Neben dem hier gespeicherten Typbezeichner *„NL“* und Suchschlüssel *2* enthält das zweite Tripel $\{,,NL"||2||H_1\}$ in $HP(Z_4)$ auch den Hashwert H_1. Dieser Hashwert wurde im nicht besuchten Geschwisterknoten des Blattes gespeichert, bei dem die erfolglose Suche schließlich endete.

- **Verifizierung von Nicht-Existenz-Beweisen:**

Während der Überprüfung eines Nicht-Existenz-Beweises zum Zertifikat Z_s bzw. eines hierfür durch die Datenbank ausgehändigten Zertifizierungspfads $ZP(Z_x)=ZP(Z_s)$ wird zunächst geprüft, ob die im Beweis enthaltene Wurzelsignatur eine gültige Signatur des nun zu rekonstruierenden Wurzelhashwerts ist. Ist dies nicht der Fall, wird der Zertifizierungspfad abgelehnt.

Ansonsten wird geprüft, ob der im $ZP(Z_s)$ enthaltene Hashpfad $HP(Z_x)$ tatsächlich durch eine erfolglose Suche nach dem Zertifikat Z_s entstand. Besteht ein Verdacht auf Manipulationen seitens der Datenbank, so wird $ZP(Z_x)$ als Nicht-Existenz-Beweis abgelehnt. Das grundsätzliche Vorgehen bei der Verifizierung eines als Nicht-Existenz-Beweis ausgestellten Zertifizierungspfads wird anhand des oben behandelten Zertifizierungspfads $ZP(Z_3)=ZP(Z_4)$ vorgestellt.
Nach Erhalt von $ZP(Z_4)$ rekonstruiert der Verifizierer den (vermeintlichen) Wurzelhashwert H_W' des Baums. Hierfür wird der Hashpfad $HP(Z_4)=[\{„NL“||2||H_1\}\ ||\ \{„L“||3||h(Z_3)\}]$ verwendet. Diese Rekonstruktion erfolgt durch die Berechnung

$$H_W'=h(\ „NL“||2||H_1\ ||\ h(„L“||3||h(Z_3))\ ).$$

Während dieser Berechnung von H_W' kann der Verifizierer die jeweilige Position der eingesetzten Hashwerte mit Hilfe der in $HP(Z_4)$ enthaltenen Suchschlüssel bestimmen. Im konkreten Beispielfall wird anhand des dem Wurzelknoten entnommenen Tripels $\{„NL“||2||H_1\}$ festgestellt, dass das gesuchte Zertifikat Z_4 mit Suchschlüssel $s=4$ im rechten Teilbaum der Wurzel liegen muss. Diese enthält ja den Suchschlüssel *2* und es gilt $4>2$. Deswegen kann der anhand vom Tripel $\{„L“||3||h(Z_3)\}$ berechnete Hashwert $h(„L“||3||h(Z_3))$ in die rechte Position bei der obigen Berechnung von H_W' eingesetzt werden.
Danach wird die in $ZP(Z_4)$ enthaltene Wurzelsignatur $S=D_{AA2}(h(H_3||CVT\text{-}Infos))$ geprüft. Hierbei wird mit Hilfe des öffentlichen Schlüssels von AA_2 durch

$$H_S=E_{AA2}(S)$$

der Hashwert H_S bestimmt, welcher von AA_2 ursprünglich signiert wurde. Um diese Berechnung durchzuführen, muss der öffentliche Schlüssel von AA_2 vorliegen und geprüft werden. Als Nächstes wird bestimmt, ob sich der Hashwert H_S aus dem eben rekonstruierten Wurzelhashwert H_W' und aus vorliegenden *CVT-Infos* berechnen lässt. Ist

$$H_S=h(H_W',\ CVT\text{-}Infos),$$

so wird die Korrektheit des rekonstruierten Wurzelhashwerts, d.h. $H_W'=H_3$, (siehe Abb. 5-2) angenommen.

Um $ZP(Z_4)$ als einen gültigen Nicht-Existenz-Beweis für das Zertifikat Z_4 zu akzeptieren, muss noch geprüft werden, ob die Suche nach Z_4 tatsächlich erfolglos war. Da das gesuchte Zertifikat Z_4 sich im rechten Teilbaum der Wurzel befinden müsste (da $4>2$), kann ein Verifizierer $ZP(Z_4)$ als Nachweis für das Fehlen dieses Zertifikats halten.
Das Tripel $\{„L“||3||h(Z_3)\}$ im Hashpfad $HP(Z_4)$ deutet nämlich eindeutig darauf hin, dass das rechte Kind der Wurzel den Suchschlüssel *3* ($3>2$) enthält. Außerdem besagt der hier abgelegte Typbezeichner *„L“* eindeutig, dass es sich hier um ein Blatt und keinen internen Knoten des Baums handelt.
Folglich kann der Verifizierer davon ausgehen, dass die Suche nach dem gefragten Zertifikat Z_4 mit dem Suchschlüssel $s=4$ korrekt ausgeführt wurde und bei einem anderen als dem gesuchten Blatt des Baums endete.

Ohne die Möglichkeit zur eindeutigen Feststellung, ob ein im Zertifizierungspfad reflektierter Knoten ein Blatt oder ein interner Knoten des Baums ist, kann die Datenbank die Nicht-Existenz eines in Wirklichkeit vorhandenen Zertifikats vortäuschen. Sie kann nämlich einen „passend abgekürzten" Zertifizierungspfad generieren. Dabei kann sie die im Übrigen erwartete Einweg-Funktionalität der bei der Baumkonstruktion verwendeten Hashfunktion *h* ausnutzen: Laut Def. 2-4 kann das Urbild *x* aus dem Hashwert *h(x)* nicht erraten werden.
Ein Beispiel ist der im rechten Blattknoten des Baums in Abb. 5-2 zusammen mit dem Suchschlüssel *3* abgelegte Hashwert $h(Z_3)$. Es lässt sich nicht eindeutig feststellen, ob $h(Z_3)$ der Hashwert eines Zertifikats ist. Es kann nämlich in diesem Fall angenommen werden, dass der Knoten, aus welchem dieser Hashwert stammt, ein Blatt des Baums ist und folglich, dass die Suche tatsächlich hier endete.
Der Hashwert $h(Z_3)$ könnte jedoch durchaus auch der Hashwert *blatt.wert* aus dem (einzigen) rechten Kind *blatt* des in diesem Fall internen Knotens des Baums stammen. Dieser rechte Kindknoten könnte ggf. genau die benötigten Informationen zum eigentlich gesuchten Zertifikat Z_4 mit dem Suchschlüssel $s=4$ enthalten, da $4>3$ gilt.
Um eindeutig zu bestätigen, dass es sich hier um einen Hashwert eines Zertifikats handelt, müsste die Datenbank auf die Anfrage des Verifizierers das gar nicht angeforderte Zertifikat Z_3 aushändigen. Dies wäre jedoch in vielen Fällen problematisch:

- **Client-Push-Systeme:** In solchen Systemen liegen die Attributszertifikate unter Kontrolle der zertifizierten Anwender (*Holder*) und nicht etwa einer zentralen Datenbank. Folglich können durch die Datenbank keine nicht angefragte Zertifikate ausgehändigt werden, um die Nicht-Existenz eines angefragten Zertifikats nachzuweisen.

- **Server-Pull-Systeme:** In Server-Pull-Systemen hingegen, in welchen sowohl die Zertifikate als auch der zugehörige CVT in einer Datenbank gespeichert werden können, stiegen die Kosten der Kommunikation zwischen den Verifizierern und der Datenbank erheblich an: Hier müsste bei jeder Anfrage nach einem nicht existierenden Zertifikat ein anderes in der Datenbank abgelegtes Zertifikat an den Verifizierer gesendet werden.

Diese Probleme werden also durch die in den Baumknoten abgelegten Typbezeichner gelöst, die eine sichere Unterscheidung zwischen Blättern und internen Knoten des Baums ermöglichen. Da die Typbezeichner der Knoten in die Berechnung des Wurzelhashwerts und folglich in die Berechnung der Wurzelsignatur des Baums einfließen (siehe Def. 5-3), kann die Datenbank keine gekürzten Zertifizierungspfade etwa durch Änderung der Typbezeichner (unentdeckt) generieren. Diese Art Manipulationsversuch wäre spätestens bei der Prüfung der Wurzelsignatur erkennbar.

5.2 Vollständigkeitsbeweise

Mit Hilfe der modifizierten Authentifizierungsbäume (siehe Def. 5-3) kann eine unvertrauenswürdige Datenbank sicher nachweisen, ob ein bestimmtes Datum (Attributszertifikat), durch den Baum (CVT) erfasst wurde. Eventuelle Manipulationsversuche seitens der Datenbank zum Abstreiten oder Vorspielen der Existenz oder Nicht-Existenz eines Datensatzes werden während der Verifizierung anhand der Authentifizierungspfade zuverlässig entdeckt.

Im Folgenden wird eine Methodik vorgestellt, mit der die vorgeschlagene Baumstruktur zur Generierung und Überprüfung von so genannten Vollständigkeitsbeweisen angewandt werden kann.
Solche Beweise ermöglichen im Wesentlichen die gleichzeitige Überprüfung der Existenz oder Nicht-Existenz mehrerer bei einer Datenbank angefragter Datensätze.

Def. 5-4: Vollständigkeitsbeweis

Ein Vollständigkeitsbeweis ist ein auf eine vertraute Instanz zurückführbarer Nachweis. Er kann durch eine nicht vertrauenswürdige Instanz anhand einer gegebenen, durch die vertraute Instanz angelegten Datenstruktur auf eine bestimmte Suchanfrage hin generiert werden. Anhand eines Vollständigkeitsbeweises kann geprüft werden, ob tatsächlich alle das jeweilige Suchkriterium erfüllenden Daten in einer unveränderten Form vorliegen bzw. ob keine das Suchkriterium erfüllenden Daten verschwiegen wurden.

Bei den bisher behandelten Existenz- und Nicht-Existenz-Beweisen (siehe Def. 5-1 respektive Def. 5-2) handelt es sich um Spezialfälle von Vollständigkeitsbeweisen: Sie werden zur gesicherten Beantwortung von Datenbankanfragen verwendet, die sich auf genau einen durch den Baum (nicht) erfassten Datensatz beziehen.

Ein Vollständigkeitsbeweis im Sinne von Def. 5-4 lässt sich z.B. aufgrund einer signierten (sortierten oder unsortierten) Liste $L=\{E_1, E_2, ..., E_n\}$ und des zugehörigen durch *Bob* berechneten Signaturwerts $S=D_{Bob}(h(L))$ erzeugen. Die E_i Elemente der Liste können beliebige Bitfolgen sein. Angenommen L und S werden in einer Datenbank abgelegt. Ein Verifizierer, welcher *Bob* vertraut, kann auf seine Anfrage, welche k der n in L einsortierten Elemente ein bestimmtes Suchkriterium erfüllen, die komplette Liste L und auch S von der Datenbank bekommen.
Seien die E_i Listenelemente ganze Zahlen. In Kenntnis von L kann der Verifizierer entscheiden, für welche k Elemente der Liste $E_i>x$ gilt, indem er den entsprechenden Vergleich für alle n Listenelemente durchführt. Erweist sich die von der Datenbank erhaltene Signatur S als eine gültige Signatur der Liste L, so kann der Verifizierer sicher sein, dass ihm tatsächlich die vollständige durch *Bob* erzeugte Liste und somit alle das obige Suchkriterium erfüllenden Elemente vorliegen. Dies gilt natürlich unter der Annahme, dass die verwendete Hashfunktion h und die Signierfunktion D_{Bob} sicher sind. Folglich kann der Verifizierer davon ausgehen, dass die k „Treffer“ tatsächlich durch die von *Bob* ursprünglich angelegte Liste erfasst wurden, und dass keine außer diesen Elementen das gestellte Suchkriterium erfüllen.

Werden durch die Wurzelsignatur S eines modifizierten Authentifizierungsbaums (siehe Def. 5-3) Datensätze, d.h. Bitfolgen endlicher Länge, gesichert, so gilt zu erreichen: Es soll auf die Anfrage eines Verifizierers, ob der Baum k verschiedene Datensätze, wie z.B. Attributszertifikate $Z_1, Z_2, ..., Z_k$ mit den diesen zugeordneten Suchschlüsseln $s_1, s_2, ..., s_k$ signiert, ein Vollständigkeitsbeweis $VB(Z_1, Z_2, ..., Z_k)$ ausgehändigt werden, der die Existenz von $0 \leq e \leq k$ respektive die Nicht-Existenz von k-e Zertifikaten sicher überprüfbar macht.

Im Kontext der zertifikatsbasierten Zugriffskontrolle lassen sich verschiedene Anwendungsfälle für solche Vollständigkeitsbeweise identifizieren. Es ist nämlich häufig erforderlich, gleichzeitig mehrere Attributszertifikate zu überprüfen, um Zugriffe zuverlässig zu erlauben oder abzulehnen. Näher betrachtet werden im Folgenden die hiermit verknüpften Abläufe und Berechnungsschritte für zwei unterschiedliche Fälle:

- **Vollständigkeitsbeweise in Client-Push-Systemen:** In solchen Systemen (siehe Kap. 2.1.3) erreichen Attributszertifikate die Zugriffskontrolle bzw. den Verifizierer, indem der Zugreifende die zum Zugriff benötigten k Zertifikate $Z_{s1}, Z_{s2}, ..., Z_{sk}$ dem Verifizierer zusendet. Dabei wird angenommen, dass der Zugreifende nicht daran interessiert ist, sich etwa durch das beabsichtigte Zurückhalten der eigenen Zertifikaten auszusperren. Nach Erhalt der gesendeten Zertifikate kann sich der Verifizierer an die den CVT speichernde Datenbank wenden und dort einen Vollständigkeitsbeweis $VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$ anfordern. Dabei kann der Verifizierer eine Liste der relevanten Zertifikatsfelder an die Datenbank senden, welche die im CVT reflektierten Suchschlüssel $s_1, s_2, ..., s_k$ der vorliegenden Zertifikate bestimmen. Beispielsweise kann hier eine Liste der Seriennummern (*SerialNumber*) der gefragten Zertifikate verwendet werden, vorausgesetzt, dass dieses Feld als Suchschlüssel im jeweiligen CVT dient. Nach Erhalt des Vollständigkeitsbeweises $VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$ muss der Verifizierer prüfen, welche e $(0 \leq e \leq k)$ Zertifikate aktuell gültig, d.h. akzeptabel, sind.

- **Vollständigkeitsbeweise in Server-Pull-Systemen:** In Systemen, welche dem Server-Pull-Modell (siehe Kap. 2.1.3) folgen, erfüllen Vollständigkeitsbeweise im Grunde eine ähnliche Funktionalität. Der wesentliche Unterschied ist in diesem Fall, dass die Attributszertifikate in einer als nicht vertrauenswürdig eingestuften Zertifikatsdatenbank liegen: Diese kann bspw. durch das Zurückhalten von Zertifikaten Zugriffe einzelner Benutzer (zugunsten anderer) verhindern. Folglich muss ein Verifizierer, der diese Datenbank nach im gegebenen Zugriffsfall (potenziell) relevanten Zertifikaten $Z_{s1}, Z_{s2}, ..., Z_{sk}$ mit den Suchschlüsseln $s_1, s_2, ..., s_k$ fragt, überzeugt werden, dass die Datenbank die schließlich ausgehändigten e $(0 \leq e \leq k)$ Zertifikate nicht manipuliert hat. Außerdem muss er feststellen können, dass die übrigen k-e Zertifikate, deren Suchschlüssel das jeweilige Suchkriterium ansonsten erfüllen, tatsächlich nicht vorliegen, d.h. durch den Baum nicht signiert wurden.

Eine seitens der Datenbank triviale Vorgehensweise zur Erzeugung von Vollständigkeitsbeweisen ist die Aushändigung des kompletten Authentifizierungsbaums an einen anfragenden Verifizierer. In Kenntnis des gesamten Baums kann dieser leicht entscheiden, welche der auf eine Anfrage passenden (ihm evtl. bereits vorliegenden) Datensätze durch den Baum erfasst wurden.
Hierfür muss er von den Blättern ausgehend den Wurzelhashwert des Baums gemäß der Berechnungsformeln in Def. 5-3 rekonstruieren. Die anschließende Signaturprüfung gibt Auskunft darüber, ob der korrekte Baum vorliegt. Außerdem kann der Verifizierer anhand der in den Blättern des Baums abgelegten Suchschlüssel feststellen, ob ihm Datensätze fehlen, die aber das jeweilige Suchkriterium erfüllen. Der mit dieser Art Beweisführung verbundene Kommunikations- und Berechnungsaufwand ist proportional zur Anzahl der durch den Baum signierten Daten. Außerdem muss der komplette Vorgang nach jeder Aktualisierung des Baums wiederholt werden.

Im Folgenden werden kostengünstigere Vorgehensweisen zur Erzeugung und Prüfung von Vollständigkeitsbeweisen diskutiert, die anhand der in Def. 5-3 angegebenen Baumstruktur generiert werden können. In den behandelten Beispielen wird davon ausgegangen, dass der Baum ein Attributszertifikate signierender CVT ist.

5.2.1 Vollständigkeitsbeweise für Client-Push-Systeme

Liegen einem Verifizierer die durch den Zugreifenden übermittelten Zertifikate Z_{s1}, Z_{s2}, ..., Z_{sk} bereits vor, so kann die CVT-Datenbank auf Anfrage des Verifizierers den Vollständigkeitsbeweis $VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$ aushändigen. Dieser kann zu jedem der k gefragten Zertifikate Z_{si} jeweils einen separaten Zertifizierungspfad $ZP(Z_{si})$ enthalten. Jeder dieser Zertifizierungspfade weist die Existenz oder eben die Nicht-Existenz eines der k Zertifikate nach, je nachdem, ob die Suche nach dem Zertifikat Z_{si} (genauer nach dessen Suchschlüssel s_i bzw. Hashwert $h(Z_{si})$) erfolgreich war.
Nach der Überprüfung dieser einzelnen Zertifizierungspfade (siehe die Vorgänge in Kap. 5.1.1 respektive in Kap. 5.1.2) kann entschieden werden, welche e der k gefragten Zertifikate durch den aktuellen Baum erfasst, bzw. welche k-e Zertifikate nicht erfasst wurden. Außerdem bestätigt ein jeder der e Zertifizierungspfade, dass das jeweilige Zertifikat in einer unveränderten Form vorliegt. Versucht die Datenbank etwa die Existenz eines der k Zertifikate abzustreiten, so kann sie dies nur durch das Fälschen des zugehörigen Nicht-Existenz-Beweises erreichen.

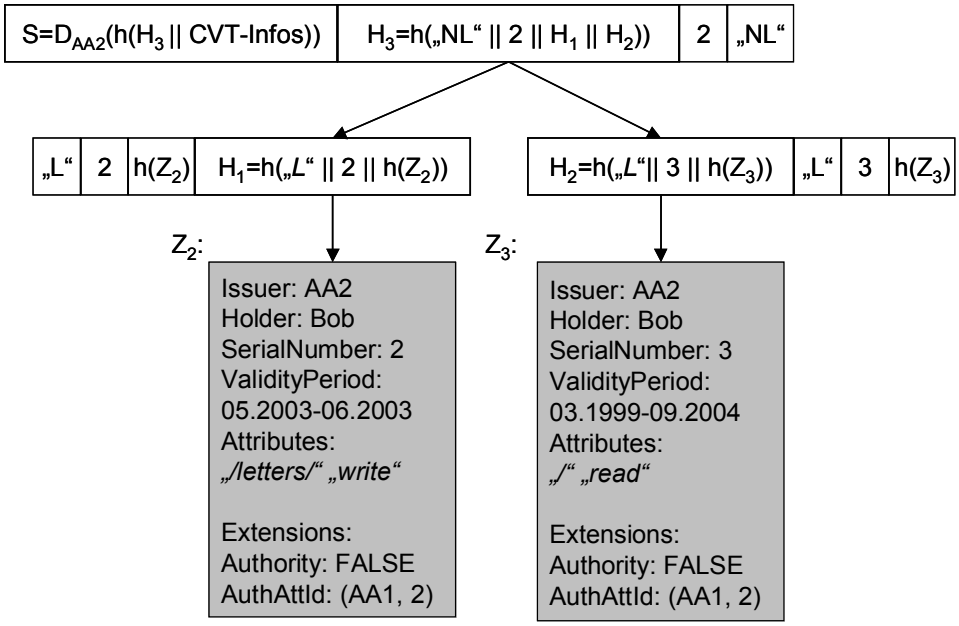


Abb. 5-3: CVT mit Zertifikaten von *Bob*

Mit Hilfe des Beispielbaums in Abb. 5-3 wurden durch die Zertifizierungsstelle AA_2 zwei Zertifikate von *Bob* signiert, die über ihre Seriennummern (*SerialNumber*) indiziert werden.
Angenommen *Bob* möchte nun auf das mit diesen Zertifikaten gesicherte Dateisystem zugreifen und dort die Datei */loveletter.htm* in das Verzeichnis */letters/* kopieren. Wie man anhand von Abb. 5-3 erkennen kann, werden hierfür sowohl Z_2 als auch Z_3 benötigt. Das Zertifikat Z_3 autorisiert nämlich *Bob* zum Lesen sämtlicher Verzeichnisse unterhalb der Wurzel *(„/“ „read“)* und Z_2 erlaubt ihm das Schreiben in das Zielverzeichnis (*„/letters/“ „write“)*.

Dementsprechend sendet er seine beiden Zertifikate an den Verifizierer, der nun deren aktuelle Gültigkeit anhand des CVT in Abb. 5-3 entscheiden kann.
Auf seine Anfrage an die CVT-Datenbank „*Bestätige die Gültigkeit von Zertifikaten mit SerialNumber 2 und 3*“ hin erhält der Verifizierer den Vollständigkeitsbeweis

$VB(Z_2, Z_3)=$
$\{$
$ZP(Z_2)=[D_{AA2}(h(H_3 \,||\, CVT\text{-}Infos)) \,||\, CVT\text{-}Infos \,||\, \{,,NL“ \,||\, 2||\, H_2\} \,||\, \{,,L“ \,||\, 2||\, h(Z_2)\}]$
$ZP(Z_3)=[D_{AA2}(h(H_3 \,||\, CVT\text{-}Infos)) \,||\, CVT\text{-}Infos \,||\, \{,,NL“ \,||\, 2||\, H_1\} \,||\, \{,,L“ \,||\, 3||\, h(Z_3)\}]$
$\}$.

Anhand von $VB(Z_2, Z_3)$ kann der Verifizierer entscheiden, welche der von *Bob* vorgelegten Zertifikate durch den aktuellen Baum erfasst, d.h. gültig, sind. Die einzelnen Zertifizierungspfade können gemäß der in Kap. 5.1.1 und 5.1.2 beschriebenen und hier nicht wiederholten Vorgehensweise geprüft werden.

Diese Vorgehensweise erweist sich jedoch als kostenintensiv, vor allem was die Übertragungskosten betrifft. Alle der in $VB(Z_{s1}, ..., Z_{sk})$ ausgehändigten k Zertifizierungspfade enthalten nämlich die Wurzelsignatur S sowie die zu deren Prüfung nützlichen Daten *CVT-Infos*. Es ist also grundsätzlich sinnvoll, diese Daten nur einmalig zu übertragen. Neben diesen Daten können auch einige der in den einzelnen Zertifizierungspfaden enthaltenen (*Typbezeichner, Suchschlüssel, Hashwert*)-Tripel mehrfach wiederholt vorkommen. Dies ist der Fall, wenn die Suchpfade, d.h. die während der Suchen nach den gefragten Zertifikaten besuchten Knoten, teilweise identisch sind. Hiervon können vor allem die Knoten im „oberen“ Bereich des Baums betroffen sein. Das heißt, auch solche Tripel sollten nur einmalig übertragen werden, um den Kommunikationsaufwand zu senken. In diesem Sinne kann die Datenbank statt des oben gezeigten, den folgenden Beweis an den Verifizierer senden:

$VB(Z_2, Z_3)'=$
$\{$
$Sig=D_{AA2}(h(H_3 \,||\, CVT\text{-}Infos));$
$Infos=CVT\text{-}Infos;$
$ZP(Z_2)=[Ref\#Sig \,||\, \{,,NL“ \,||\, 2||\, H_2\} \,||\, \{,,L“ \,||\, 2||\, h(Z_2)\} \,||\, Ref\#Infos];$
$ZP(Z_3)=[Ref\#Sig \,||\, \{,,NL“ \,||\, 2||\, H_1\} \,||\, \{,,L“ \,||\, 3||\, h(Z_3)\} \,||\, Ref\#Infos];$
$\}$

Hierbei bezeichnet *Ref#D* eine Referenz zum jeweiligen (ansonsten redundanten) Datensatz (Bitfolge) *D*. Anhand der eingefügten Referenzen kann der Verifizierer die beiden Zertifizierungspfade $ZP(Z_2)$ und $ZP(Z_3)$ einfach rekonstruieren, wobei die identisch ausfallenden Daten nur einmalig übertragen werden müssen. Des Weiteren genügt es, die für beide Pfade ebenfalls identisch ausfallende Wurzelsignatur $D_{AA2}(h(H_3||CVT\text{-}Infos))$ nur einmal zu prüfen.
Es ist noch zu beachten, dass $ZP(Z_2)$ und $ZP(Z_3)$ die Nicht-Existenz der durch den Baum in Abb. 5-3 nicht erfassten Z_i Zertifikate mit den Suchschlüsseln (*SerialNumber*) $s_i<2$ respektive $s_i>3$ bestätigen (vgl. Überprüfung von $ZP(Z_3)=ZP(Z_4)$ in Kap. 5.1.2.).

Folglich kann beispielsweise der Vollständigkeitsbeweis $VB(Z_1, Z_{42})$ identisch mit $VB(Z_2, Z_3)$ ausfallen. Das bedeutet wiederum, dass ein durch *Bob* vorgelegtes bereits abgelaufenes oder von AA_2 gar nie ausgestelltes Attributszertifikat mit dem Suchschlüssel (*SerialNumber*) $s<2$ oder $s>3$ durch den Verifizierer als ungültig erkannt wird.

5.2.2 Vollständigkeitsbeweise für Server-Pull-Systeme

In Abb. 5-4 wird ein binärer CVT gezeigt, welcher die Zertifikate von mehreren Benutzern *Alice, Bob, Fred* und *Sam* signiert. Aus Übersichtlichkeitsgründen wird angenommen, dass sowohl dieser CVT als auch die damit häufig aktualisierten Zertifikate in derselben Datenbank (zentralem Rechner) abgelegt wurden.
Seien die Attributszertifikate in dieser Datenbank als Elemente der Relation *AttrCerts* gespeichert. Die Attribute von *AttrCerts* seien durch die Felder des X.509-Formats *AttributeCertificateInfo* [ITU01] (siehe auch Abb. 2-5) definiert.
Für diesen CVT wurden durch die Zertifizierungsstelle AA_2 Suchschlüssel des Typs *{Holder||SerialNumber}* gewählt, wobei *Holder* (dargestellt als *Hold*) den Besitzer und *SerialNumber* (dargestellt als *SN*) die Seriennummer eines von AA_2 erstellten Zertifikats bezeichnen. Aufgrund der über diese Schlüssel definierten Ordnung bilden die Blätter des Baums, welche die einzelnen der k ($0 \le k \le |SerialNumber|$) Zertifikate eines bestimmten Benutzers (*Holder*) reflektieren, ein abgeschlossenes Intervall.

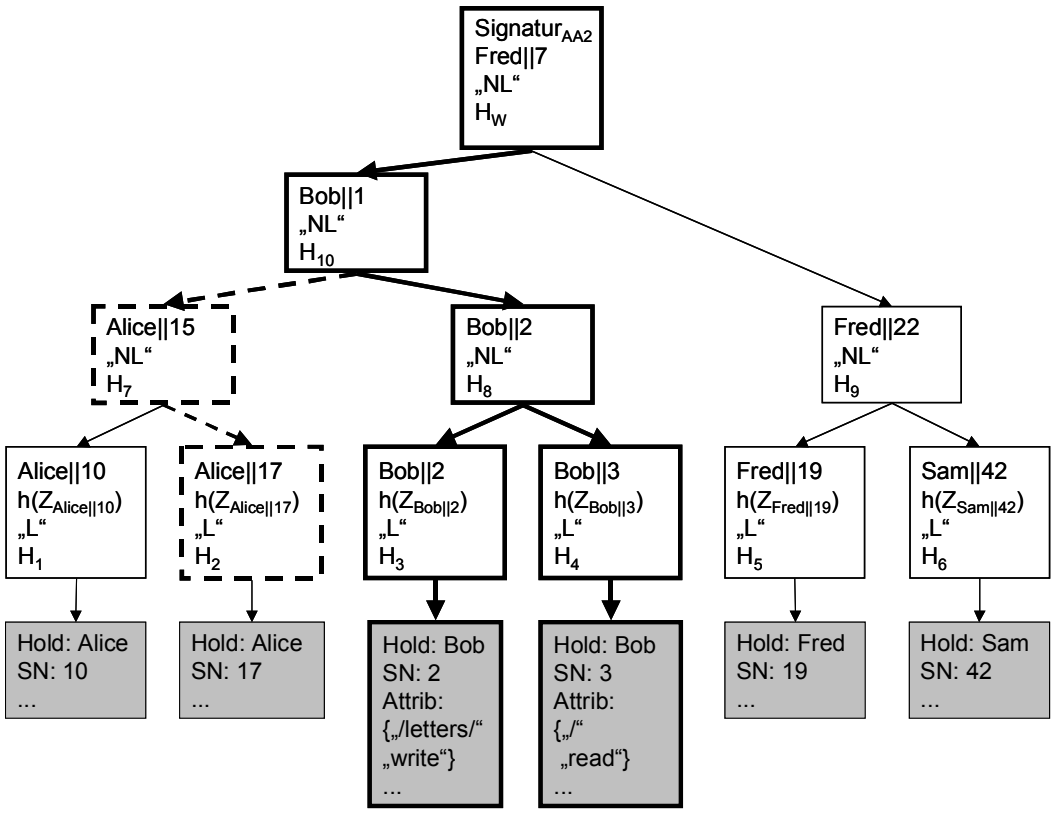


Abb. 5-4:Beispielbaum mit Zertifikaten mehrerer Benutzer

Angenommen *Bob* möchte auf das jeweilig gesicherte System zugreifen und dort die Datei */loveletter.htm* in das Verzeichnis */letters/* kopieren. Um ihm diesen Zugriff zu erlauben, werden sowohl $Z_{Bob||2}$ als auch $Z_{Bob||3}$ benötigt:
Wie man in Abb. 5-4 sieht, autorisiert das Zertifikat $Z_{Bob||3}$ *Bob* zum Lesen aller Inhalte unterhalb des Wurzelverzeichnisses *(„/“ „read“)* und $Z_{Bob||2}$ erlaubt ihm das Schreiben in das Zielverzeichnis (*„/letters/“ „write“)*.

Der zuständige Verifizierer, der AA_2 als vertrauenswürdig anerkennt, fordert nach Erhalt des Zugriffswunsches *(Bob, /loveletter.htm, copy(/letters/))* die Datenbank auf, die für *Bob* ausgestellten Attributszertifikate gemäß der Selektion $\sigma_{Holder="Bob"}(AttrCerts)$ auszuhändigen. Zu beachten ist, dass der Verifizierer zum Zeitpunkt des Zugriffs nur den Namen *Bob* des Zugreifenden und folglich das diesem Namen zugeordneten Zertifikatsfeld *Holder=Bob* kennen kann. Die Seriennummern der einzelnen passenden Zertifikate sind ihm zunächst unbekannt.
Um die obige Selektion durchzuführen, muss die Datenbank in der Relation *AttrCerts* nach allen Z_{si} Zertifikaten mit Suchschlüsseln

$$\{„Bob“||min(SerialNumber)\} \leq s_i \leq \{„Bob“||max(SerialNumber)\}$$

suchen. Die korrekte Antwort der Datenbank sollte im gewählten Beispielfall genau zwei Attributszertifikate, nämlich $Z_{Bob||2}$ und $Z_{Bob||3}$ enthalten, da genau diese das Suchkriterium erfüllen. Nach Erhalt der durch die Datenbank jeweilig ausgehändigten Zertifikate kann sich der Verifizierer erneut an die Datenbank wenden und einen Vollständigkeitsbeweis $VB(Holder="Bob")$ bzw. $VB(Z_{Bob||0}, Z_{Bob||1}, ..., Z_{Bob||max(Serialnumber)})$ ausstellen lassen. Dieser Beweis muss die vollständige Beantwortung der Datenbankanfrage zuverlässig bestätigen, das heißt, dass

1. die beiden ausgehändigten Zertifikate tatsächlich gültige, d.h. durch den Baum signierte Zertifikate sind und, dass
2. der Benutzer *Bob* keine weiteren (aktuell) gültigen Zertifikate besitzt, welche ihm den oben gewünschten Zugriff etwa verbieten würden.

Alternativ kann die Datenbank die Zertifikate sowie den zugehörigen Vollständigkeitsbeweis in derselben Antwortnachricht an den Verifizierer senden. Dies setzt voraus (wie oben angenommen), dass dieselbe Datenbank die Zertifikate als auch den diese sichernden CVT speichert.

Im Folgenden wird gezeigt, wie sich anhand von modifizierten Authentifizierungsbäumen Vollständigkeitsbeweise im Speziellen zur Absicherung von Intervallsuchen generieren und prüfen lassen. Konkret bedeutet dies, den Nachweis zu führen, welche der k im Baum durch ihre zugeordneten Suchschlüssel $s_1, s_2, ..., s_k$ potenziell reflektierten Datensätze (in unserem Fall Zertifikate) $Z_{s1}, Z_{s2}, ..., Z_{sk}$ tatsächlich durch den Baum erfasst wurden, wenn diese Suchschlüssel ins geordnete Intervall $[s_1, s_k]$ fallen. Hierbei kann $s_1=min(K)$ oder/und $s_k=max(K)$ gelten, wobei K die Menge der Suchschlüssel des Baums bezeichnet.

- **Generierung von Vollständigkeitsbeweisen für Intervallsuchen:**

Ein solcher Vollständigkeitsbeweis enthält die Wurzelsignatur S des Baums sowie die bei deren Prüfung verwendeten Daten *CVT-Infos*. Außerdem enthält er Typbezeichner, Suchschlüssel und Hashwerte aus verschiedenen Knoten des Baums. Um die jeweilig benötigten Daten auszuwählen, wird zunächst eine Suche nach den Blättern des Baums durchgeführt, die im betrachteten Fall jeweils einen Suchschlüssel s_i aus dem gesuchten Intervall $[s_1, s_k]$ enthalten.

Der Beweis enthält entsprechend die Tripel *(„L“||s_i||$h(Z_{si})$}*, welche den schließlich gefundenen Blättern zu entnehmen sind. Auf die Zusendung der Hashwerte $h(Z_{si})$ der durch die Datenbank ausgehändigten Zertifikate Z_{si} kann grundsätzlich verzichtet werden.
Diese lassen sich nämlich aus den jeweiligen Zertifikaten berechnen (siehe Verifizierung unten). Des Weiteren enthält der Vollständigkeitsbeweis Daten zu verschiedenen während der Suche besuchten respektive nicht besuchten internen Knoten des Baums.
Anhand dieser Daten muss zum Einen die Rekonstruktion des korrekten Wurzelhashwerts zwecks Signaturprüfung möglich sein. Zum Anderen muss aus diesen Daten klar hervorgehen, ob die Datenbank eine korrekte Suche nach den passenden Zertifikaten durchgeführt, d.h. alle passenden Blätter in ihrer Antwort reflektiert hat.
Hierfür enthält der Vollständigkeitsbeweis die Suchschlüssel und Typbezeichner aus den während der Suche besuchten *k* Knoten des Baums. Neben diesen Daten enthält er die Hashwerte, welche in den (nicht besuchten) Geschwisterknoten der besuchten Knoten abgelegt wurden. Diese Angaben werden in einzelnen Tupeln zusammengefasst.
Eine Ausnahme hiervon bilden interne Knoten, deren sämtliche (im Fall eines Binärbaums beide) Kinder besucht wurden. In diesem Fall reicht es, den Typbezeichner und den Suchschlüssel aus diesem Knoten auszuhändigen.
Der Beweis muss auch eindeutige Hinweise über mögliche „Sackgassen“ enthalten, die während der Suche nach den schließlich gefundenen Blättern begangen wurden. Das heißt, die während der Suche betretenen Knoten, deren Teilbäume schließlich keine passenden Blätter enthielten, müssen durch die in diesen Knoten abgelegten Typbezeichner und Suchschlüssel reflektiert werden. Solche Sackgassen enden grundsätzlich in Blättern des Baums. Hierbei gilt zu beachten, dass im Fall einer gänzlich erfolglosen Suche der Vollständigkeitsbeweis ausschließlich Sackgassen enthält, und belegt damit die Erfolglosigkeit der Suche nach den jeweilig gefragten Zertifikaten.

Wird beispielsweise nach sämtlichen für den Benutzer *(Holder) Bob* ausgestellten Attributszertifikaten gefragt, so kann die Datenbank anhand vom CVT in Abb. 5-4 den (minimalen) Vollständigkeitsbeweis

VB(Holder=“Bob“)=VB($Z_{Bob||0}$, $Z_{Bob||1}$, …., $Z_{Bob||max(Serialnumber)}$)
{
Sig: D_{AA2}(h(H_w, CVT-Infos));
Infos: CVT-Infos;
{„L“, Bob||2}; {„L“, Bob||3}; {„NL“, Bob||2}; {„L“, Alice||17, h($Z_{Alice||17}$)};
{„NL“, Alice||15, H_1}; {„NL“, Bob||1}; {„NL“, Fred||7, H_9}
}

erstellen und diesen dem Verifizierer zusenden. Wie man sieht, enthält *VB(Holder=“Bob“)* die Wurzelsignatur *Sig=D_{AA2}(h(H_w, CVT-Infos))* und auch *CVT-Infos*, die zur Prüfung der Wurzelsignatur benötigt werden.
Während der Suche nach den Zertifikaten von *Bob* anhand des verwendeten Suchschlüssels vom Typ {*Holder*||*SerialNumber*} wurden die in Abb. 5-4 fett gedruckten Knoten des CVT besucht.
Da die Suche schließlich zwei Blätter des Baums als Ergebnis hatte, enthält *VB(Holder=“Bob“)* die beiden Tupeln *{„L“, Bob||2}* und *{„L“, Bob||3}*. Die in diesen Blättern abgelegten Hashwerte H_3 und H_4 der Zertifikate $Z_{Bob||2}$ respektive $Z_{Bob||3}$ wurden nicht ausgeliefert, da die beiden Zertifikate dem Verifizierer bereits vorliegen.

Diese beiden Blätter des Baums wurden über deren gemeinsamen Vaterknoten betreten, welcher den Suchschlüssel *Bob||2* enthält. Da beide Kinder dieses Knotens durch den Beweis erfasst wurden, reicht hier die Aushändigung des Tupels *{„NL“, Bob||2}*. Der in diesem Knoten abgelegte Hashwert H_8 kann nämlich mit Hilfe dieser Angaben während der Verifizierung eindeutig bestimmt werden (siehe unten).
Der Vater dieses Knotens enthält den Suchschlüssel *Bob||1*. An dieser Stelle wurde zunächst die linke Abzweigung durchsucht (siehe die fett und gestrichelt gedruckten Knoten in Abb. 5-4). Der Schlüssel *Bob||1* ist nämlich ein Hinweis, dass der linke Teilbaum dieses Knotens evtl. für *Bob* ausgestellte Zertifikate, nämlich $Z_{Bob||0}$ oder/und $Z_{Bob||1}$ enthalten kann, da $Bob||0 \leq Bob||1$ und $Bob||1 \leq Bob||1$ gilt.
Wie man aber in Abb. 5-4 sieht, erwies sich dieser Zweig als eine Sackgasse: Das linke Kind des Knotens enthält den Suchschlüssel *Alice||15*. Das heißt, dass die gesuchten Zertifikate $Z_{Bob||0}$ oder/und $Z_{Bob||1}$ im rechten Teilbaum jenes Knotens sein müssten, da $Alice||15 < Bob||0$ und $Alice||15 < Bob||1$ gilt. Wie man aber sieht, besteht dieser rechte Teilbaum aus einem einzigen Blatt, welches aber keines der erhofften Zertifikate von *Bob,* sondern das Zertifikat $Z_{Alice||17}$ von *Alice* mit Suchschlüssel *Alice||17* referenziert. Dementsprechend wurde die Sackgasse durch *{„NL“, Alice||15,* H_1*}* und *{„L“, Alice||17,* $h(Z_{Alice||17})$*}* im Beweis *VB(Holder=“Bob“)* reflektiert.
Der Beweis enthält zum Schluss noch Angaben zum Wurzelknoten des Baums, welche den Suchschlüssel *Fred||7* enthält.
Da sich alle Zertifikate von *Bob* im linken Teilbaum der Wurzel befinden müssen (da $Bob||max(Serialnumber) < Fred||min(Serialnumber)$ gilt), war an dieser Stelle kein Abstieg in den rechten Teilbaum der Wurzel notwendig. Folglich enthält der Beweis *{„NL“, Fred||7,* H_9*}*.

- **Verifizierung von Vollständigkeitsbeweisen für Intervallsuchen:**

Die Vorgehensweise zur Verifizierung eines Vollständigkeitsbeweises wird anhand des oben erzeugten Vollständigkeitsbeweises vorgestellt. Erhält ein Verifizierer auf seine Anfrage nach allen für *Bob* ausgestellten Zertifikaten den oben behandelten Vollständigkeitsbeweis

$VB(Holder=“Bob“) = VB(Z_{Bob||0}, Z_{Bob||1}, ..., Z_{Bob||max(Serialnumber)})$
{
$Sig: D_{AA2}(h(H_w, CVT\text{-}Infos));$
Infos: CVT-Infos;
{„L“, Bob||2}; {„L“, Bob||3}; {„NL“, Bob||2}; {„L“, Alice||17, $h(Z_{Alice||17})$*};*
{„NL“, Alice||15, H_1*}; {„NL“, Bob||1}; {„NL“, Fred||7,* H_9*}*
},

so muss er diesen überprüfen. Diese Überprüfung besteht aus zwei wesentlichen Schritten, nämlich

1. Rekonstruktion des Wurzelhashwerts H_W' des Baums,

2. Prüfung, ob H_W' identisch mit dem ursprünglichen durch den Erzeuger AA_2 errechneten und signierten Wurzelhashwert H_W des Baums ist.

Während der Rekonstruktion des Wurzelhashwerts wird geprüft, ob die Datenbank die Existenz eventuell durch den Baum erfasster aber nicht ausgehändigter Zertifikate abstreiten wollte. Im konkreten Beispielfall gilt zu entscheiden, ob der Baum noch weitere für *Bob* ausgestellte Z_{si} Zertifikate mit Suchschlüssel $Bob||0 \leq s_i < Bob||2$ und/oder $Bob||3 < s_i \leq Bob||max(SerialNumber)$ signiert.

Die Verifizierung von *VB(Holder="Bob")* kann ausgehend von den Informationen zu den Blättern des Baums gestartet werden (siehe Abb. 5-4).
Da die beiden Zertifikate $Z_{Bob||2}$ und $Z_{Bob||3}$ vorliegen, können zunächst die Hashwerte $H_3=h(„L", Bob||2, h(Z_{Bob||2}))$ und $H_4=h(„L", Bob||3, h(Z_{Bob||3}))$ berechnet werden. Hierfür werden die ersten beiden Tupel *{„L", Bob||2}* respektive *{„L", Bob||3}* aus dem Beweis *VB(Holder="Bob")* verwendet.
In Kenntnis von H_3 und H_4 kann anhand des Tupels *{„NL", Bob||2}* der Hashwert $H_8=h(„NL", Bob||2, H_3, H_4)$ bestimmt werden. Die Positionierung der beiden Hashwerte H_3 (links) und H_4 (rechts) ist eindeutig, da für die zugehörigen Suchschlüssel $Bob||2 \leq Bob||2$ respektive $Bob||2 < Bob||3$ gilt.
Um den Hashwert $H_{10}=h(„NL", Bob||1, H_7, H_8)$ im nächsthöheren Knoten mit dem Suchschlüssel *Bob||1* zu bestimmen, muss zunächst H_7 ermittelt werden. Anhand vom Suchschlüssel *Bob||1* „merkt" der Verifizierer, dass der linke Teilbaum dieses Knotens evtl. noch ihm nicht vorliegende Zertifikate von *Bob* enthalten kann. In diesem Sinne versucht er den Abstieg (oben als „Sackgasse" bezeichnet) der Datenbank in diese Richtung zu rekonstruieren.
Hierbei stellt er anhand von *{„NL", Alice||15, H_1}* zunächst fest, dass der Knoten mit dem Suchschlüssel *Alice||15* noch passende Kindknoten in ihrem rechten Teilbaum haben kann, da *Bob>Alice* gilt. Erst bei der Bearbeitung von $\{„L", Alice||17, h(Z_{Alice||17})\}$ kann der Verifizierer „einsehen", dass es sich hier um eine Sackgasse handelte. Hiervon überzeugt ihn der Typbezeichner *„L"*, welcher bescheinigt, dass die Suche in diesem Zweig in einem Blatt endete und erfolglos war. An dieser Stelle „weiß" also der Verifizierer, dass keine Z_{si} Zertifikate mit $Bob||min(SerialNumber) < s_i < Bob||2$ durch den korrekt sortierten Baum erfasst wurden.
Nach diesem Schritt berechnet der Verifizierer den Hashwert $H_7=h(„NL", Alice||15, H_2, H_1)$, wobei $H_2=h(„L", Alice||17, h(Z_{Alice||17}))$ gilt. Hierfür werden die beiden Tripel *{„NL", Alice||15, H_1}* und $\{„L", Alice||17, h(Z_{Alice||17})\}$ aus dem Beweis *VB(Holder="Bob")* benötigt.
Anschließend kann der Hashwert $H_{10}=h(„NL", Bob||1, H_7, H_8)$ bestimmt werden. Aufgrund von H_{10} sowie des Tripels *(„NL", Fred||7, H_9)* aus dem Beweis kann schließlich der (vermeintliche) Wurzelhashwert $H_W'=h(„NL", Fred||7, H_{10}, H_9)$ des Baums berechnet werden. Der im Wurzelknoten abgelegte Suchschlüssel *Fred||7* überzeugt den Verifizierer, dass im rechten Teilbaum der Wurzel keine für *Bob* ausgestellten, evtl. zurückgehaltenen Zertifikate Z_{si} mit $Bob||2 < s_i < Bob||max(SerialNumber)$ reflektiert werden können, da $Fred||min(Serialnumber) > Bob||max(Serialnumber)$ gilt.
In einem finalen Schritt muss noch geprüft werden, ob der ermittelte Wurzelhashwert H_W' korrekt ist.
Hierfür wird zunächst die Signatur $S=D_{AA2}(h(H_w, CVT\text{-}Infos))$ des Baums verifiziert. Gilt dabei $E_{AA2}(S)=h(H_W', CVT\text{-}Infos)$, so kann der Verifizierer von der Korrektheit und Authentizität des ermittelten Wurzelhashwerts H_W' ausgehen und den Vollständigkeitsbeweis *VB(Holder="Bob")* akzeptieren. Andernfalls wird der Beweis wegen Verdachts auf Fälschung abgelehnt.

5.3 Realisierung auf Basis eines B^+-Baums

Die Datenstruktur mit der Bezeichnung *Improved Certification Verification Tree* (I-CVT) wurde konstruiert, um die Vorteile der in diesem Kapitel diskutierten Thesen innerhalb von Zertifikats-Datenbanken auszunutzen.

Ein I-CVT ist ein verbesserter CVT, welcher von einer vertrauenswürdigen Instanz, in der Regel eine Zertifizierungsstelle, erzeugt bzw. signiert wird. Er ermöglicht einer nicht vertrauenswürdigen Speicherinstanz, wie z.B. einer Zertifikatsdatenbank, die Generierung von Existenz-, Nicht-Existenz- und Vollständigkeitsbeweisen, um nachzuweisen, dass diese Instanz tatsächlich im Sinne des Baumerzeugers handelt. Durch die Verifizierung dieser Beweise kann nämlich eindeutig entschieden werden, welche der jeweilig gefragten Zertifikate durch den Baum erfasst wurden, d.h. als aktuell gültig gelten.
Als Basis für die I-CVTs wurden so genannte B-Bäume, genauer gesagt B^+-Bäume gewählt. B-Bäume [BM70] [BM72] [Knu98] werden in den meisten heutigen (kommerziellen) Datenbankmanagementsystemen zur Indizierung von Daten verwendet. Sie ermöglichen es, mit vergleichbar einfachen Algorithmen in großen Datenbeständen effizient zu suchen und/oder Änderungen am Datenbestand vorzunehmen. Die Effizienz bedeutet hier vor allem die Reduzierung der jeweilig notwendigen Anzahl von Blockzugriffen in Sekundärspeichern (Festplatten) innerhalb eines Datenbanksystems.
Ein B-Baum ist im Allgemeinen ein Suchbaum, in dessen internen Knoten sowie Blättern die durch den Baum referenzierten Datensätze (Nutzdaten) gespeichert werden können. Da weder die Authentifizierungsbäume [Mer90] noch die hiervon abgeleiteten CVTs [GGM00] eine Speicherung der Nutzdaten (Zertifikate) innerhalb des Baums vorsehen, erscheint die Verwendung so genannter B^+-Bäume sinnvoll [Ebi01] [NEA02] [Oet02]. Diese B-Baum-Variante zeichnet sich dadurch aus, dass die Knoten selbst keine Nutzdaten beinhalten [Com79]. Die Nutzdaten werden vielmehr durch die in den Blättern des Baums enthaltenen Zeiger referenziert und dadurch auffindbar. Außerdem ermöglichen B^+-Bäume, die referenzierten Daten in einer sequentiellen Reihenfolge effizient zu besuchen. Ein jedes Blatt enthält hierfür Zeiger auf seine unmittelbaren Nachbarblätter, falls es diese gibt. Aufgrund dieser Eigenschaft sind B^+-Bäume beispielsweise für die Intervallsuche besonders gut geeignet [Elm00].
Ein B^+-Baum der Ordnung m ($m>2$) wird durch die folgende Definition angegeben:

Def. 5-5: B^+-Baum der Ordnung *m*

- Die internen Knoten des Baums haben mindestens $\lceil m/2 \rceil$ und höchstens m Kindknoten. Die Wurzel hat mindestens zwei Kindknoten, außer sie ist ein Blatt.
- Interne Knoten mit $k \leq m$ Kindknoten enthalten $k-1$ Suchschlüssel sowie k Zeiger auf die Kindknoten. Die s_i Suchschlüssel eines Knotens werden aufsteigend sortiert, so dass $s_1 < s_2 < \ldots < s_{m-1}$ gilt. Als Suchschlüssel für den Baum eignen sich deshalb Datentypen, über welche die ordnungshaltende Abbildung „<“ definiert ist, wie z.B. ganze Zahlen. Der Zeiger P_i eines Knotens verweist dabei auf den Teilbaum, welcher alle Suchschlüssel enthält, die gemäß der Ordnung größer als s_{i-1} und kleiner oder gleich s_i sind. Falls $i=1$ ($i=m$), so zeigt P_i auf den Teilbaum, der Suchschlüssel enthält, die alle kleiner oder gleich (größer) als s_i sind.
- Die Blätter des Baums enthalten mindestens $\lceil m/2 \rceil - 1$ und höchstens $m-1$ Suchschlüssel, sowie genau so viele Zeiger auf die (nicht im Baum abgelegten) Nutzdaten.

- Ein B^+-Baum muss stets balanciert sein, so dass alle Suchpfade von der Wurzel zu den Blättern stets die gleiche Länge haben. Dies impliziert, dass während der Suche nach den einzelnen Datensätzen die gleiche Anzahl von Knoten besucht werden muss.

Die Operationen an einem B^+-Baum werden hier kurz diskutiert, um die aus unserer Sicht relevanten Kosten abzuschätzen. Details sind in [Com79] und [Knu98] zu finden.

- **Suche:** Zunächst wird der Schlüssel s_D des gefragten Datensatzes D mit den in der Wurzel gespeicherten s_i ($1 \le i \le m-1$) Suchschlüsseln verglichen. Über den entsprechenden Zeiger P_k wird der k-te Kindknoten ausgewählt, wobei $s_{k-1} < s_D \le s_k$ gilt. Diese Vorgehensweise wird auf jeder Ebene des Baums wiederholt, bis zur Erreichung eines Blattes. Falls gesuchtes D mit dem Schlüssel s_D durch den Baum indiziert wird, ist der Zeiger auf D in diesem Blatt zu finden.

- **Einfügen:** Das Einfügen eines neuen Verweises auf Datum D in den Baum erfolgt in zwei Schritten: Zunächst wird das Blatt gesucht (siehe oben), in welches ein neuer Suchschlüssel s_D und ein Zeiger auf D gespeichert werden. Nach dem Einfügen dieser beiden Daten wird dafür gesorgt, dass die evtl. verletzten B^+-Baum-Bedingungen (siehe Def. 5-5) wieder erfüllt werden. Die wichtigste Operation hierbei ist die Aufspaltung (engl. *Splitting*) von Knoten, die durch das Einfügen von neuen Schlüsseln „vollgelaufen" sind.

- **Löschen:** Beim Löschen von D wird zunächst eine Suche durchgeführt, um das entsprechende Blatt zu finden. Dann werden der Suchschlüssel s_D von D sowie der Zeiger auf D gelöscht. Es muss nun sichergestellt werden, dass in jedem Knoten des Baums mindestens $\lceil m/2 \rceil - 1$ Suchschlüssel verbleiben (gemäß Def. 5-5). Dabei ist evtl. eine Neuverteilung (engl. *Redistribution*) der Schlüssel beziehungsweise die Verschmelzung (engl. *Concatenation*) von Knoten erforderlich. Falls es dabei schließlich zu einer Verschmelzung der Kinder der Wurzel kommt, wird der Baum um eine Ebene kleiner.

- **Kostenbetrachtungen:**

Ein B^+-Baum bleibt laut Def. 5-5 nach jedem Einfüge- oder Löschvorgang balanciert. Dadurch bleiben also die Pfadlängen, d.h. die Anzahl der während einer Suche besuchten Knoten, gleich und entsprechen der Höhe h des Baums. Diese Eigenschaft ist nützlich, um etwa gleich lange Zertifizierungspfade mithilfe eines I-CVT mit n reflektierten Zertifikaten zu erzeugen. Die erwartete Länge dieser Pfade lässt sich wie folgt bestimmen: Laut Def. 5-5 haben interne Knoten im B^+-Baum mindestens $\lceil m/2 \rceil$ Kinder und die Wurzel mindestens zwei Kinder (außer sie ist ein Blatt). Folglich gibt es mindestens $2, 2*\lceil m/2 \rceil, 2*(\lceil m/2 \rceil)^2, \ldots$ Knoten auf den einzelnen Ebenen unterhalb der Wurzel. Demnach hat ein B^+-Baum mindestens $2*\lceil m/2 \rceil^{h-1}$ Blätter. Jedes Blatt verweist wiederum auf mindestens $\lceil m/2 \rceil - 1$ Datensätze. Daher gilt für die Anzahl n der durch den Baum referenzierten Datensätze $n \ge 2*\lceil m/2 \rceil^{h-1}*(\lceil m/2 \rceil - 1)$. Für die Höhe h eines minimal vollen Baums gilt damit $h = 1 + \log_{\lceil m/2 \rceil}(n / 2*(\lceil m/2 \rceil - 1))$, oder grob geschätzt $h \approx \log_{\lceil m/2 \rceil} n$.

Für einen maximal vollen B^+-Baum, in welchem also jeder interne Knoten einschließlich der Wurzel *m* Kinder hat und jedes Blatt *m-1* Datensätze referenziert, gilt hingegen, dass der Baum $n=m^h*(m-1)$ Datensätze referenziert und dass er eine Höhe von $h=log_m(n/(m-1)) \approx log_m n$ hat.
Ein Suchpfad von der Wurzel bis zu einem Blatt wird also bei einem Baum der Ordnung *m* ungefähr $(log_{\lceil m/2 \rceil} n) < k < (log_m n)$ Knoten beinhalten. Während der Suche nach einem bestimmten Datum muss daher höchstens auf ungefähr $log_m n$ Knoten zugegriffen werden. Dies kann als eine Verbesserung gegenüber einer Binärbaum-basierten Lösung mit $log_2 n$ besuchten Knoten bezeichnet werden.
Beim Einfügen und Löschen eines Datensatzes in den Baum fallen neben den Suchkosten auch Kosten durch die zwingende Balancierung des Baums an. Die Balancierung kann in einem rekursiven Durchlauf der Knoten entlang des jeweiligen Suchpfads zum betroffenen Blatt von „unten" bis nach „oben" zur Wurzel erfolgen. Diese Art Verwaltungskosten machen sich bei der Pflege des I-CVT bemerkbar, indem also neue (alte) Zertifikate in den Baum eingefügt (entfernt) werden müssen.

Da die Kosten vom gewählten Parameter *m* des Baums abhängen, kann durch die geeignete Wahl von *m* das jeweilig erwünschte Betriebsoptimum für einen erwarteten Bestand mit *n* Datensätzen gefunden werden. Um einen solchen Optimum zu finden, ist der Diagramm in Abb. 5-5 hilfreich [Ebi01].
Dort werden die Suchkosten mithilfe der oben ermittelten Formeln für minimal und maximal volle Bäume der Höhe *h* in Abhängigkeit von *n={10, 100, 1000, 10000}* und *m={3, 4, ..., 15}* aufgezeigt. Bei der Berechnung von *Suchkosten(n, m)* wurde angenommen, dass innerhalb eines während der Suche besuchten Knotens der jeweilige Zeiger zum nächsten Abstieg durchschnittlich in $1+log_2(\#Anzahl\ Suchschlüssel\ pro\ Knoten)$ Schritten gefunden wird.

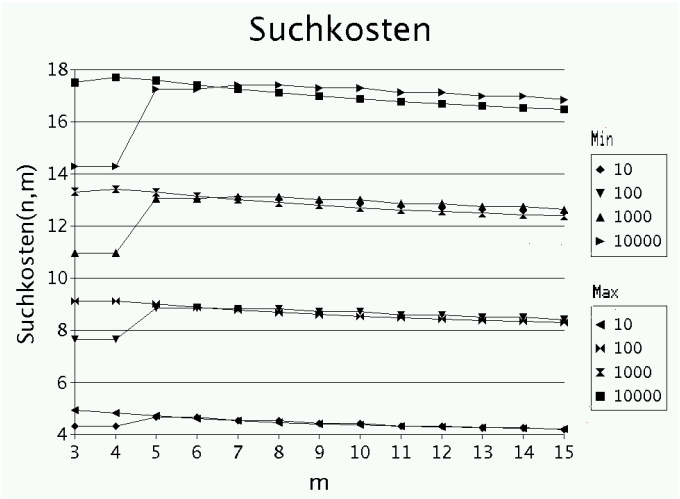


Abb. 5-5: Suchkosten in minimal und maximal vollen B^+-Bäumen aus [Ebi01]

Aus dem Diagramm wird ersichtlich, dass die Kosten im Falle eines minimal vollen Baums zunächst von *m=4* zu *m=5* ansteigen. Der Grund hierfür ist, dass ein jeder Knoten eines solchen Baums mit *m≤4* genau einen Suchschlüssel enthält, was den Abstieg auf die nächsttiefere Baumebene erleichtert. In einem maximal vollen Baum steigen hingegen die Kosten von *m=3* zu *m=4* für *n>100* zunächst leicht an, mit zunehmendem *m* sinken sie leicht ab.

In Abb. 5-6 wird ein simpler Beispielbaum der Ordnung *m=4* gezeigt. Dieser Baum referenziert neun Datensätze D_1, D_2, ..., D_9, welche aus mehreren als $Attr_i$ bezeichneten Attributen bestehen. Als Suchschlüssel für den Baum wurde die Attribut-Sequenz $\{Attr_1||Attr_2\}$ der referenzierten Datensätze gewählt. Hierbei ist das Attribut $Attr_1$ vom Typ einer (endlich langen) Zeichenfolge und $Attr_2$ einer ganzen Zahl.

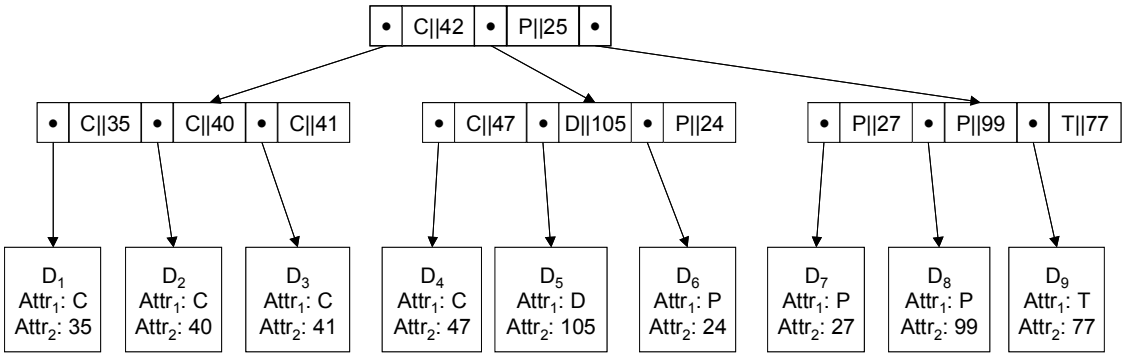


Abb. 5-6: Beispiel für einen B$^+$-Baum der Ordnung *m=4*

Nachfolgend wird zunächst die Erzeugung eines B$^+$-Baum-basierten I-CVT erläutert. Diesem folgt die Diskussion von Algorithmen zur Generierung und Verifizierung von verschiedenen Beweisen für die zertifikatsbasierte Zugriffskontrolle ausgehend von einem gegebenen I-CVT.

5.3.1 Erzeugung eines I-CVT auf Basis eines B$^+$-Baums

Die Erzeugung eines I-CVT auf Basis eines gegebenen B$^+$-Baums der Ordnung *m* erfolgt im Sinne von Def. 5-3, d.h. durch das Einfügen von sicherheitsbezogenen Informationen in die verschiedenen Baumknoten.
Hierbei wird in jedem Knoten *v* des B$^+$-Baums zusätzlich ein Typbezeichner $Typ_v=\{L, NL\}$ sowie ein Hashwert $H[v]$ gespeichert. Ein Knoten *v* erhält dabei den Typbezeichner $Typ_v=L$ (engl. *Leaf*) falls er ein Blatt ist, ansonsten erhält er $Typ_v=NL$ (engl. *Non-Leaf*). Der im Knoten *v* abgelegte Hashwert $H[v]$ wird durch die folgende Formel ermittelt:

Def. 5-6: Hashwertberechnung für die Knoten eines I-CVT

$$H[v]:=h(Typ_v \{L, NL\}, (s_1[v], s_2[v], ..., s_{t_v}[v]), (H[w_1], H[w_2], ..., H[w_{t_{v+1}}]))$$

Hierbei bezeichnen $s_1[v]$ bis $s_{t_v}[v]$ die im Knoten *v* jeweilig abgelegten insgesamt t_v (mit $t_v<m$) Suchschlüssel. $H[w_1], H[w_2], ..., H[w_{t_{v+1}}]$ sind die Hashwerte, welche in den w_1, w_2, ..., $w_{t_{v+1}}$ Kinderknoten von *v* gespeichert sind. In der gewählten Darstellung bezeichnet „ ,“ die Operation der Konkatenation. Zu beachten ist, dass die Anzahl der Kinderknoten eines Knotens in einem B$^+$-Baum variabel ausfällt. Sie hängt von der Ordnung *m* sowie der aktuellen „Gefülltheit“ des Baums ab.
Ist *v* ein Blatt des Baums, so sind w_1 bis w_{t_v} die durch den Baum signierten Datensätze, wie z.B. Attributszertifikate, die durch ihre in den I-CVT-Blättern abgelegten Hashwerte $H[w_1], H[w_2], ..., H[w_{t_v}]$ repräsentiert werden. Die Sequenz $s_1[v], s_2[v], ..., s_{t_v}[v]$ enthält die diesen Datensätzen zugeordneten Suchschlüssel.

Neben dem Hashwert *H[v]* in einem internen Knoten *v* kann dort auch die Sequenz $H[w_1], ..., H[w_{t_{v+1}}]$ der Hashwerte der Kinderknoten gespeichert werden. Dadurch kann die Generierung der Zertifizierungspfade bzw. Vollständigkeitsbeweise beschleunigt werden: Während der Beweisgenerierung muss nämlich nicht auf die nicht besuchten Kinderknoten $w_1, ..., w_{t_{v+1}}$ von *v* zugegriffen werden (was evtl. teure Blockzugriffe verursacht), nur um die dort abgelegten Hashwerte $H[w_i]$ auszulesen. Die weiter unten diskutierten Algorithmen zur Beweisgenerierung gehen von dieser Art Speicherung der Daten aus.
Im Wurzelknoten des Baums werden noch zusätzlich die Wurzelsignatur *S* sowie die als *CVT-Infos* bezeichnete Daten (siehe Abb. 4-3 in Kap. 4.2) zur Unterstützung der Zertifikatsprüfung abgelegt.

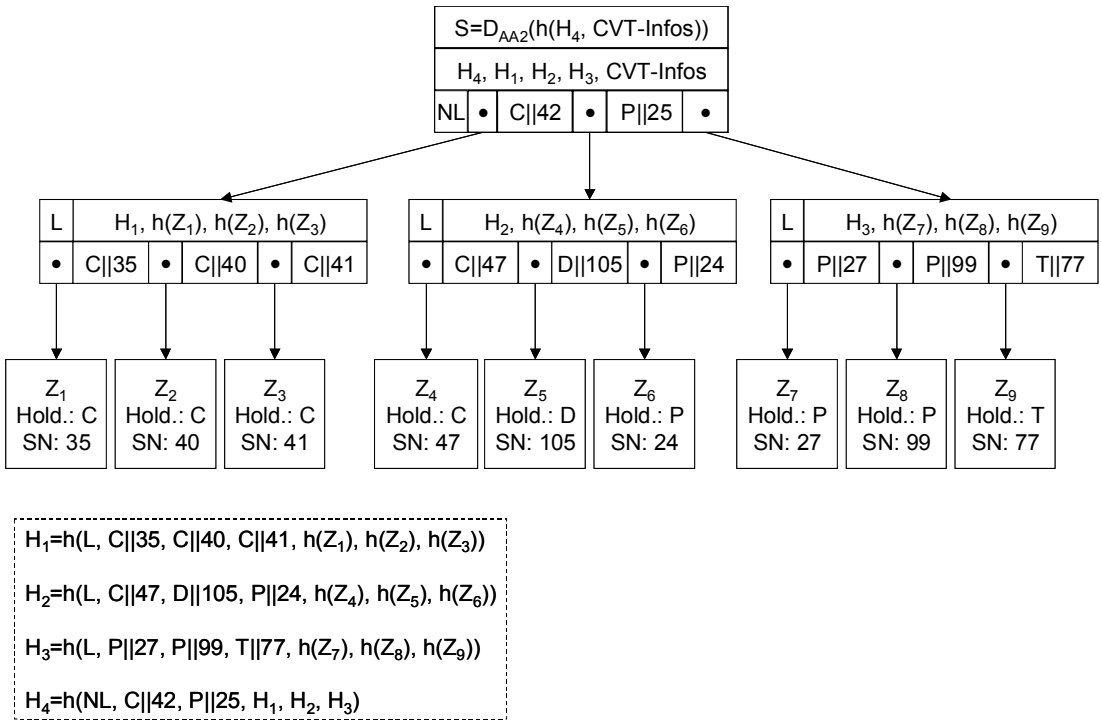


Abb. 5-7: Einfacher I-CVT auf Basis eines B$^+$-Baums der Ordnung *m=4*

In Abb. 5-7 wird ein auf diese Weise erzeugter I-CVT gezeigt (vgl. Abb. 5-6). Mithilfe dieses Baums wurden insgesamt neun Attributszertifikate $Z_1, ..., Z_9$ durch die Zertifizierungsstelle *AA2* signiert.
Als Sortier- bzw. Suchschlüssel für den Baum wurde die Attributs-Sequenz *{Holder||SerialNumber}* gemäß der beiden X.509-Zertifikatsfelder *Holder* (dargestellt als *Hold*) respektive *SerialNumber* (dargestellt als *SN*) gewählt. Die Knoten dieses Baums enthalten Suchschlüssel, Zeiger auf Kindknoten, einen Typbezeichner und einen H_i Hashwert, welcher gemäß Def. 5-6 berechnet wird (siehe Abb. 5-7 unten links).

5.3.2 Generierung und Verifizierung von Zertifizierungspfaden

Zunächst werden die Algorithmen zur Erstellung und Überprüfung von Zertifizierungspfaden anhand von einem I-CVT erläutert. Solche Zertifizierungspfade ermöglichen die Überprüfung einzelner durch den Baum erfasster oder eben nicht erfasster Zertifikate und erfüllen damit die Funktion eines Existenz- respektive Nicht-Existenz-Beweises.

Anhand eines Zertifizierungspfads kann folglich eindeutig entschieden werden, ob ein bestimmtes (vorliegendes) Attributszertifikat aktuell gültig ist.

- **Generierung von Zertifizierungspfaden:**

Ein Zertifizierungspfad $ZP(Z_s)$, der anhand eines I-CVT zu einem bestimmten Zertifikat Z mit dem Suchschlüssel s erzeugt werden kann, enthält folgende Daten: Die Wurzelsignatur S des I-CVT, die zur Überprüfung von S benötigten Daten *CVT-Infos* und eine Sequenz von *(Typbezeichner; Sequenz Suchschlüssel; Sequenz Hashwerte)*-Tripeln:

Def. 5-7: Struktur von Zertifizierungspfaden ausgehend von einem I-CVT

$\{ S; CVT\text{-}Infos; (Typ_1; s_{1_1}, ..., s_{1_{t_1}}; H_{1_1}, ..., H_{1_{t_1}}); ...; (Typ_m; s_{m_1}, ..., s_{m_{t_1}}; H_{m_1}, ..., H_{m_{t_m}}) \}$

Hierbei bezeichnen s_{ij} die Suchschlüssel des i-ten besuchten Knotens auf dem Suchpfad zum jeweilig gefragten Zertifikat. Typ_i ist der Typbezeichner des i-ten Knotens. Die Hashwerte H_{ij} sind die Hashwerte der nicht besuchten Kinder des i-ten besuchten Knotens. Die Sequenz der H_{ij} Hashwerte der Kinder des i-ten Knotens muss den Hashwert aus dem nächstgefundenen Kindknoten nicht enthalten. Dieser Hashwert kann nämlich während der Verifizierung (siehe weiter unten) des Zertifizierungspfads berechnet und in die entsprechende Position eingesetzt werden. Die relative Position eines Knotens zu seinen Geschwistern wird ebenfalls nicht benötigt, da sich diese aus den in den Knoten sortiert gespeicherten Suchschlüsseln s_{ij} und dem Suchschlüssel s des gesuchten Zertifikats Z_s ergibt. Das Tripel $(Typ_1; s_{1_1}, ..., s_{1_{t_1}}; H_{1_1}, ..., H_{1_{t_1}})$ enthält die Einträge aus dem I-CVT-Blatt, bei dem die erfolgreiche oder erfolglose Suche nach dem gefragten Zertifikat endete. Die übrigen Tripel beziehen sich auf interne Knoten (Wurzel) des Baums.

Beispielsweise der Zertifizierungspfad zum existierenden Zertifikat aus Abb. 5-7 mit dem Suchschlüssel *{C||40}*, welches für den Benutzer (*Holder,* Hold.) *C* ausgestellt wurde und die Seriennummer (*SerialNumber,* SN) *40* trägt, ist:

$ZP(Z_{C||40}) = (S; CVT\text{-}Infos; (L; \{C||35\}, \{C||40\}, \{C||41\}; h(Z_1), h(Z_2), h(Z_3));$
$(NL; \{C||42\}, \{P||25\}; H_2, H_3)).$

Der Beweis für die Nicht-Existenz eines bestimmten durch den I-CVT nicht erfassten Zertifikats Z_s mit Suchschlüssel s ist der Zertifizierungspfad $ZP(Z_s)$ zum Blatt, welches den Hashwert $h(Z_s)$ von Z_s enthalten müsste, wenn Z_s durch den I-CVT signiert wäre. Beispielsweise ist der Zertifizierungspfad zum Nachweis, dass kein Zertifikat für Benutzer (*Holder*) *Bob* mit der Seriennummer *42* ausgestellt wurde, der folgende:

$ZP(Z_{Bob||42}) = (S; CVT\text{-}Infos; (L; \{C||35\}, \{C||40\}, \{C||41\}; h(Z_1), h(Z_2), h(Z_3));$
$(NL; \{C||42\}, \{P||25\}; H_2, H_3))$

Um einen Zertifizierungspfad zu einem bestimmten Zertifikat Z_s mit dem Suchschlüssel s zu generieren, wird ausgehend von der Wurzel des I-CVT eine Suche nach dem entsprechenden Blatt, das s enthält, gestartet. Während dieser Suche werden die in den besuchten Knoten abgelegten Tripel *(Typbezeichner, Sequenz Suchschlüssel, Sequenz Hashwerte)* gesammelt.

Beispielsweise werden bei der Generierung des oben gezeigten $ZP(Z_{C||40})$ die beiden Tripel *(L; {C||35}, {C||40}, {C||41}; h(Z1), h(Z2) h(Z3))* und *(NL; {C||42}, {P||25}; H1, H2, H3)* aus den entsprechend besuchten Knoten des Baums ermittelt.
Diese Tripel werden für die Weiterbearbeitung durch den in Abb. 5-8 gezeigten Algorithmus zwischengespeichert. Dieser Algorithmus (dargestellt in Pseudo-Code) gibt den Zertifizierungspfad $ZP(Z_s)$ zum gefragten Zertifikat Z_s mit dem Suchschlüssel *s* aus. Falls der gesuchte Schlüssel *s* im Baum enthalten ist, liefert der Algorithmus einen Zertifizierungspfad, welcher die Existenz und Gültigkeit von Z_s bestätigt. Falls *s* im jeweiligen Baum nicht reflektiert ist, wird ein Zertifizierungspfad zum Blatt generiert, in welchem gemäß der Sortierregeln *s* enthalten sein müsste.

***Eingabe**: Suchschlüssel s eines Zertifikats*
***Ausgabe**: Zertifizierungspfad $ZP(Z_s)$ für das gefragte Zertifikat. Falls s im Baum nicht vorhanden ist, beweist $ZP(Z_s)$ die Nicht-Existenz von Z_s*

```
zp0← leerer Zertifizierungspfad
Knoten←Blatt, das Suchschlüssel s enthält bzw. enthalten müsste
fertig← false

Solange nicht fertig
    Typ← Typbezeichner von Knoten
    S← Sequenz der Suchschlüssel, die in Knoten gespeichert sind
    H←Sequenz der Hashwerte, die in Knoten gespeichert sind

    Falls Typ=NL (für interne Knoten)
          pos← Position von s in S
          (so dass S_pos-1<s≤S_pos beziehungsweise pos=1 oder pos=t+1)
          Lösche Hashwert an Position pos in H

    Füge (Typ, S, H) zu zp0 hinzu

    Falls Knoten nicht der Wurzelknoten ist
     Knoten← Vaterknoten von Knoten
    Ansonsten
     fertig← wahr

Ausgabe: ZP(Zs)={Wurzelsignatur, CVT-Infos, zp0}
```

Abb. 5-8 Ein Algorithmus zur Erstellung von Zertifizierungspfaden

Die Erzeugung eines Zertifizierungspfads erfolgt anhand der während der Suche nach dem Zertifikat zwischengespeicherten Tripel (siehe oben). Dabei werden sämtliche Daten aus dem jeweilig gefundenen Blattknoten zum entstehenden Zertifizierungspfad *zp0* hinzugefügt. Im Fall von internen Knoten des Baums, d.h. mit Typbezeichner *Typ=NL,* werden die jeweilig unnötigen Hashwerte, nämlich genau einer pro Knoten an der Position *pos*, entfernt. Dementsprechend wird aus dem oben gezeigten zweiten Tripel *(NL; {C||42}, {P||25}; H1, H2, H3)* der Hashwert H_1 entfernt. Dieser Hashwert wird während der Verifizierung des Zertifizierungspfads (siehe unten) berechnet und in die entsprechende Position eingesetzt.

- **Verifizierung von Zertifizierungspfaden:**

Ein Verifizierer, der von der CVT-Datenbank einen Zertifizierungspfad bekommt, muss diesen prüfen. Dabei wird anhand der im $ZP(Z_s)$ enthaltenen *(Typbezeichner, Sequenz Suchschlüssel, Sequenz Hashwerte)*-Tripeln (siehe *zp0* in Abb. 5-8*)* der - zunächst vermeintliche - Wurzelhashwert des Baums bestimmt. Dabei werden die charakteristischen Sortierregeln der jeweiligen Baumkonstruktion (hier B^+-Baum) beachtet. Außerdem wird geprüft, ob der Zertifizierungspfad tatsächlich bei einem Blattknoten sein Ende hat. Dadurch kann das Vorspielen gekürzter Zertifizierungspfade ausgeschlossen werden. Ebenfalls muss geprüft werden, ob der aus dem Zertifizierungspfad errechnete Wurzelhashwert und der durch die Prüfung der Wurzelsignatur ermittelte tatsächliche Wurzelhashwert des Baums gleich sind.

Der in Abb. 5-9 in Pseudo-Code dargestellte Algorithmus rekonstruiert den Suchpfad, welcher von der Wurzel zum jeweiligen durch die Datenbank ausgehändigten Blatt des I-CVT führt. Besteht dabei Verdacht auf einen Manipulationsversuch, wird der Zertifizierungspfad abgelehnt. Dieser Verifizierungsalgorithmus, welcher als Eingabe den gefragten Suchschlüssel s und den von der Datenbank ausgehändigten Zertifizierungspfad $ZP(Z_s)$ bekommt, bestimmt zunächst den Wert

$$h_1 := h(Typ_1; s_{1_1}, \ldots, s_{1_{t_1}}; H_{1_1}, \ldots H_{1_{t_1}}).$$

Danach erfolgt für die $1<i<m$ Ebenen des Baums, d.h. von „unten“ nach „oben“, die Berechnung

$$h_i := h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, h_{i-1}, H_{i_2}, \ldots, H_{i_{t_i}}), \text{ falls } s \le s_{i_1}$$
$$h_i := h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, H_{i_1}, \ldots, H_{i_{(l-1)}}, h_{i-1}, H_{i_{(l+1)}}, \ldots H_{i_{t_i}}), \text{ falls } s_{i_{(l-1)}} < s \le s_{i_l}$$
$$h_i := h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, H_{i_1}, \ldots, H_{i_{t_{(i-1)}}}, h_{i-1}), \text{ falls } s > s_{i_{t_i}}$$

Der Algorithmus gibt die Meldung „*Ungültiger Zertifizierungspfad*“ zurück, falls

- die jeweilige in $ZP(Z_s)$ enthaltene Wurzelsignatur keine gültige Signatur für *(h_m, CVT-Infos)* sein kann, oder
- für ein bestimmtes j die gefundenen Schlüssel s_{ij} nicht richtig geordnet sind, oder
- für ein bestimmtes j es einen Schlüssel gibt, der nicht in den Bereich passt, der von einem seiner Vorfahren bestimmt wird. Dies ist ein Schlüssel s_{ij} für den für ein bestimmtes g $s_{(i+g)_l} < s_{ij} \le s_{(i+g)_{(l+1)}}$ nicht gilt, wobei l durch $s_{(i+g)_l} < id \le s_{(i+g)_{(l+1)}}$ bestimmt ist (analog für $id \le s_{(i+g)_l}$ oder $id > s_{(i+g)_{t_i}}$) oder
- der Typbezeichner $Typ_1 \neq L$, in diesem Fall wurde ein weiter tiefer liegender Knoten des Baums evtl. verschwiegen.

Es ist zu beachten, dass aus der zweiten und dritten Bedingung folgt, dass der Algorithmus mit „*Ungültiger Zertifizierungspfad*“ zurückkehrt, wenn das gerade aus dem Zertifizierungspfad rekonstruierte Baumfragment kein Teil eines korrekt sortierten B^+-Baums (siehe Def. 5-5) sein kann.

Dadurch können auch während der Erzeugung des Baums begangenen Fehler entdeckt werden. Im Fall anderer Baumkonstruktionen bzw. Sortierregeln für die Schlüssel ändern sich diese spezifischen Angaben zur Prüfung.

***Eingaben**: Eindeutiger Suchschlüssel s des Zertifikats, Zertifizierungspfad $ZP(Z_s)$*
***Ausgabe**: Gültigkeit des Zertifikats Z_s*

$i \leftarrow 1$
$Typ \leftarrow Typ_1$
$S \leftarrow (s_{i_1}, ..., s_{i_t})$
$H \leftarrow (H_{i_1}, ..., H_{i_t})$
$h \leftarrow h(Typ, S, H)$
$min_{id} \leftarrow min(min(S), s)$
$max_{id} \leftarrow max(max(S), s)$

***Solange** $i<m$*
  $S \leftarrow (s_{i_1}, ..., s_{i_t})$
  $H \leftarrow (H_{i_1}, ..., H_{i_t})$
  $Typ \leftarrow Typ_i$
  pos ← Position von s in S
  (so dass $S_{pos-1}<s\leq S_{pos}$ bzw. $pos \leftarrow 1$ oder $pos \leftarrow t+1$ am „Knotenrand“)

  ***Falls** s_{ij} in S nicht aufsteigend geordnet **oder** $pos>1$ und $min_{id}<min(S)$*
  ***oder** $pos<t+1$ und $max_{id}>max(S)$*
    Ausgabe:** „Ungültiger Zertifizierungspfad.“, **Abbrechen

  $min_{id} \leftarrow min(S)$
  $max_{id} \leftarrow max(S)$
  Füge h in H an der Stelle pos ein
  $h \leftarrow h(Typ, S, H)$
  $i \leftarrow i+1$

  ***Falls** Wurzelsignatur keine gültige Signatur für (h, ICVT-Infos) ist **oder** $Typ_1 \neq L$*
    ***Ausgabe**: „Ungültiger Zertifizierungspfad.“*

Ansonsten
    Falls** s in $(s_{11}, s_{12}, ..., s_{1t})$ **Ausgabe: „**Gültiger Zertifizierungspfad: s in ICVT**“
    Ansonsten Ausgabe: „**Gültiger Zertifizierungspfad: s nicht in ICVT**“

Abb. 5-9 Verifizierungsalgorithmus für einen Zertifizierungspfad

Am Ende der Prüfung liefert der Algorithmus die Ausgabe, ob der Zertifizierungspfad $ZP(Z_s)$ gültig ist und ob ein Zertifikat mit dem Suchschlüssel *s* durch den Baum signiert wurde. Ist die Wurzelsignatur keine gültige Signatur für den letztlich ermittelten Wurzelhashwert (die Prüfung der Wurzelsignatur wurde bereits in Kap. 5.1.1 ausführlich behandelt), so wird der Zertifizierungspfad abgelehnt. Ist der Zertifizierungspfad gültig und bestätigt, dass Z_s durch den Baum erfasst wurde, so gibt der Algorithmus „*Gültiger Zertifizierungspfad: s in I-CVT*“ zurück.

In diesem Fall muss zusätzlich geprüft werden, ob sich der Hashwert $h(Z_s)$ des vorliegenden Zertifikats Z_s innerhalb der Sequenz $H_{1_1},... H_{1_{t_1}}$ in der richtigen Position befindet, d.h. ob für einen i $H_{1_i}=h(Z_s)$ gilt.
Ist das der Fall, so gilt das gefragte Zertifikat Z_s als aktuell gültig, ansonsten wurde Z_s zwischenzeitlich verändert und muss deshalb abgelehnt werden.
Die Funktionsweise des Algorithmus wird anhand des weiter oben exemplarisch gezeigten gültigen Nicht-Existenz-Beweises

$$ZP(Z_{Bob||42})=(S; CVT\text{-}Infos; (L; \{C||35\}, \{C||40\}, \{C||41\}; h(Z_1), h(Z_2), h(Z_3));$$
$$(NL; \{C||42\}, \{P||25\}; H_2, H_3))$$

demonstriert.
Dabei wird zunächst der Hashwert des ersten in $ZP(Z_{Bob||42})$ enthaltenen *(Typbezeichner; Sequenz Suchschlüssel; Sequenz Hashwerte)*-Tripels durch $h_1=h(L; \{C||35\}, \{C||40\}, \{C||41\}; h(Z_1), h(Z_2), h(Z_3))$ berechnet. Danach kann der Hashwert h_2 des Knotens auf der übergeordneten Ebene (in diesem Fall die Wurzel des Baums in Abb. 5-7) ermittelt werden.
Hierfür muss zunächst anhand der Suchschlüssel in jenem Knoten entschieden werden, in welche Position h_1 bei der Berechnung von h_2 gesetzt werden soll. Da der kleinste Schlüssel $\{C||42\}$ innerhalb des zweiten mit dem Zertifizierungspfad $ZP(Z_{Bob||42})$ ausgehändigten Tripels $(NL; \{C||42\}, \{P||25\}; H_2, H_3)$ größer ist, als der gesuchte Schlüssel $\{Bob||42\}$, wird dem Verifizierer „klar“, dass h_1 der Hashwert eines linken Kindknotens der Wurzel sein muss.
Folglich wird h_2 durch $h_2=h(NL; \{C||42\}, \{P||25\}; h_1, H_2, H_3)$ berechnet. Da der Zertifizierungspfad nur diese zwei Knoten reflektiert, kann anschließend die Wurzelsignatur $S=D_{AA2}(h(H_4, CVT\text{-}Infos))$ des Baums (siehe Abb. 5-7) geprüft werden. Hierbei wird festgestellt, ob $E_{AA2}(S)=h(h_2, CVT\text{-}Infos)$ und folglich $H_4=h_2$ gilt.

5.3.3 Abhängigkeit von der Sicherheit der verwendeten Hashfunktion

Die Sicherheit eines mit Hilfe der in Kap. 5.3.2 behandelten Algorithmen hängt maßgeblich von der Sicherheit der Hashfunktion h ab, die bei der Baumkonstruktion verwendet wurde. Das heißt, ein Zertifizierungspfad gilt nur dann als ein sicherer Existenz- oder Nicht-Existenz-Beweis für ein durch den jeweiligen I-CVT erfasstes oder nicht erfasstes Zertifikat, wenn h kollisionsresistent (siehe Def. 2-4) ist. Ansonsten kann nämlich die Datenbank den I-CVT, genauer die hierin enthaltenen Hashwerte, Suchschlüssel und Typbezeichner manipulieren, um einen falschen Wurzelhashwert vorzutäuschen und damit gefälschte Zertifizierungspfade unentdeckt auszustellen.
Im Folgenden wird gezeigt: Falls das in Def. 5-6 vorgeschlagene Berechnungsschema für die in den Knoten des I-CVT abgelegten Hashwerte mit der Erfolgswahrscheinlichkeit ε gebrochen werden kann, dann können für die Hashfunktion h mit der gleichen Wahrscheinlichkeit Kollisionen gefunden werden. Das „Brechen“ der Hashwerte in den I-CVT-Knoten bedeutet hier, dass anhand solcher Hashwerte sowohl die Existenz als auch die Nicht-Existenz eines bestimmten Zertifikats vorgetäuscht werden kann.
Sei A ein Algorithmus, der mit Wahrscheinlichkeit ε ein Paar von sich widersprechenden Zertifizierungspfaden $ZP(Z_s)$ und $\underline{ZP(Z_s)}$ zu demselben Zertifikat Z_s mit dem Suchschlüssel s anhand von einem bestimmten I-CVT zurückliefern kann, so dass der Verifizierungsalgorithmus V (siehe Abb. 5-9) diese als *„Gültiger Zertifizierungspfad: s in I-CVT“* respektive *„Gültiger Zertifizierungspfad: s <u>nicht</u> in I-CVT“* auswertet.

Das bedeutet, mit Hilfe von A kann die Datenbank sowohl die Existenz als auch die Nicht-Existenz von Z_s vortäuschen und damit das Ergebnis der Zertifikatsprüfung beeinflussen. Behauptet wird nun: Wenn es einen solchen Algorithmus gibt, so kann man mit Wahrscheinlichkeit ε eine Kollision für die verwendete Hashfunktion h finden. Dies gilt unter der Annahme, dass die zum Signieren des Wurzelhashwerts verwendete Funktion, z.B. RSA, sicher ist. Ansonsten kann nämlich einem Verifizierer jederzeit ein beliebiger gefälschter Wurzelhashwert, bzw. ein beliebiger Zertifizierungspfad als „authentisch" vorgetäuscht werden.
Der Algorithmus A gibt mit der Wahrscheinlichkeit ε ein Tupel $(s, ZP(Z_s), \underline{ZP(Z_s)})$ zurück, so dass $V(s, ZP(Z_s))=$„*Gültiger Zertifizierungspfad: s in I-CVT*" und $V(s, \underline{ZP(Z_s)})=$„*Gültiger Zertifizierungspfad: s nicht in I-CVT*" und $ZP(Z_s)\neq\underline{ZP(Z_s)}$ gelten.

Deshalb ist

$$ZP(Z_s)=[S;\ CVT\text{-}Infos;\ (Typ_1;\ k_{1_1},\ldots k_{1_{t_1}};\ H_{1_1},\ldots H_{1_{t_1}});\ \ldots;\ (Typ_m;\ k_{m_1},\ldots k_{m_{t_m}};\ H_{m_1},\ldots H_{m_{t_m}})]$$

und

$$\underline{ZP(Z_s)}=[S;\ CVT\text{-}Infos;\ (\underline{Typ_1};\ \underline{k_{1_1}},\ldots \underline{k_{1_{t_1}}};\ \underline{H_{1_1}},\ldots \underline{H_{1_{t_1}}});\ \ldots;\ (\underline{Typ_m};\ \underline{k_{m_1}},\ldots \underline{k_{m_{t_m}}};\ \underline{H_{m_1}},\ldots \underline{H_{m_{t_m}}})]$$

für ein bestimmtes m, wobei $m\geq 1$ die Anzahl der jeweiligen Baumebenen bedeutet.

Seien nun die beiden durch A generierten Zertifizierungspfade $ZP(Z_s)$ und $\underline{ZP(Z_s)}$ mit Hilfe des Verifizierungsalgorithmus in Abb. 5-9 parallel bearbeitet. Bei der Bearbeitung von $ZP(Z_s)$ gilt dabei

$$h_i:=h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, h_{i-1}, H_{i_2}, \ldots, H_{i_{t_i}}),\ falls\ s\leq s_{i_1}$$
$$h_i:=h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, H_{i_1}, \ldots, H_{i_{(l-1)}}, h_{i-1}, H_{i_{(l+1)}}, \ldots, H_{i_{t_i}}),\ falls\ s_{i_{(l-1)}}<s\leq s_{i_l}$$
$$h_i:=h(Typ_i, s_{i_1}, \ldots, s_{i_{t_i}}, H_{i_1}, \ldots, H_{i_{t_{(i-1)}}}, h_{i-1}),\ falls\ s>s_{i_{t_i}}.$$

Analog wird $\underline{ZP(Z_s)}$ behandelt, wobei alle Variablen entsprechend einen Unterstrich erhalten. Da gemäß der obigen Annahme die Wurzelsignatur S eine gültige Signatur für die beiden Hashwerte $h(h_m, CVT\text{-}Infos)$ und $h(\underline{h_m}, CVT\text{-}Infos)$ ist, folgt $h_m=\underline{h_m}$. Dies gilt jedoch nur unter der Annahme, dass *CVT-Infos* nicht manipuliert wurde. Das bedeutet also, dass die beiden anhand von $ZP(Z_s)$ und $\underline{ZP(Z_s)}$ ermittelten Wurzelhashwerte h_m respektive $\underline{h_m}$ nach den obigen geschachtelten Hashwertberechnungen identisch ausfallen.
Aus dem Ergebnis der Verifizierung $V(s, ZP(Z_s))=$„*Gültiger Zertifizierungspfad: s in I-CVT*" und aus $V(s, \underline{ZP(Z_s)})=$„*Gültiger Zertifizierungspfad: id nicht in I-CVT*", folgt aber $s\in\{s_{1_1},\ldots, s_{1_{t_1}}\}$ respektive $s\notin\{\underline{s_{1_1}}, \ldots, \underline{s_{1_{t_1}}}\}$. Hieraus folgt $\{s_{1_1}, \ldots, s_{1_{t_1}}\}\neq\{\underline{s_{1_1}}, \ldots, \underline{s_{1_{t_1}}}\}$, d.h. die Sequenz der Suchschlüssel im vorgetäuschten Blatt muss verschieden sein von der Sequenz der Suchschlüssel im wirklich vorhandenen Blatt des Baums. Das heißt, dass die während der ersten Iteration in die Berechnung von h_m und $\underline{h_m}$ einfließenden Daten an dieser Stelle unterschiedlich sind.

Um aber am Ende der Verifizierung von $ZP(Z_s)$ und $\underline{ZP(Z_s)}$ identische Wurzelhashwerte $h_m=\underline{h_m}$ zu erhalten, muss ab einer bestimmten Baumebene e ($e \leq m$) $\underline{h_e}=h_e$, $\underline{h_{e+1}}=h_{e+1}$, ..., $\underline{h_m}=h_m$ mit $\underline{h_{e-1}} \neq h_{e-1}$ gelten.. Um dies zu erreichen, muss A im jeweilig betrachteten I-CVT-Knoten auf der Ebene e des Baums entweder

- passende $s_{ej} \neq \underline{s_{ej}}$ Suchschlüssel mit $h(\ldots, s_{ej}, \ldots) = h(\ldots, \underline{s_{ej}}, \ldots)$, oder
- passende $H_{ej} \neq \underline{H_{ej}}$ Hashwerte mit $h(\ldots, H_{ej}, \ldots) = h(\ldots, \underline{H_{ej}}, \ldots)$, oder
- passende $Typ_e \neq \underline{Typ_e}$ Typbezeichner mit $h(Typ_e, \ldots) = h(\underline{Typ_e}, \ldots)$

und damit (mindestens) eine Kollision der Hashfunktion h finden. Aus diesen Überlegungen folgt unmittelbar, dass man (wie oben behauptet) eine Kollision für h mit der Erfolgswahrscheinlichkeit ε von A finden wird.

5.3.4 Generierung und Verifizierung von Vollständigkeitsbeweisen

Im Folgenden wird anhand von einem Beispiel gezeigt, wie von einem I-CVT ausgehend Vollständigkeitsbeweise für die sichere Zugriffskontrolle erzeugt und geprüft werden können. Mit dem in Abb. 5-10 gezeigten I-CVT wurden insgesamt neun Attributszertifikate $Z_1, Z_2, \ldots, Z_9$ der Benutzer *A, B, Bob, D, P* und *T* signiert.

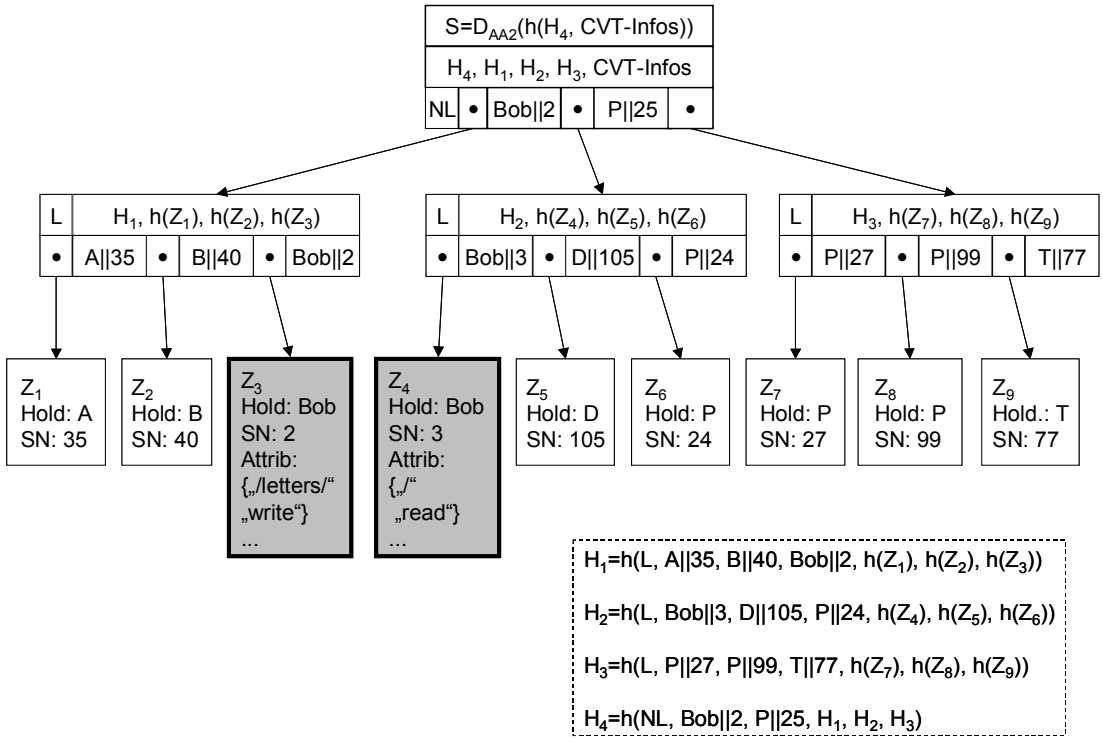


Abb. 5-10: I-CVT mit Zertifikaten mehrerer Benutzer

Für die Sortierung des Baums wurde der aus den X.509-Zertifikatsfeldern *Holder* und *SerialNumber* zusammengesetzte Schlüssel *{Holder||SerialNumber}* gewählt. Dies hat zu Folge, dass bspw. die fettgedruckten Zertifikate Z_3 und Z_4 von *Bob* mit den Suchschlüsseln *Bob||2* respektive *Bob||3* durch benachbarte Blätter des Baums reflektiert sind.

Wie bereits in Kap. 5.2.1 diskutiert, kann anhand von einem I-CVT ein Vollständigkeitsbeweis $VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$ generiert werden, welcher einem Verifizierer das Zusammensetzen und Prüfen einzelner $ZP(Z_i)$ Zertifizierungspfade zu den gefragten k Zertifikaten ermöglicht.
Solche Beweise anhand von einem I-CVT auszustellen bereitet natürlich keine Probleme: Die einen solchen Beweis bildenden Zertifizierungspfade können gemäß der in Kap. 5.3.2 diskutierten Algorithmen durch die Datenbank erstellt (siehe Abb. 5-8) und anschließend durch den Verifizierer geprüft (siehe Abb. 5-9) werden. Aus diesem Grund werden hier diese Art Beweise nicht weiter diskutiert.

Statt dessen wird die Erzeugung sowie Überprüfung von Vollständigkeitsbeweisen $VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$ behandelt, die zur Absicherung von Intervallsuchen innerhalb eines I-CVT geeignet sind.
Konkret bedeutet dies den Nachweis, welche der k im Baum durch ihre zugeordneten Suchschlüssel $s_1, s_2, ..., s_k$ potenziell reflektierten Zertifikate $Z_{s1}, Z_{s2}, ..., Z_{sk}$ tatsächlich durch den Baum erfasst wurden, wenn diese Suchschlüssel ins (geordnete) Intervall $[s_1; s_k]$ fallen. Eine wesentliche Anwendung innerhalb der Zugriffskontrolle ist der Aufbau von vollständigen Benutzerprofilen, die sämtliche Berechtigungen und folglich sämtliche Zertifikate jeweils eines Benutzers enthalten. Um dies zu erreichen, müssen alle an den jeweiligen Benutzer (*Holder*) ausgestellten Attributszertifikate bei der Datenbank angefordert und anhand eines Vollständigkeitsbeweises auf ihre Vollständigkeit und aktuelle Gültigkeit hin überprüft werden.

- **Erstellung von Vollständigkeitsbeweisen für die Intervallsuche:**

In Abb. 5-11 wird ein Algorithmus (in Pseudo-Code) gezeigt, welcher ausgehend von der Wurzel eines I-CVT einen Vollständigkeitsbeweis für eine Intervallsuche nach Z_{si} Zertifikaten mit den Suchschlüsseln s_i aus dem Intervall $[s_{min}; s_{max}]$ zurückliefert. Der Algorithmus setzt voraus, dass dem anfragenden Verifizierer die notwendigen Eingabeparameter s_{min} und s_{max} bekannt sind. Das heißt, der Verifizierer muss den kleinst- bzw. größtmöglichen Suchschlüssel kennen, den die jeweilig gefragten Zertifikate bei der Generierung des Baums haben erhalten können.
Den Kern des Algorithmus bildet die rekursive Tiefensuche *tiefensuche()*. Diese fahndet innerhalb des I-CVT nach den Blättern des Baums, welche Suchschlüssel s_i aus dem gefragten Intervall $[s_1; s_k]$ sowie Hashwerte $h(Z_{si})$ der gefragten Z_{si} Zertifikate enthalten können. Der entstehende Vollständigkeitsbeweis *VB* enthält Angaben zu sämtlichen hierbei besuchten Knoten des Baums. Dazu können neben erfolgreichen Treffern auch Hinweise zu „Sackgassen“ gehören, je nach Ablauf der Suche. Damit ermöglicht *VB* die Rekonstruktion von Abstiegen, welche tatsächlich zu keinem Ergebnis führten.
Die Funktion *tiefensuche()* fügt von der Wurzel *wurzel* ausgehend Daten aus internen Knoten und Blättern des Baums in *VB* hinzu. Diese Daten sind Quadrupel der Form $(H_{Knoten}, Typ_{Knoten}, S_{Knoten}, H_{Kindknoten})$. Hier bezeichnen H_{Knoten} den Hashwert, Typ_{Knoten} den Typbezeichner und S_{Knoten} die Sequenz der Suchschlüssel, die im jeweiligen Knoten des Baums abgelegt sind. $H_{Kindknoten}$ symbolisiert die (evtl. leere) Sequenz von Hashwerten der Kinder eines Knotens, welche während der Tiefensuche nicht besucht wurden. Dabei wird angenommen, dass jeder interne Knoten die Hashwerte seiner Kinder enthält. Sollte dies jedoch nicht der Fall sein, so muss auf die im Übrigen nicht zu betretenden Knoten zugegriffen werden, um ihnen diese Hashwerte zu entnehmen.

Im Fall eines Blatts enthält $H_{Kindknoten}$ die Hashwerte von allen durch dieses Blatt erfassten Zertifikaten.

***Eingabe**: Intervall (s_{min}; s_{max}) von I-CVT-Sortierschlüsseln*
***Ausgabe:** Vollständigkeitsbeweis VB für alle Zertifikate mit Suchschlüssel innerhalb des Intervalls. VB beweist die Gültigkeit der existierenden Zertifikate und die Nicht-Existenz aller anderen.*

```
wurzel: Wurzelknoten des I-CVT
m: maximale Anzahl von Kindern eines internen Knotens (Ordnung des Baums)
pos_First, pos_Last: Hilfsvariablen zur Bestimmung rekursiver Abstiege

VB ←leerer Vollständigkeitsbeweis
Füge Wurzelsignatur, CVT-Infos zu VB hinzu
tiefensuche(wurzel)

Funktion tiefensuche(knoten)

    H ←Sequenz der Hashwerte aus allen Kindern von knoten
    S ←Sequenz der Suchschlüssel von knoten
    Typ ←Typbezeichner von knoten
    P ←Sequenz der Zeiger auf Kinder von knoten

    Füge {h(Typ, S, H), Typ, S, H} knoten zu VB hinzu
    Falls Typ=L
        return („Aufstieg innerhalb der Rekursion“)

    Falls Typ=NL
        i←m
        Solange (i>1 und S_{i-1} ≥ s_min) {i←i–1}
        pos_First←i
        i←1
        Solange (i<m und S_i≤s_max) {i←i+1}
        pos_Last←i

        Für alle i: pos_First ≤ i ≤pos_Last
        {Lösche Hashwert aus H an Position i („Zu besuchende Kinder“)
        tiefenSuche(P_i)}
        return

    Andernfalls Abbruch („Fehlerhaft angelegter Baum“)
```

Abb. 5-11: Algorithmus zur Erstellung eines Vollständigkeitsbeweises

Die geschachtelt dargestellte Struktur eines mit diesem Algorithmus erzeugten Vollständigkeitsbeweises $VB(Z_{s1}, ..., Z_{sk})$ (unter der vereinfachenden Annahme, dass die Wurzel kein Blatt ist) ist die folgende:

Def. 5-8: Struktur von Vollständigkeitsbeweisen

$VB(Z_{s1}, Z_{s2}, ..., Z_{sk})$:=
{
Wurzelsignatur;
CVT-Infos;
Erfasste Knoten:
Wurzel: (H_{Wurzel}; Typ_{Wurzel}; s_1, ..., s_{m-1}; Hashwerte nicht besuchter Kinder);
Besuchte Kinder aus $Knoten_{11}$, $Knoten_{12}$, ..., $Knoten_{1m}$:
$Knoten_{11}$: (H_1; $Typ_{Knoten\,11}$; s_m, ..., s_{2m-2}; Hashwerte nicht besuchter Kinder);
Besuchte Kinder: aus $Knoten_{21}$, $Knoten_{22}$, ..., $Knoten_{2m}$;
...
...
$Knoten_{1m}$:(H_m; $Typ_{Knoten\,1m}$; $s_{m*(m-1)-m-1}$, ..., $s_{m(m-1)}$; Hashwerte nicht besuchter Kinder);
Besuchte Kinder: aus $Knoten_{2m*(m-1)}$, $Knoten_{2m*(m-1)+1}$, ..., $Knoten_{2m*m}$;
...
}

Hierbei werden die während der Tiefensuche betretenen Knoten des Baums entsprechend durch „*Besuchte Kinder*“ markiert und gruppiert, um die zu den Ebenen *1, 2, ..., h* des Baums gehörenden Knoten $Knoten_{11}$, $Knoten_{12}$, ..., $Knoten_{h_m}$ zu identifizieren. Welche Kinder eines internen Knotens während der Suche besucht werden müssen, wird mit Hilfe der Hilfsvariablen pos_{First} und pos_{Last} bestimmt. Beim Setzen deren Werte werden die Sortierregeln des B^+-Baums (siehe Def. 5-5) berücksichtigt. Dabei werden genau die Teilbäume des aktuellen Knotens bestimmt, die einen Eintrag (Zertifikat) zu ins gefragte Intervall passenden Suchschlüsseln enthalten (können). Erreicht die erfolgreiche oder erfolglose Suche ein Blatt ($Typ=L$) des Baums, so kehrt die Funktion *tiefensuche()* zurück. Der Algorithmus terminiert unter der Annahme, dass alle Abstiege in einem der Blätter des (endlichen) Baums enden. Im Fall eines fehlerbehafteten Baums, welcher einen Typbezeichner $Typ \notin \{L, NL\}$ enthält, wird der Vorgang abgebrochen.
Ist die Ergebnismenge der Suche nach den gefragten Zertifikaten leer, so enthält der erzeugte Vollständigkeitsbeweis *VB* neben Daten aus den besuchten internen Knoten auch Daten aus maximal zwei (benachbarten) Blättern des I-CVT. Diese Blätter enthalten im Fall der hier betrachteten Intervallsuche Suchschlüssel s_i, für die $s_i<s_{min}$ respektive $s_i>s_{max}$ gilt. In diesem Fall ist *VB* also ein Nicht-Existenz-Beweis für sämtliche Z_{si} Zertifikate mit s_i Suchschlüsseln aus dem Intervall $[s_{min}; s_{max}]$.
Ansonsten werden Blätter des Baums gefunden, welche Suchschlüssel s_i enthalten, für welche $s_{min} \leq s_i \leq s_{max}$ gilt. Da der Weg zu diesen Blättern (Treffern) durchaus auch über „Sackgassen“ führen kann, kann *VB* Angaben zu Blättern des Baums mit Suchschlüsseln $s_i<s_{min}$ respektive $s_i>s_{max}$ enthalten.

Angenommen im Beispielbaum in Abb. 5-10 wird nach „Spuren“ zu allen aktuell gültigen Attributszertifikaten von *Bob* gesucht. Der gemäß des Algorithmus in Abb. 5-11 erzeugte Vollständigkeitsbeweis *VB(Holder="Bob")* ist:

VB(Holder="Bob")=VB($Z_{Bob||0}$, $Z_{Bob||1}$, ..., $Z_{Bob||max(SerialNumber)}$):=
{
Wurzelsignatur: S=D_{AA2}(H_4, CVT-Infos);
CVT-Infos: CVT-Infos;
Erfasste Knoten:
Wurzel: (H_4, NL, Bob||2, P||25, H_3);
Besuchte Kinder: $Knoten_1$, $Knoten_2$;
$Knoten_1$: (H_1, L, A||35, B||40, Bob||2, h(Z_1), h(Z_2), h(Z_3));
$Knoten_2$: (H_2, L, Bob||3, D||105, P||24, h(Z_4), h(Z_5), h(Z_6));
}

Wie man sieht, werden in *VB(Holder="Bob")* zwei benachbarte Blätter, hier bezeichnet als $Knoten_1$ und $Knoten_2$, des Baums reflektiert, die neben anderen die auf die Anfrage passenden Suchschlüssel *Bob||2* und *Bob||3* enthalten. Da die Wurzel des Baums die beiden Schlüssel *Bob||2* und *P||25* enthält, wurden von dieser ausgehend zwei (erfolgreiche) Abstiege über die entsprechenden Zeiger getätigt.

Wird hingegen im Beispielbaum (siehe Abb. 5-10) nach Zertifikaten des Benutzers *K* gesucht, dient der Vollständigkeitsbeweis:

VB(Holder="K")=VB($Z_{Bob||3}$, ..., $Z_{K||0}$, ..., $Z_{K||max(SerialNumber)}$, ..., $Z_{P||24}$):=
{
Wurzelsignatur: S,
CVT-Infos: CVT-Infos,
Wurzel: (H_4; NL; Bob||2, P||25; H_1, H_3);
Besuchte Kinder: $Knoten_2$;
$Knoten_2$: (H_2; L; Bob||3, D||105, P||24; h(Z_4), h(Z_5), h(Z_6))
}

als Nachweis der Nicht-Existenz aller für *K* ausgestellten Zertifikate. In diesem Fall werden Daten aus einem einzigen Blattknoten ausgehändigt, da alle Zertifikate von *K* hierin hätten reflektiert sein müssen.
Wie oben angedeutet, bestätigt der Beweis *VB(Holder="K")* die Existenz bzw. Nicht-Existenz der Zertifikate einer Reihe von eigentlich nicht gefragten Benutzern: Ein Verifizierer kann nämlich anhand von *VB(Holder="K")* feststellen, dass für die Benutzernamen *C..., E..., F..., ..., O* gar keine Zertifikate durch den Baum signiert wurden. Dies kann als ein Nachteil des Verfahrens angesehen werden, besonders wenn diese Information geheim gehalten werden muss. Hierbei muss man aber bedenken, dass ein Verifizierer grundsätzlich immer nachfragen kann, ob Zertifikate beispielsweise für den Benutzer *F* vorliegen. Sollte dies in einem System jedoch verhindert werden, so muss eine entsprechende Zugriffskontrolle seitens der CVT-Datenbank erfolgen, indem es bestimmt wird, welche Anfragen von welchen Verifizierern eines Systems beantwortet werden müssen.

- **Verifizierung von Vollständigkeitsbeweisen:**

Nachfolgend wird diskutiert, wie mit dem Algorithmus in Abb. 5-11 erzeugte Vollständigkeitsbeweise überprüft werden können. Angenommen wird dabei, dass dem Verifizierer *k* Zertifikate Z_{s1}, Z_{s2}, ..., Z_{sk} vorliegen. Diese erhielt er von der Datenbank auf seine Anfrage hin nach allen Z_{si} Zertifikaten mit Suchschlüsseln $s_i \in [s_{min}; s_{max}]$.
Mit Hilfe des zugehörigen Vollständigkeitsbeweises muss nun entschieden werden, welche der erhaltenen Zertifikate gültig, d.h. durch den jeweiligen I-CVT tatsächlich erfasst, sind. Außerdem muss sichergestellt werden, dass der Baum keine weiteren (durch die Datenbank verschwiegenen) Zertifikate mit den zugeordneten Suchschlüsseln $s_i \in [s_{min}; s_{max}]$ signiert.

Eingabe: *Vollständigkeitsbeweis VB, Intervall* $[s_{min}; s_{max}]$
Ausgabe: *Aussage, ob VB korrekt und authentisch ist und, ob Knoten des Baums, d.h. Zertifikate, verschwiegen wurden.*

Sig: In VB enthaltene Wurzelsignatur des I-CVT
CVT-Infos: Prüfdaten zu Sig. bzw. Zertifikaten im I-CVT
wurzel: Wurzelknoten in VB (i.d.R. auch im I-CVT)
knoten: in VB erfasster Knoten des I-CVT
H_{wurzel}*: Angabe zum Wurzelhashwert des I-CVT in VB*
m: maximale Anzahl von Kindern eines Knotens (Ordnung des I-CVT)
pos_{First}, pos_{Last}*: Hilfsvariablen zur Bestimmung von Abstiegen innerhalb von VB*
CertsOK[]: (leere) Liste von Schlüsseln gültiger Zertifikate

valid←true;
verifiziereVB(wurzel);
Falls *Sig gültige Signatur über* (H_{wurzel}, *CVT-Infos)* ***und*** *valid=true*
Ausgabe *„Vollständigkeitsbeweis gültig, Zertifikate CertsOK sind durch den Baum signiert.“*
}
Ansonsten Ausgabe „Manipulierter Beweis, ablehnen!“

Abb. 5-12: Prüfung von Vollständigkeitsbeweisen

In Abb. 5-12 und Abb. 5-13 wird eine der möglichen Vorgehensweisen zur Prüfung von Vollständigkeitsbeweisen (dargestellt in Pseudo-Code) gezeigt. Diesem liegt die rekursive Funktion *verifiziereVB(knoten)* zugrunde. Diese Funktion setzt die Variable *valid*, die anfangs den Wert *true* erhält, auf den Wert *false*, falls während der Prüfung Verdacht auf Manipulation seitens der Datenbank besteht.
Außerdem schreibt die Funktion die Suchschlüssel der anhand von *VB* als gültig anerkannten Zertifikate in die Liste *CertsOK*. Bleibt der Wert der Variable *valid=true* nach der Prüfung unverändert, so werden der Vollständigkeitsbeweis *VB* sowie die in *CertsOK* aufgelisteten Zertifikate als gültig erkannt. Eine weitere Bedingung hierfür ist natürlich, dass die in *VB* enthaltene Wurzelsignatur *Sig* authentisch und korrekt ist.

Die Prüfung wird am Wurzelknoten *wurzel* von *VB* (bzw. dem ursprünglichen I-CVT) gestartet. Dabei werden die in *VB* enthaltenen $(H_{Knoten}, Typ_{Knoten}, S_{Knoten}, H_{Kindknoten})$-Tupeln zu den einzelnen Knoten des Baums in einem rekursiven Vorgang „von oben nach unten“ geprüft (vgl. mit dem Ansatz „von unten nach oben“ bei der Prüfung von Zertifizierungspfaden in Kap. 5.3.2).

```
Funktion verifiziereVB(knoten) {
    S←Sequenz von Suchschlüsseln in knoten;
    H← Sequenz von Hashwerten in knoten;
    Typ←Typbezeichner in knoten;
    P← Sequenz von Zeigern auf in VB erfassten Kinder von knoten;
    H_knoten:←Im knoten ursprünglich abgelegter Hashwert;

    Falls Typ=L {
        Falls h(Typ, S, H)≠H_knoten
        {valid=false; Abbruch („Falscher Hashwert im Blatt“)}

        Für alle j: von 1≤j≤m-1 („Für alle Suchschlüssel im Blatt“) {
        Falls s_min≤S_j≤s_max und Z_Sj liegt nicht vor
        {valid=false; Abbruch („Zertifikat Z_sj wurde verschwiegen“)}

        Falls s_min≤S_j≤s_max und Z_Sj liegt vor {
        Falls h(Z_Sj)≠H_j {valid=false; Abbruch („Zertifikat Z_Sj gefälscht“)}
        Andernfalls füge S_j in CertsOK hinzu;}
        }
        return („Daten im Blatt für korrekt befunden, Aufstieg“) }

    Falls Typ=NL {
        int i←m;
        Solange (i>1 und S_i-1≥s_min) {i←i–1;}
        pos_First←i; i←1;
        Solange (i<m und S_i≤s_max) {i←i+1;}
        pos_Last←i;
        Für alle i: pos_First ≤ i ≤ pos_Last {
            Falls Kindknoten P_i in VB nicht vorhanden
            {valid=false; Abbruch („Ast wurde verschwiegen“)}
            Andernfalls füge P_i.H_knoten in H ein, so dass die
            Ordnung der Hashwerte in I-CVT hergestellt wird}

        Falls h(Typ, S, H)≠H_knoten
        {valid=false; Abbruch („Falscher Hashwert in Knoten“)}

        Für alle i: pos_First ≤ i ≤ pos_Last
        verifiziereVB(P_i) („Wiederhole Prüfung für Teilbäume in VB“)
        return }
}
```

Abb. 5-13: Prüfung von Vollständigkeitsbeweisen (Fortsetzung)

Ist der aktuell besuchte Knoten in *VB* ein Blatt (*Typ=L*) des Baums, so wird zunächst geprüft, ob der hierin enthaltene Hashwert H_{knoten} ein gültiger Hashwert über die in diesem Blatt abgelegten Typbezeichner *Typ*, Suchschlüssel *S* und Hashwerte *H* ist.
Ist das nicht der Fall, so wird *valid* auf den Wert *false* gesetzt und der Vorgang wird abgebrochen.
Ansonsten wird geprüft, ob sämtliche in diesem Blatt reflektierten Z_{si} Zertifikate mit Suchschlüsseln $s_i \in [s_{min};\ s_{max}]$ dem Verifizierer auch tatsächlich vorliegen, und ob die vorliegenden Zertifikate tatsächlich die im Blatt, d.h. in der jeweiligen Sequenz *H*, gespeicherten Hashwerte haben.
Wird eines dieser Kriterien verletzt, wird *valid* auf *false* gesetzt und der Vorgang wird abgebrochen. Ansonsten werden die s_i Suchschlüssel der im Blatt durch ihre Hashwerte korrekt repräsentierten Zertifikate in die Liste *CertsOK* eingefügt. Anschließend erfolgt ein „Aufstieg“ innerhalb der Rekursion.
Im Fall von internen Knoten (*Typ=NL*) des I-CVT, die in *VB* reflektiert sind, wird folgendermaßen vorgegangen: Zunächst wird geprüft, welche der im Knoten enthaltenen Suchschlüssel $s_i \in S$ das Suchkriterium $s_i \in [s_{min};\ s_{max}]$ erfüllen. Dabei werden die Variablen pos_{First} und pos_{Last} gesetzt. Für die zutreffenden Positionen wird anschließend geprüft, ob alle entsprechenden Kinder des Knotens in *VB* aufgenommen wurden. Ist das nicht der Fall, so wurde seitens der Datenbank ein gültiger Abstiegszweig innerhalb des I-CVT, mit womöglich noch gültigen das Suchkriterium erfüllenden Zertifikaten, verschwiegen. Dem entsprechend wird der Vorgang abgebrochen. Ansonsten wird geprüft, ob der im Knoten abgelegte Hashwert H_{knoten} korrekt ist. Wenn ja, so wird die Funktion *verifiziereVB* für alle vorher ermittelten Kindknoten aufgerufen.

6 Konzepte für die effiziente und sichere Ursprungsprüfung

Mithilfe von *Improved Certification Verification Trees* (I-CVT) können durch eine Zertifizierungsstelle ausgestellte Attributszertifikate häufig aktualisiert und sicher auf ihre Gültigkeit überprüft werden. Während dieser Überprüfung kann entschieden werden, ob ein vorliegendes Attributszertifikat in einer integeren und authentischen Form vorliegt. Dank der eingeführten Vollständigkeitsbeweise kann ebenfalls entschieden werden, ob dem Verifizierer tatsächlich sämtliche im gegebenen Zugriffsfall benötigte, durch die jeweilige Stelle erzeugte Zertifikate vorliegen.
Ein Problem, welches sich allein durch die Verwendung von I-CVTs nicht lösen lässt, ist die Ursprungsprüfung der auf diese (oder andere) Weise signierten Zertifikate. Ist die ein vorliegendes Attributszertifikat signierende Stelle als nicht vertrauenswürdig eingestuft, so muss herausgefunden werden, ob das Zertifikat durch Delegierungen entstand, die einen vertrauten Ursprung haben.
Diese Prüfung erfolgt gemäß der in Abb. 2-12 gezeigten und in Kap. 2.3.3 im Detail behandelten Prozessschritte. Während der Ursprungsprüfung, die eine Rekonstruktion der Delegierungshistorie eines Zertifikats voraussetzt, müssen mehrere Zertifikate sowie die hierin abgelegten delegierungsbezogenen Daten (siehe die relevanten X.509-Erweiterungsfelder in Kap. 2.3.2) entsprechend ausgewertet werden. Diese Prüfung ist ein zeitaufwändiger Prozess, der zu zusätzlichen Wartezeiten innerhalb der Zugriffskontrolle führen kann, wie dies auch aus einem Beispiel hervorgeht. Abb. 6-1 zeigt eine (aus Abb. 2-9 bereits bekannte) Konstellation, in welcher Delegierungen mittels X.509-konformen Attributszertifikaten festgehalten wurden.

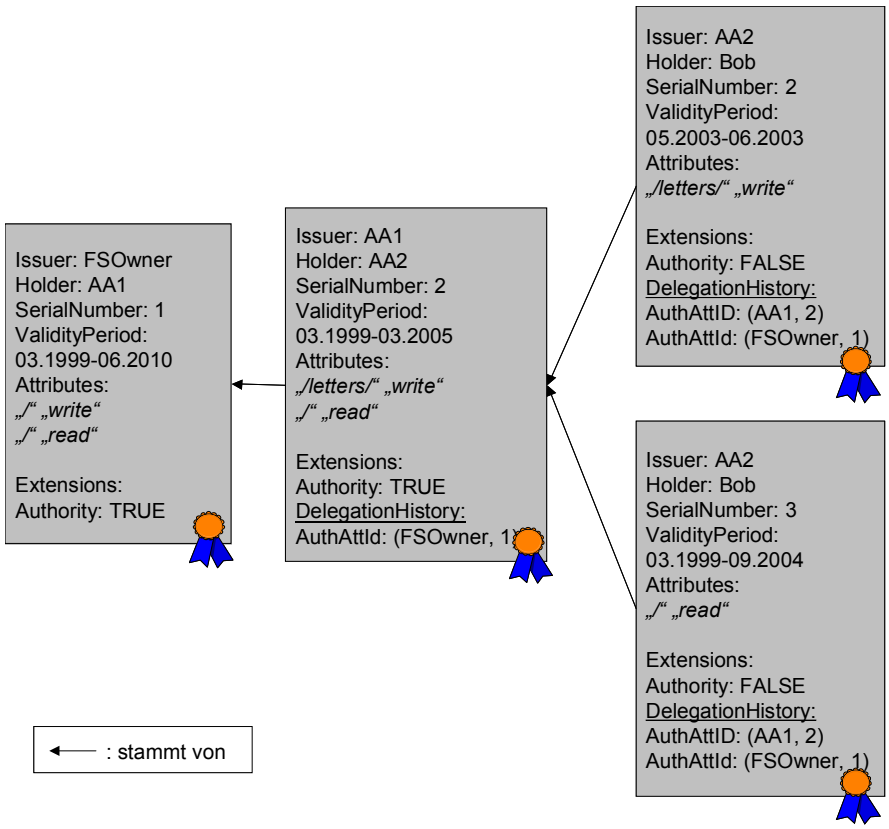


Abb. 6-1: In Attributszertifikaten festgehaltene Delegierungen

Angenommen, die beiden für *Bob* ausgestellten Zertifikate (siehe Abb. 6-1 rechts) wurden bereits von einem Verifizierer geprüft, der ausschließlich *FSOwner* als vertrauenswürdig, d.h. als *Source of Authority* (SOA) anerkennt. Bei der Prüfung wurde festgestellt, dass diese Zertifikate zwar korrekt und gültig sind, sie aber nicht von *FSOwner* sondern von einem unvertrauten „Makler" (*Attribute Authority*, AA) namens *AA2* (siehe das *Issuer*-Feld) signiert wurden.
Um den evtl. vertrauten Ursprung dieser Zertifikate zu finden, müssen in einem ersten Schritt die delegierungsbezogenen Erweiterungsfelder (*Extensions*) der beiden Zertifikate geprüft werden. Aufgrund des (in den beiden Zertifikaten identischen) Eintrags *AuthAttId* kann sich der Verifizierer an die Zertifikatsdatenbank wenden und dort die SQL-Anfrage

*SELECT * FROM AttrCerts WHERE Issuer="AA1" AND SerialNumber=2;*

absetzen. Aus Übersichtlichkeitsgründen wird hier angenommen, dass sämtliche in diesem System ausgestellten Attributszertifikate in einer relationalen Datenbank als Elemente der Relation *AttrCerts* abgelegt sind.
Erst nach Prüfung des daraufhin erhaltenen Zertifikats (siehe Abb. 6-1 Mitte) kann durch eine weitere an die Datenbank gerichtete Anfrage

*SELECT * FROM AttrCerts WHERE Issuer="FSOwner" AND SerialNumber=1;*

das von *FSOwner* signierte und daher vertrauenswürdige Attributszertifikat (siehe Abb. 6-1 links) ermittelt und anschließend überprüft werden. Da dieses Zertifikat einen vertrauten Ursprung der beiden Zertifikate von *Bob* darstellt, werden diese Zertifikate schließlich akzeptiert und ihre Inhalte (*Attributes*) in die Zugriffskontrolle einbezogen.
Die durchschnittlich erwarteten Kosten einer solchen Ursprungsprüfung, welche anhand von im X.509-Standard [ITU01] vorgesehenen Daten möglich ist, lassen sich wie folgt festhalten:

- **Kommunikationskosten:** Zum Einen muss die jeweilige Datenbank mehrfach nach Attributszertifikaten gefragt werden, die dort entsprechend gesucht werden müssen. Das Absetzen und Beantworten dieser Datenbankanfragen führen zu unerwünschten Verzögerungen innerhalb der Zugriffskontrolle. Entstand ein Zertifikat durch Delegierungen der Tiefe *t*, fällt der Aufwand im Bereich *O(t)*. Im obigen Beispiel waren zwei Datenbank-Anfragen erforderlich, um den Ursprung der beiden durch Delegierung entstandenen Attributszertifikate von *Bob* zu finden. Hinzu kommen die zur Überprüfung der ermittelten Zertifikate notwendigen Datenbankanfragen, wie beispielsweise das Anfordern von Zertifizierungspfaden oder eben Sperrlisten, je nach eingesetzter Technik. Die Anzahl solcher Datenbankanfragen entspricht wiederum *O(t)*.

- **Berechnungskosten:** Zum Anderen müssen von unterschiedlichen in der Regel nicht vertrauenswürdigen Stellen ausgestellte Attributszertifikate überprüft werden. Dies bedarf im Schnitt *O(t)* Signaturprüfungen, was ebenfalls zu langen Wartezeiten im System führen kann. Im obigen Beispiel müssen zwei Signaturen geprüft werden, unter der Annahme, dass sowohl *AA1* als auch *FSOwner* ihre eigens ausgestellten Zertifikate mit Hilfe eines I-CVT signierten.

Nachfolgend werden verschiedene Verfahren zur Reduzierung dieses Aufwands vorgestellt und bewertet.

6.1 Festhalten der Delegierungshistorie in den Attributszertifikaten

Im oben behandelten Beispiel musste die Zertifikatsdatenbank mehrfach kontaktiert werden, um den evtl. vertrauten Ursprung eines durch Delegierung entstandenen Zertifikats zu finden. Der Grund hierfür ist, dass durch das hierfür im X.509-Standard vorgesehene Zertifikatsfeld *AuthAttId* nur Zertifikate des (direkten) Delegierungsgebers (*Issuer*) referenziert werden können. (Siehe auch Kap. 2.3.2 und [ITU01]). Erst nach dem Erhalten dieser Zertifikate kann nach den die nächste Stufe innerhalb der Delegierungshistorie bildenden Zertifikaten gefragt werden. Die Folge sind erhöhte Kommunikationskosten zwischen den Verifizierern und der Zertifikatsdatenbank (siehe oben).

Ein Ansatz zur Reduzierung dieses Aufwands ist das Festhalten der eigenen, kompletten Delegierungshistorie in den Attributszertifikaten [NEA02]. In Abb. 6-2 werden die nun um ihre Delegierungshistorie erweiterten, aus Abb. 6-1 bekannten Attributszertifikate gezeigt.

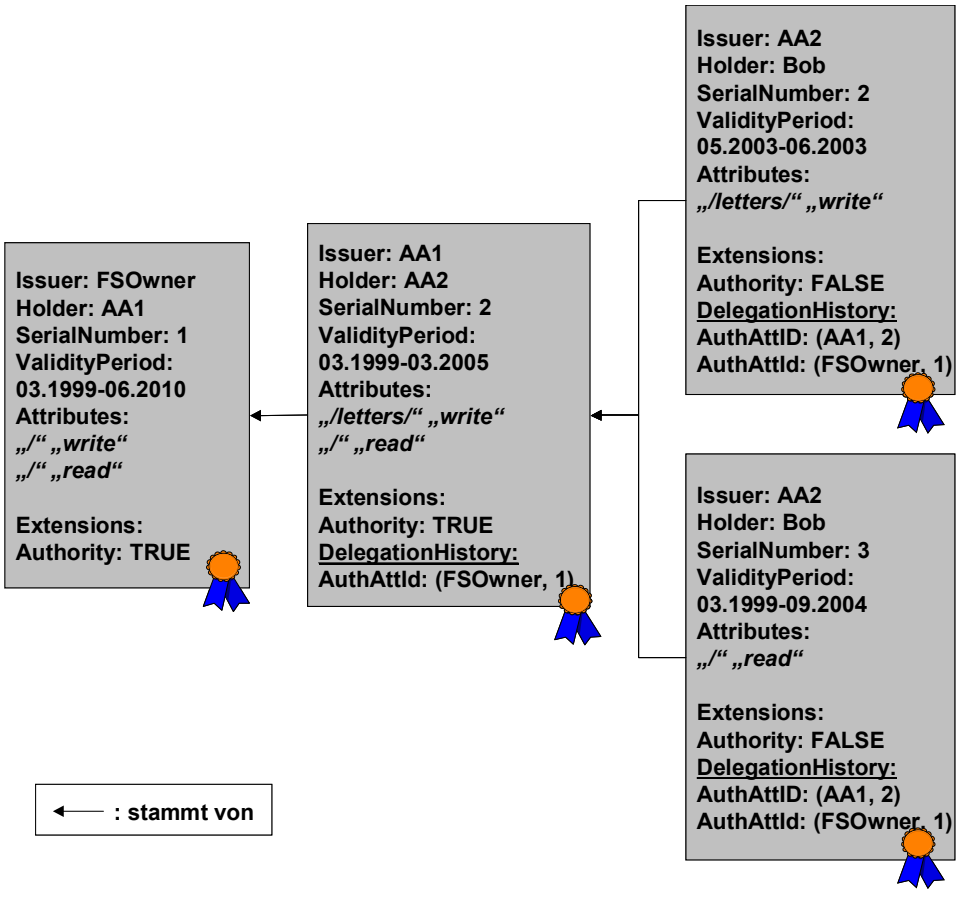


Abb. 6-2: Einfügen der Delegierungshistorie in die Attributszertifikate

Der unter dem *Extensions*-Feld platzierte Eintrag mit dem Namen *DelegationHistory* enthält hier jeweils eine Liste (Sequenz) von *k* *AuthAttId*-Einträgen aus welchen die vorher getätigten insgesamt *k* Delegierungen (genauer die dabei entstandenen Zertifikate) entnommen werden können.

Angenommen, der Ursprung der beiden für *Bob* ausgestellten Zertifikate (siehe Abb. 6-2 rechts) muss überprüft werden. In Kenntnis der *AuthAttId*-Einträge *(AA1, 2)* und *(FSOwner, 1)* kann nun ein Verifizierer bei der Zertifikatsdatenbank eine einzige, d.h. *O(1)* Anfrage nach den passenden Zertifikaten der jeweiligen Delegierungsgeber (hier *AA1* und *FSOwner*) absetzen. Durch die reduzierte Anzahl der benötigten Datenbankanfragen lässt sich hier also ein Performanzgewinn erkennen. Das Konzept ist ebenso hilfreich, wenn ein vorliegendes Zertifikat seinen Ursprung nicht in einer als vertrauenswürdig eingestuften Stelle hat. Da dies aus der eingefügten *DelegationHistory* direkt zu entnehmen ist, müssen keine weiteren Datenbankanfragen getätigt werden.

Die in Abb. 6-2 gezeigte Situation kann als ein Spezialfall für Delegierungen angesehen werden. Die dort gezeigten Zertifikate enthalten nämlich jeweils eine Teilmenge der Rechte (*Attributes*) aus genau einem Vorgängerzertifikat des jeweiligen Delegierungsgebers (*Issuer*).
Beispielsweise entstand das von *AA1* an *AA2* ausgestellte Zertifikat (siehe Abb. 6-2 Mitte), indem eine Teilmenge der Rechte aus dem von *FSOwner* an *AA1* ausgestellten Zertifikat (siehe Abb. 6-2 links) delegiert wurde. Der X.509-Standard [ITU01] erlaubt grundsätzlich auch die Delegierung der Vereinigungsmenge von in verschiedenen Zertifikaten abgelegten Rechten (siehe auch Kap. 2.3.2), indem für den Delegierungsnehmer ein einziges neues Zertifikat ausgestellt wird.

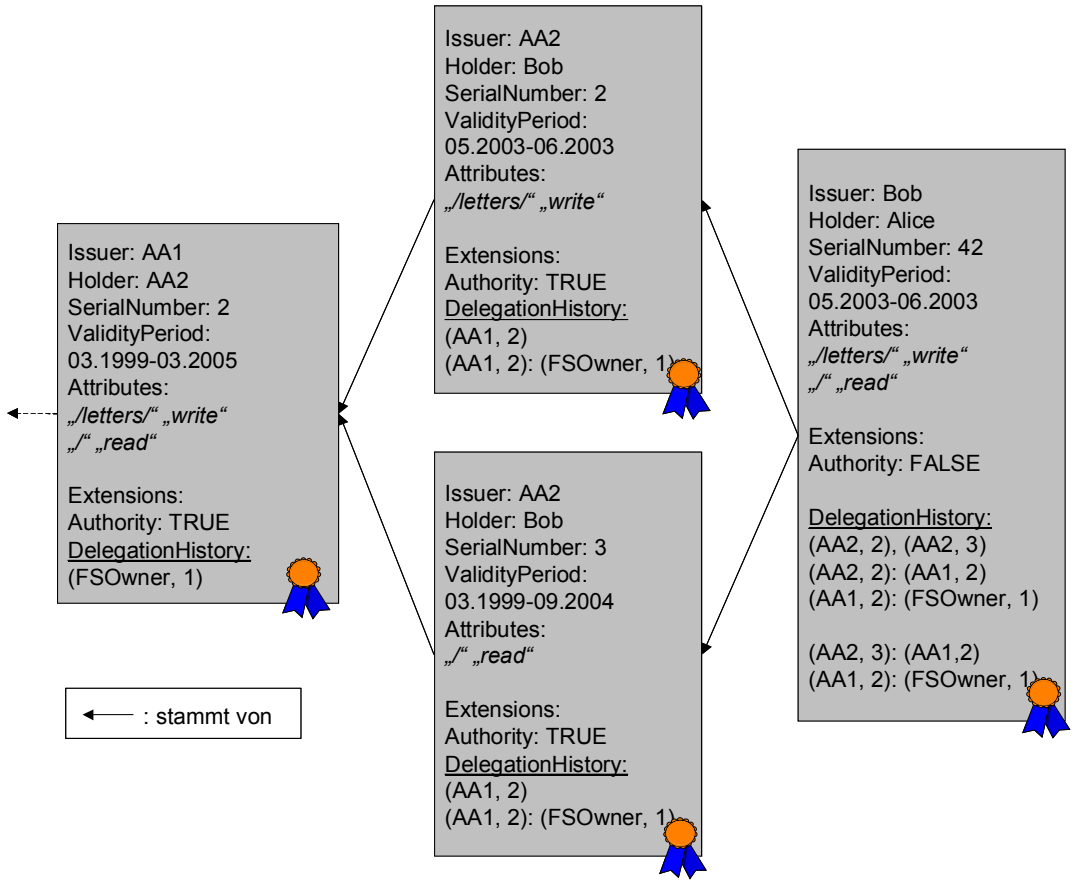


Abb. 6-3: Delegierung einer Vereinigungsmenge von Rechten

In Abb. 6-3 wird das in diesem Sinne erweiterte Beispielszenario aus Abb. 6-2 gezeigt. Hierbei wurde auf die Darstellung des von *FSOwner* an *AA1* ausgestellten Zertifikats (siehe Abb. 6-2 links) aus Platzgründen verzichtet.

Wie man sieht, hat hier *Bob* seine in zwei von *AA2* ausgestellten Attributszertifikaten (siehe Abb. 6-3 Mitte) abgelegten Rechte an *Alice* delegiert. Dabei hat er *Alice* ein einziges Attributszertifikat (siehe Abb. 6-3 rechts) ausgestellt, welches die Vereinigungsmenge der weitergegebenen Rechte enthält.
Eine Folge dieser Art Delegierung ist, dass das Zertifikat von *Alice* nun zwei (direkte) Vorgängerzertifikate hat. Um die dadurch verzweigte Delegierungshistorie dieses Zertifikats festzuhalten, muss hier durch den Eintrag *DelegationHistory* die Delegierungshistorie der beiden von *AA2* für *Bob* ausgestellten Zertifikate festgehalten werden.
In Abb. 6-3 werden diese Zertifikate durch die entsprechenden *AuthAttId*-Einträge *(AA2, 2)* und *(AA2, 3)* unterhalb des Eintrags *DelegationHistory* in *Alice*' Zertifikat repräsentiert. Wie man sieht, enthält das Zertifikat von *Alice* entsprechend Einträge zum (gemeinsamen) Ursprung dieser beiden Zertifikate, deren Delegierungshistorie keine weitere Verzweigungen enthält.
Im Allgemeinen kann es natürlich vorkommen, dass sämtliche Vorgängerzertifikate eines gegebenen Zertifikats über eine (mehrfach) verzweigte Delegierungshistorie verfügen. Angenommen, es wurden in einem System keine Rechte an frühere Delegierungsgeber „zurückdelegiert", kann eine solche (azyklische) verzweigte Struktur als ein Delegierungsbaum (engl. *Delegation Tree*) bezeichnet werden.
Entsteht ein Zertifikat durch Delegierungen ohne Verzweigung, kann der zugehörige Delegierungsbaum bestehend aus einem einfachen Pfad als eine Delegierungskette (engl. *Delegation Chain*) bezeichnet werden. Die Delegierungshistorie sämtlicher in Abb. 6-2 gezeigter Zertifikate ist somit als eine Delegierungskette zu bezeichnen.

Der Delegierungsbaum eines Zertifikats kann mittels unterhalb von *DelegationHistory* platzierten geschachtelten Sequenzen von *AuthAttId*-Einträgen zu den Vorgängerzertifikaten festgehalten werden. Hierfür werden v geschachtelte Sequenzen benötigt, um die Delegierungshistorie von den v direkten Vorgängerzertifikaten festzuhalten.
Zu einem jeden dieser Zertifikate werden wiederum $w_1, w_2, ..., w_v$ geschachtelte Sequenzen benötigt, um die Delegierungshistorie deren direkter Vorgänger zu erfassen usw.. Beispielsweise wird das Feld *DelegationHistory* des Zertifikats von *Alice* aus Abb. 6-3 den folgenden Wert enthalten:

```
DelegationHistory=
                    {
                        {(AA2, 2);
                            {(AA1, 2);
                            {(FSOwner, 1)}
                            }
                        };

                        {(AA2, 3);
                            {(AA1, 2);
                            {(FSOwner, 1)}
                            }
                        }
                    }
```

Hierbei wurden die zu einer „Vorgängergeneration" gehörenden Zertifikate durch ihre (entsprechend eingerückte) Identifikatoren *AuthAttId* erfasst.

Anhand dieser Information kann also die komplette Delegierungshistorie der beiden durch *(AA2, 2)* respektive *(AA2, 3)* gekennzeichneten Vorgängerzertifikate zu *Alice*' Zertifikat ermittelt werden.
Mit Hilfe der in diesem Abschnitt eingeführten Zertifikatserweiterungen können vor allem die erwarteten Kommunikationskosten während der Ursprungsprüfung eines Zertifikats reduziert werden. Ein Nachteil dieser Vorgehensweise ist, dass sie nicht konform zum X.509-Standard ist. Es wird hier zwar auf die in [ITU01] vorgeschriebenen Rückverweise (*AuthAttId*) in einer unveränderten Form zurückgegriffen, aber diese werden als Bestandteile eines in dieser Form nicht standardisierten Erweiterungsfelds *DelegationHistory* behandelt.

6.2 Online-Dienste zur Unterstützung der Ursprungsprüfung

Eine weitere denkbare Maßnahme für die Entlastung eines Verifizierers und damit die Beschleunigung der gesamten Zugriffskontrolle ist der Einsatz bei der Ursprungsprüfung von Zertifikaten spezialisierter externer Systeme. Die sinnvollen Einsatzbereiche für diese liegen in der Rekonstruktion und/oder Prüfung der Delegierungshistorie von Attributszertifikaten. Im Folgenden werden die Chancen zur Realisierung sowie der betrieblichen Konsequenzen dieser Vorgehensweise diskutiert.
Um die Ursprungsprüfung von Zertifikaten zu unterstützen, welche (im Einklang mit dem X.509-Standard [ITU01]) ihre komplette Delegierungshistorie nicht enthalten, ist der Aufbau einer effizienten „Suchmaschine" denkbar (siehe auch [Nik99]). Diese kann auf Anfrage eines Verifizierers die evtl. verteilt betriebenen, aus technischer Sicht heterogenen Datenablagen der einzelnen Zertifizierungsstellen nach den entsprechenden Zertifikaten durchsuchen und die gefundenen Zertifikate an den Verifizierer ausliefern.
Hierbei kann die Suchmaschine die jeweilig benötigten Datenbankanfragen anhand des weiter oben bereits behandelten Felds *AuthAttId* bereits gefundener Zertifikate generieren. Um die Suche nach der Delegierungshistorie eines bestimmten Attributszertifikats starten beziehungsweise stoppen zu können, muss die Anfrage des Verifizierers das *AuthAttId*-Feld aus dem gefragten Zertifikat respektive den Namen der jeweilig vertrauten Stelle (SOA) enthalten.

In einem System, welches auf der I-CVT-Technik aufbaut, kann die Suchmaschine die einzelnen Datenbanken gleich die passenden Zertifizierungspfade generieren lassen und diese in die Antwortnachricht einfügen. In Kombination mit einem in Kap. 3.2.3 diskutierten DPD-Dienst können auch die zur Signaturprüfung der einzelnen Zertifizierungspfade benötigten Schlüsselzertifikate sowie die zu deren Prüfung benötigten Daten übermittelt werden. Eine auf diese Weise funktionierende Architektur stellt Abb. 6-4 schematisch dar.
Die Verwendung eines solchen Dienstes kann verschiedene Vorteile in einem System haben. Zum Einen wird die Implementierung der einzelnen Verifizierer einfacher, denn diese müssen keine komplizierteren Suchfunktionen und evtl. verschiedene Datenbanksprachen unterstützen. Zum Anderen werden durch die Einführung eines solchen Suchdienstes keine Vorschriften des X.509-Standards bezüglich der Formatierung von Attributszertifikaten verletzt.
Außerdem kann der systemweite Kommunikationsaufwand durch eine Zwischenspeicherung der jeweilig gefundenen Zertifikate seitens der Suchmaschine reduziert werden.

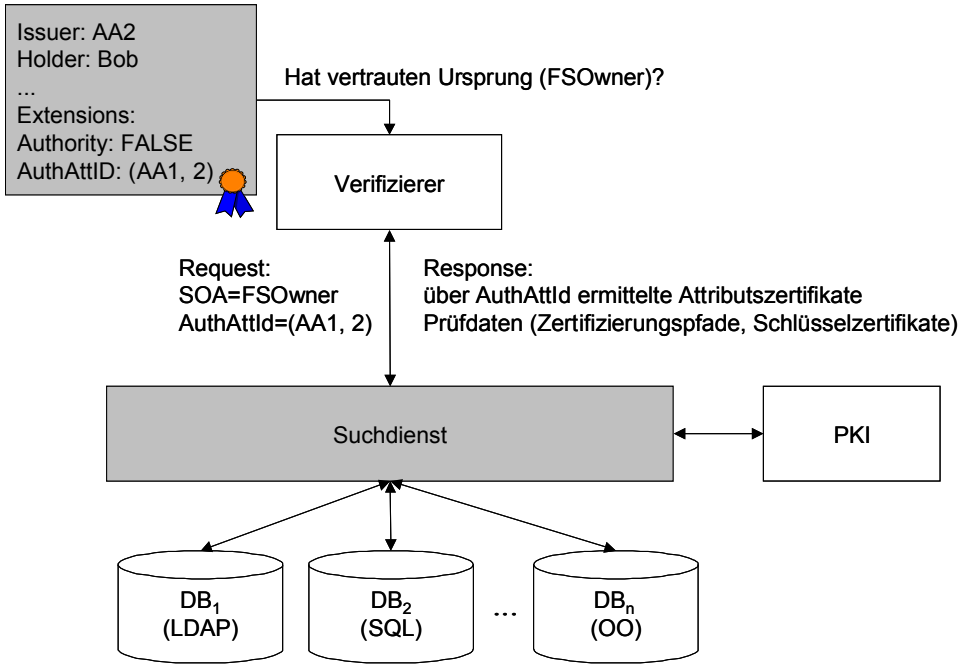


Abb. 6-4: Dienst zur Suche nach der Delegierungshistorie von Zertifikaten

Beispielsweise wird während der Ermittlung der Delegierungshistorie der beiden in Abb. 6-1 gezeigten Zertifikate von *Bob* auch die Delegierungshistorie des für *AA2* ausgestellten Zertifikats, welches ein Vorgänger von *Bobs* Zertifikaten ist, bestimmt. Wird nun der Dienst (bspw. durch einen anderen Verifizierer) nach der Delegierungshistorie dieses Zertifikats gefragt, kann er die diese bildenden Zertifikate sowie die weiteren zugehörigen Daten ohne weitere Datenbankanfragen aushändigen.

Alternativ oder als Ergänzung zu einer Suchmaschine kann der komplette Prozess der Ursprungsprüfung einem hierfür spezialisierten (zentralen) Dienst überlassen werden. Dieser Dienst prüft auf Anfrage eines Verifizierers, der dem Dienst das gefragte Attributszertifikat übersendet, ob dieses Zertifikat ursprünglich von einer bestimmten durch den Verifizierer genannten vertrauenswürdigen Stelle stammt.
Das Urteil des Dienstes kann als *„good“ „not good“* oder *„unknown“* ausfallen, je nach Ergebnis der Untersuchungen. Selbstverständlich sollten die Antworten eines solchen Prüfdienstes eindeutig auf ihn zurückzuführen sein, was die Verwendung von digital signierten Antwortnachrichten (siehe auch die Diskussion von OCSP in Kap. 3.2.1) impliziert.
Dieses Konzept entlastet die Verifizierer, die nun nur noch die signierten Antworten des Prüfdienstes validieren müssen. Allerdings werden die möglichen Anwendungen nun durch das einem solchen Prüfdienst entgegenzubringende Vertrauen eingeschränkt. Der Prüfdienst kann nämlich beliebige Attributszertifikate als (in-)akzeptabel erklären. Dadurch kann dieser Dienst einen großen Einfluss auf die Zugriffskontrolle innerhalb der abgesicherten Systeme gewinnen, was ihn wiederum zu einem begehrten Angriffsziel machen kann. Außerdem muss im Auge behalten werden, dass beim etwaigen Ausfall eines zentralisierten Dienstes keine Ursprungsprüfung mehr im System möglich wird.

6.3 Zertifikatsreduktion

Um den Aufwand der Ursprungsprüfungen in einem System zu senken, wurde in [Aur99] die Technik der so genannten Zertifikatsreduktion (engl. *Certificate Reduction*) vorgeschlagen. Betrachtet wurden dabei zum Standard SPKI [EFL+99] konforme, so genannte Delegierungszertifikate (engl. *Delegation Certificates*). Die wesentlichen Erkenntnisse sind ohne weiteres auf andere Zertifikatsformate, wie z.B. X.509 übertragbar.
Um die Funktionsweise der Zertifikatsreduktion leichter verständlich zu machen, wird auf eine Darstellungsform der in einem System getätigten Delegierungen zurückgegriffen, die von der bisherigen Darstellungsform abweicht. Diese Darstellungsform ist ein gerichteter Graph, der sog. Delegierungsgraph (engl. *Delegation Graph*) [Aur98a] [Aur98b] [Aur99]. Die Knoten dieses Graphen symbolisieren jeweils einen Benutzer im System. Die gerichteten Kanten des Graphen stehen jeweils für eine getätigte Delegierung, welche zur Ausstellung eines Zertifikats führte. Sie zeigen vom jeweiligen Delegierungsgeber zum Delegierungsnehmer und sind mit den jeweilig delegierten Rechten markiert. Ein zwischen zwei Knoten des Graphen existierender gerichteter Pfad wird als Delegierungspfad (engl. *Delegation Path*) bezeichnet.

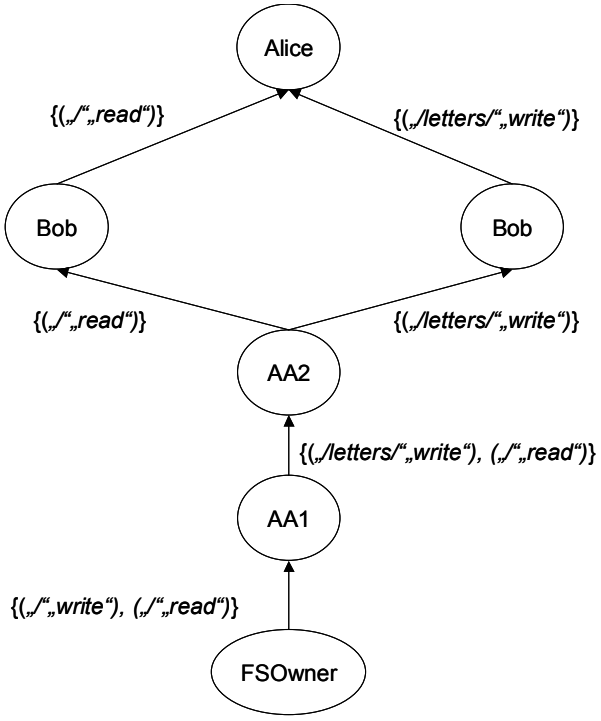


Abb. 6-5: Beispiel für einen Delegierungsgraphen

In Abb. 6-5 wird der aus den in Abb. 6-2 beziehungsweise in Abb. 6-3 behandelten mehrstufigen und verzweigten Delegierungen hervorgehende Delegierungsgraph gezeigt. Es gibt hier beispielsweise den Delegierungspfad *FSOwner*→*AA1*→*AA2* zwischen den beiden Benutzern *FSOwner* und *AA2*.

Die Zertifikatsreduktion besteht im Wesentlichen darin, die entlang eines Delegierungspfads $A \rightarrow B \rightarrow ... \rightarrow Y \rightarrow Z$ getätigten Delegierungen durch ein einziges neues Zertifikat zu repräsentieren. Ein durch Reduktion entstandenes neues Zertifikat enthält genau die Rechte, welche den letzten Kanten ($Y \rightarrow Z$) innerhalb des jeweilig betrachteten Pfads markieren.

Das neue Zertifikat wird vom durch den ersten Knoten dieses Pfads repräsentierten Benutzers (*A)* signiert. Diese Zertifikatsreduktion bedeutet natürlich nicht, dass die bereits vorhandenen (zu reduzierenden) Zertifikate aus dem System verschwinden.
Wird beispielsweise der Delegierungspfad *FSOwner→AA1→AA2* (siehe Abb. 6-5) reduziert, entsteht ein neues durch *FSOwner* an *AA2* ausgestelltes Zertifikat mit den Rechten („*/letters/*“„*write*“) und („*/*“„*read*“).

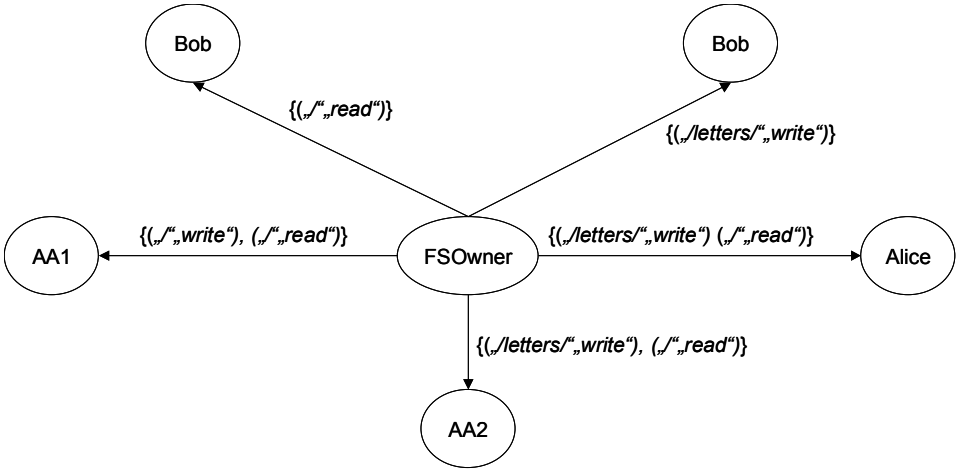


Abb. 6-6 Ausgehend von *FSOwner* reduzierter Delegierungsgraph

Werden sämtliche *k* von einer bestimmten Zertifizierungsstelle ausgehenden Delegierungspfade auf diese Weise reduziert, entstehen *k* neue Zertifikate. Ein jedes dieser *k* Zertifikate repräsentiert dabei eine mehrstufige (indirekte) Delegierung, wobei diese Zertifikate durch eine einfache Rechtevergabe entstehen.
Sinnvollerweise werden in einem System Delegierungspfade reduziert, welche ihren Anfang an einer der vertrauten Stellen (*Source of Authority*, SOA) haben. Dem entsprechend wird in Abb. 6-6 der Delegierungsgraph nach einer von *FSOwner* ausgehenden Zertifikatsreduktion dargestellt. Wie man sieht, wurden die Ergebnisse aller auffindbaren Delegierungspfade aus Abb. 6-5 in jeweils einem der insgesamt fünf durch *FSOwner* signierten Zertifikate festgelegt.
Um die Zertifikatsreduktion in einer verteilten Umgebung zu unterstützen, wurde in [Aur99] die Etablierung eines zentralen „Reduktionsdienstes“ vorgeschlagen. Dieser soll regelmäßig (z.B. periodisch) sämtliche auffindbaren und relevanten Delegierungspfade durch jeweils ein neues Zertifikat ersetzen. Nach der Reduktion aller relevanten Delegierungspfade innerhalb eines Zertifikatsbestands beziehungsweise nach der Verteilung der hierdurch entstehenden neuen Zertifikate kann auf die komplizierte und aufwändige Ursprungsprüfung innerhalb der Zugriffskontrolle verzichtet werden. Verifizierer können nämlich direkt von den jeweiligen SOAs ausgestellte Zertifikate bekommen und diese prüfen.
Es kann jedoch nicht ohne Weiteres garantiert werden, dass Verifizierer tatsächlich nur solche nach einer Reduktion entstandenen Zertifikate erhalten werden. Folglich muss ein Verifizierer weiterhin in der Lage sein, gegebenenfalls Ursprungsprüfungen durchzuführen, sprich die hierfür notwendigen Funktionen zu implementieren. Neben diesem Aspekt führt diese Vorgehensweise zu weiteren betrieblichen Problemen in einem System:

- **Wachsende Datenmenge**: Durch die Zertifikatsreduktion entstehen neue Zertifikate, die zusätzlich zum „alten“ Bestand gepflegt und gespeichert werden müssen. Dies führt zu einer Aufwandserhöhung vor allem seitens der Datenablagesysteme.

- **Aufwändige Suche:** Das algorithmische Auffinden von evtl. sämtlichen Pfaden in einem gerichteten Graphen kann leicht kostenintensiv werden. Da potenziell ein jeder der n Benutzer eines Systems sowohl ein vertrauenswürdiger Delegierungsgeber als auch ein Delegierungsnehmer sein kann [ITU01], wird die Anzahl der aufzufindenden und bearbeitenden Delegierungspfade in den Bereich von $O(n^2)$ fallen. Angenommen, der gesamte Zertifikatsbestand wird in einer zentralen stets verfügbaren Ablage gespeichert, können die einzelnen Zertifikate entlang eines Pfades (relativ) schnell gefunden werden. Um hier die Anzahl der benötigten Datenbankanfragen zu drücken, können die in Kap. 6.1 vorgestellten Zertifikatserweiterungen (*DelegationHistory*) nützlich sein.
  Liegen aber die Attributszertifikate räumlich verteilt vor, beispielsweise unter der Obhut deren Besitzer (siehe Client-Push-Modell), wird die Reduktion aus Erreichbarkeitsproblemen der einzelnen Datenquellen gefährdet.

- **Signieren der neuen Zertifikate:** Eine wichtige Frage ist, durch wen die neuen Zertifikate, d.h. die Ergebnisse der Reduktion, signiert werden sollen. Prädestiniert sind hierfür die einzelnen Zertifizierungsstellen, von denen die einzelnen Pfadreduktionen jeweils ausgehen. Dementsprechend muss der zentrale Reduktionsdienst die Ergebniszertifikate bei den passenden Stellen zum Signieren einreichen. Ein damit verbundenes Problem ist, dass die Zertifizierungsstellen bezüglich der Inhalte dieser nun zu signierenden Zertifikate leicht getäuscht werden können. Wird nämlich der Delegierungspfad $A \rightarrow B \rightarrow ... \rightarrow Y \rightarrow Z$ reduziert, so reicht der Reduktionsdienst die Inhalte der letzten durch Y getätigte $Y \rightarrow Z$ Delegierung bei der Zertifizierungsstelle A zum Signieren ein. Da A von der Delegierung $Y \rightarrow Z$ zuvor kein Kenntnis hatte, muss er den Reduktionsdienst bezüglich der eingereichten Inhalte als vertrauenswürdig einstufen. Dies impliziert wiederum, dass die neuen Zertifikate auch gleich vom zentralen Reduktionsdienst „eigenhändig“ ausgestellt (signiert) werden können.
  Um eine derartige Machtkonzentration im System zu verhindern, kann natürlich eine jede Zertifizierungsstelle einen eigenen Reduktionsdienst implementieren. Dieser kann die von dieser Stelle ausgehenden Delegierungspfade regelmäßig reduzieren, wobei die hieraus resultierenden Ergebniszertifikate durch die Zertifizierungsstelle signiert werden können. Auf diese Art und Weise kann das Einführen einer quasi allmächtigen zentralen Instanz in das System vermieden werden.

- **Handhabung von Sperrungen:** Auch die frühzeitige Sperrung von Delegierungen bereitet einige Probleme. Wird eine Delegierung $K \rightarrow L$ innerhalb eines zuvor reduzierten Delegierungspfads $A \rightarrow B \rightarrow ... \rightarrow K \rightarrow L \rightarrow ... \rightarrow Y \rightarrow Z$ widerrufen, so bleibt dies bis zur nächsten Reduktion unwirksam. Entzieht beispielsweise *AA2* die an *Bob* delegierten Rechte (siehe Abb. 6-5), so verschwindet die eine Grundlage der beiden in Abb. 6-6 gezeigten durch *FSOwner* an *Bob* ausgestellten „reduzierten“ Zertifikate. Dies kann leicht zu Konfliktsituationen innerhalb der Zugriffskontrolle führen.

Angesichts dieser sowie der in [Kor02] und [KHS00] diskutierten weiteren Aspekte erweist sich die regelmäßige nachträgliche Zertifikatsreduktion als eine problematische Vorgehensweise zur Vereinfachung der Ursprungsprüfung.

6.4 Technik der Offline-Delegierung

Motiviert durch die in Kap. 6.3 diskutierten Probleme, welche die Umsetzung einer nachträglichen Reduktion von Delegierungspfaden erschweren, wurde die als „Offline geprüfte Delegierung“ oder kurz „Offline-Delegierung“ bezeichnete Vorgehensweise entwickelt [NOA03] [NA02a] [NA02b] [Oet02].
Die Kernidee lässt sich folgendermaßen zusammenfassen (siehe Abb. 6-7): Der Zugriffskontrolle bzw. den Verifizierern werden ausschließlich durch die jeweiligen Ressourcenbesitzer (SOA) ausgestellte Attributszertifikate zugänglich gemacht, auch wenn diese Zertifikate im Zuge von Delegierungen entstehen.
Angenommen ein Benutzer hat bereits Zugriffsrechte zu bestimmten Ressourcen bzw. die entsprechenden von der jeweiligen SOA ausgestellten Zertifikate erhalten. Möchte dieser Benutzer einen Teil seiner Rechte delegieren, muss er seine Absicht der jeweiligen SOA auf eine gesicherte Art und Weise (in Form eines signierten Antrags) melden. Diese SOA kann den Antrag überprüfen und diesen genehmigen oder ablehnen. Wird der Antrag genehmigt, stellt die SOA die jeweils benötigten Attributszertifikate aus und macht diese den Verifizierern und/oder dem jeweiligen Delegierungsnehmer zugänglich.

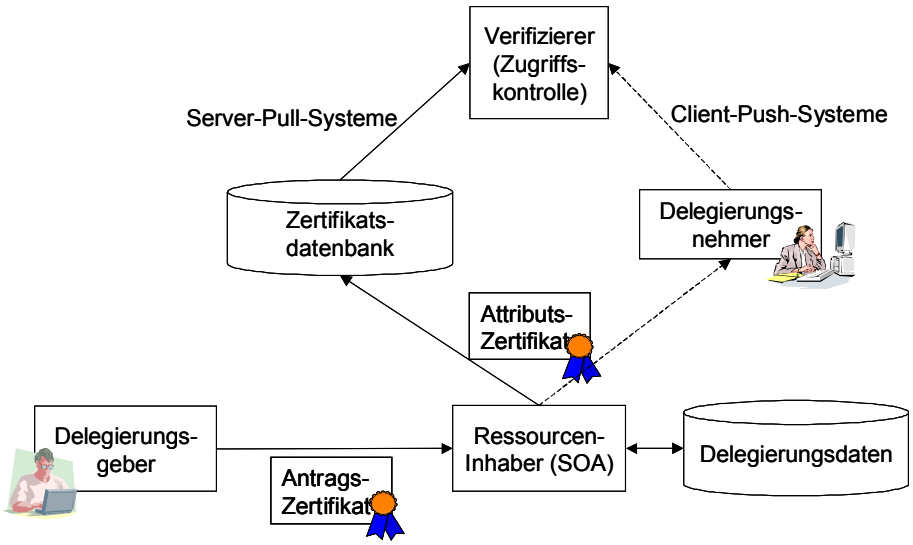


Abb. 6-7: Schematischer Ablauf der Offline-Delegierung

Ein solches durch Offline-Delegierung entstandenes Zertifikat kann durchaus Informationen (z.B. Rückverweise) zu seinen Vorgängern enthalten. Diese Informationen sind jedoch aus Sicht eines Verifizierers bzw. der Zugriffskontrolle irrelevant. Liegt nämlich einem Verifizierer ein bestimmtes Attributszertifikat vor, so wird nach der Signaturprüfung klar, ob dieses Zertifikat von einer vertrauenswürdigen Stelle (SOA) ausgestellt wurde und folglich akzeptiert werden *muss*.
Von anderen Stellen ausgestellte Zertifikate können ohne weitere Prüfschritte abgelehnt werden, da diese ihren Ursprung nun auf keinen Fall in einer vertrauten SOA haben können. Die Bedingung hierfür ist, dass sämtliche Zertifizierungsstellen in einem System gemäß der Offline-Delegierung arbeiten.

Diese Tatsache impliziert, dass auf den Prozessschritt der Ursprungsprüfung (siehe Kap. 2.3.3) innerhalb der Zertifikatsprüfung gänzlich verzichtet werden kann, selbst dann wenn im System die mehrstufige Delegierung der Rechte erlaubt ist.
Den diesbezüglich modifizierten, vereinfachten Prozess der Zertifikatsprüfung zeigt Abb. 6-8.

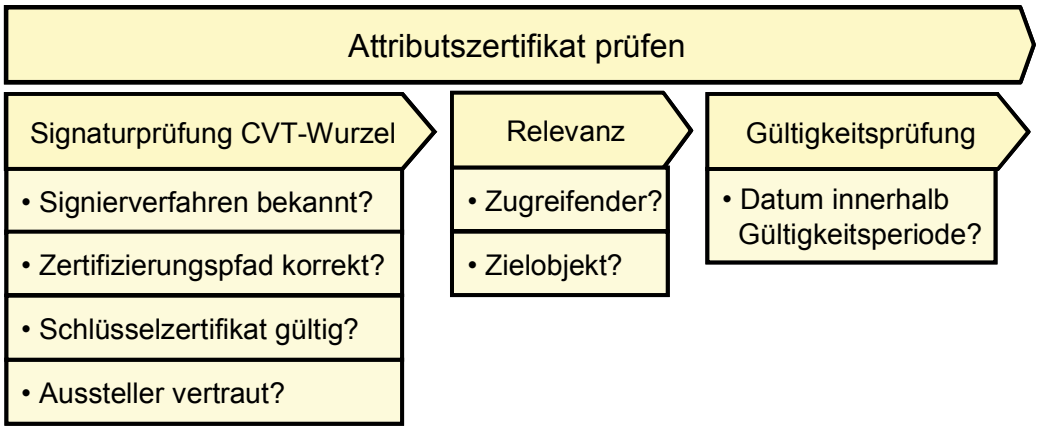


Abb. 6-8: Durch die Offline-Delegierung vereinfachte Zertifikatsprüfung

Hierbei wurde angenommen, dass die Zertifikate im System mit Hilfe von I-CVTs verwaltet werden (vgl. Abb. 4-4). Durch diese Vereinfachung wird die Prüfung der Zertifikate und damit die gesamte Zugriffskontrolle grundsätzlich effizienter, ohne auf die Möglichkeit der Rechtedelegierung (siehe Def. 2-8) im System verzichten zu müssen.
Um den genaueren Ablauf der Offline-Delegierung zu erklären, wird auf die weiter oben in Abb. 6-1 gezeigten Beispieldelegierungen zurückgegriffen. Angenommen *AA1*, der von *FSOwner* (SOA) bereits ein Zertifikat (siehe Abb. 6-1 links) mit den Rechten *(„/“„write“)* und *(„/“„read“)* erhielt, möchte einen Teil der Rechte an *AA2* delegieren. *AA1* stellt hierfür ein digital signiertes so genanntes Antragszertifikat AZ_{AA1} aus.
Ein Antragszertifikat enthält das aktuelle Datum (*Date*), die Uhrzeit (*Time*), den Namen des Antragstellers (Delegierungsgeber *Grantor*), den Namen des beabsichtigten Delegierungsempfängers (*Target-User*), den Namen des künftigen Zertifikatsaustellers (*Issuer*), die weiterzugebenden Rechte (*Attributes*) und eine beabsichtigte Gültigkeitsperiode (*ValidityPeriod*) für die Delegierung. Beispielweise kann *AA1* den Antragszertifikat

$AZ_{AA1}=D_{AA1}$*(h(Date: 27.2.1999, Time: 12:35:22, Grantor:„AA1“, Target-User: „AA2“, Issuer: „FSOwner“, Attributes: {(„/letters/“„write“) („/“„read“)}, ValidityPeriod: {NotBefore: 01.03.1999; 23:00, NotAfter: 01.03.2005; 12:00}))*.

an *FSOwner* senden. Hierfür muss ein zuvor vereinbartes Kommunikationsmedium, wie z.B. das Internet benutzt werden. Ist der Vertraulichkeitsschutz der delegierten Berechtigungen erforderlich, kann das Antragszertifikat für *FSOwner* verschlüsselt werden. Optional kann *AA1* zusätzlich das eigene Zertifikat an *FSOwner* senden, welches die zu delegierenden Rechte enthält.
Nach Erhalt des Antragszertifikats prüft *FSOwner* zunächst, ob dieses von *AA1* signiert wurde. Hierfür wird der öffentliche Schlüssel von *AA1* benötigt, wobei die Dienste einer PKI in Anspruch genommen werden können.
Danach wird die Aktualität des Antrags geprüft (*Date, Time*), um das Einschleusen eines alten Antragszertifikats (so genannte *Replay-Attacke*) zu unterbinden.

Anschließend kann *FSOwner* gemäß eigener oder systemweiter Richtlinien prüfen, ob *AA1* (*Grantor*) überhaupt berechtigt ist, Rechte zu delegieren beziehungsweise, ob *AA2* (*Target-User*) Rechte im System erhalten kann. Dabei kann entschieden werden, ob ein zunächst unbekannter (evtl. ganz neuer) Benutzer ein Zertifikat erhalten darf. Ist z.B. *AA1* ein Administrator, der für *FSOwner* arbeitet, kann er auf diese Weise neue Benutzer (hier *AA2*) registrieren bzw. ihnen gleich Rechte zukommen lassen.
Des Weiteren kann *FSOwner* prüfen, ob *AA1* die nun zu delegierenden Rechte (*Attributes*) tatsächlich besitzt. Hierfür kann das (optional mitgeschickte) früher von *FSOwner* an *AA1* ausgestellte Zertifikat geprüft werden. Ist beispielsweise dieses Zertifikat bereits abgelaufen, kann der Antrag abgelehnt werden.
Die Ablehnung des Antrags wird durch eine digital signierte Nachricht an den Delegierungsgeber (*Grantor*) mitgeteilt werden. In diese Nachricht wird der Hashwert $h(AZ_{AA1})$ des abgelehnten Antragszertifikats sowie der Grund der Ablehnung, wie z.B. „*Granting Certificate Expired*“ eingefügt.
Läuft die Prüfung des Antragszertifikats erfolgreich ab und der Antrag wird genehmigt, wird das Antragszertifikat durch *FSOwner* archiviert (siehe das Symbol „Delegierungsdaten“ in Abb. 6-7). Ein archiviertes Antragszertifikat kann später evtl. als Nachweis zum Auslösen einer in die Tat umgesetzten Delegierung benutzt werden. Außerdem sind die gespeicherten Antragszertifikate bei der Umsetzung etwaiger Sperrungen früher getätigter Delegierungen hilfreich.
Abschließend stellt *FSOwner* ein neues Attributszertifikat für *AA2* aus (siehe Abb. 6-7) und fügt es in die den Verifizierern zugängliche Datenbank (in Server-Pull-Systemen) ein, beziehungsweise sendet es direkt dem Delegierungsnehmer, d.h. *AA2* zu (in Client-Push-Systemen). *AA2* kann danach ebenfalls als Delegierungsgeber auftreten und sich entsprechend an *FSOwner* wenden.

Es ist leicht einzusehen, dass durch diese Vorgehensweise auf der Betrachtungsebene der ausgestellten Zertifikate solche Delegierungsstrukturen entstehen, die eine nachträgliche Reduktion des Zertifikatsbestands überflüssig machen. Der Grund ist, dass nun sämtliche Zertifikate, die in die Zugriffskontrolle einfließen, durch die vertrauten Stellen (SOAs) ausgestellt werden.
Diese Tatsache wird mit Hilfe von Abb. 6-9 verdeutlicht. Die obere Hälfte von Abb. 6-9 zeigt den aus Abb. 6-5 bereits bekannten Delegierungsgraphen. Dieser Graph hält die Delegierungen fest, welche im in Abb. 6-3 behandelten Beispielsszenario getätigt wurden.
Erfolgen die durch diesen Graphen festgehaltenen Delegierungen, wie z.B. die oben im Detail erklärte Delegierung der Rechte von *AA1* an *AA2,* gemäß der Offline-Delegierung, so spannen die dabei ausgestellten Attributszertifikate den in Abb. 6-9 unten gezeigten Delegierungsgraphen auf. (Aus Übersichtlichkeitsgründen wurden die Kanten dieses Graphen zusätzlich mit dem Namen des jeweiligen Delegierungsgebers (DG) markiert.)
Beim genaueren Hinschauen kann festgestellt werden, dass dieser durch die Offline-Delegierung entstandene Delegierungsgraph und der nach der Zertifikatsreduktion des oberen Graphen entstandene und in Abb. 6-6 gezeigte Delegierungsgraph homomorphe Graphen sind (abgesehen von den hier zusätzlich dargestellten DG-Bezeichnern).

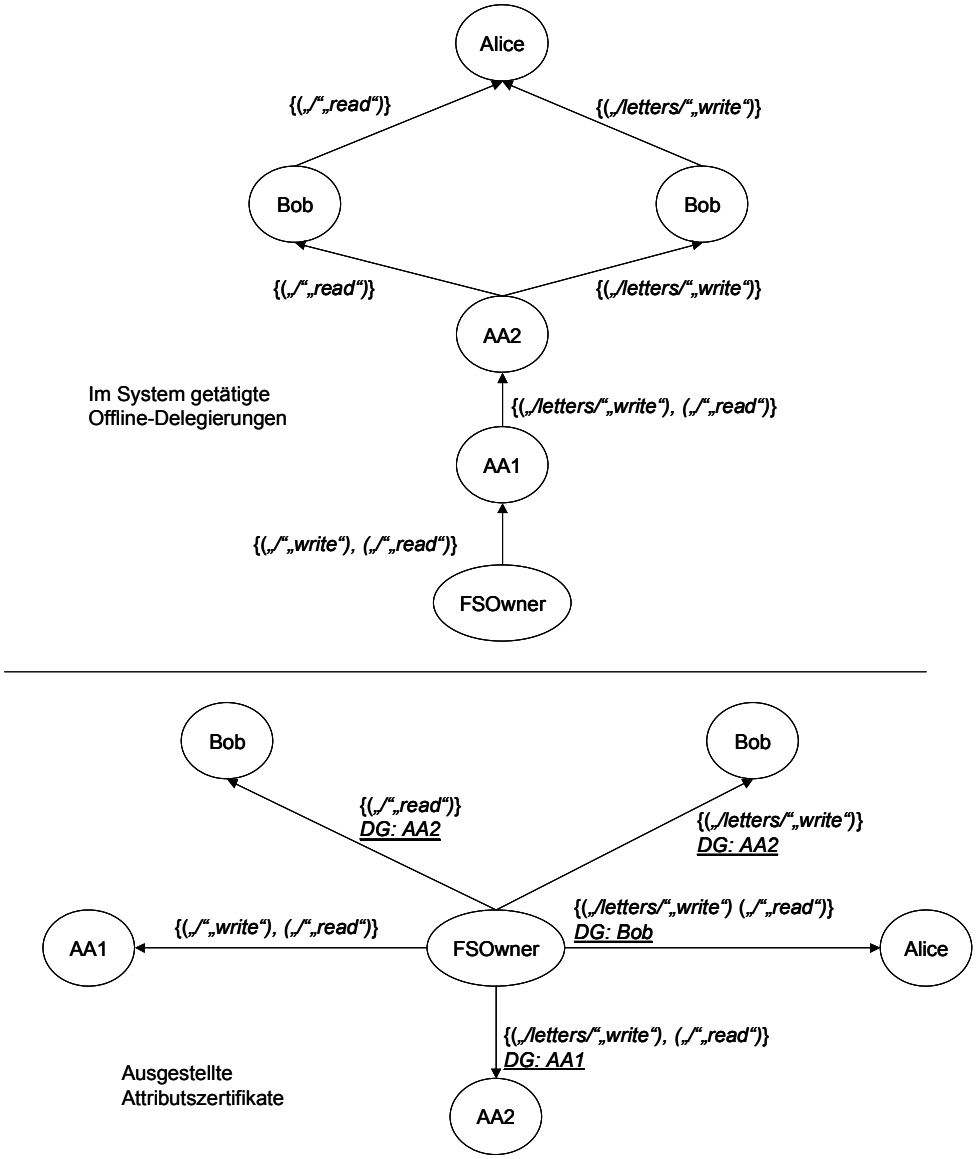


Abb. 6-9: Durch Offline-Delegierung entstehen „reduzierte“ Delegierungspfade

Das bedeutet, dass durch die Offline-Delegierung solche Zertifikate entstehen, die auch im Zuge einer nachträglichen Reduktion eines nicht durch Offline-Delegierung entstandenen Zertifikatsbestands erzeugt werden können.
Folglich kann in einem System, welches der Strategie der Offline-Delegierung folgt, auf eine nachträgliche aktive Zertifikatsreduktion ohne Weiteres verzichtet werden.

6.4.1 Methodik zur Pflege von Delegierungsinformationen

Die Technik der Offline-Delegierung vereinfacht und beschleunigt den Verifizierungsprozess von Attributszertifikaten, welche aufgrund einer Delegierung entstanden. Die hierbei entstehenden Attributszertifikate tragen nicht zwangsläufig Angaben zu deren Vorgängerzertifikaten. Das Vorhandensein solcher Informationen ist jedoch in vielen Fällen wünschenswert.
Ein Beispiel hierfür ist der Widerruf einer früher getätigten Delegierung, welche als Ergebnis ein durch den jeweiligen Ressourcenbesitzer ausgestelltes Attributszertifikat hatte. In diesem Fall muss dieses Zertifikat frühzeitig gesperrt werden. Außerdem kann es je nach internen Richtlinien erforderlich sein, sämtliche aus diesem Zertifikat hervorgegangene Delegierungen bzw. die zugehörigen Zertifikate zu sperren.

Um solche administrativen Vorgänge bezüglich der Rechtedelegierung zu unterstützen, ist es nützlich, die Delegierungshistorie der einzelnen ausgestellten Attributszertifikate in einem hierfür vorgesehenen Datenbestand festzuhalten. Hat ein bestimmter administrativer Vorgang, wie z.B. der oben erwähnte Widerruf einer Delegierung, Auswirkungen auf die Nachfolger eines bestimmten Zertifikats, so ist es ebenfalls von Interesse, diese möglichst schnell aufzufinden.
Da bei der Technik der Offline-Delegierung die Kontrolle der Delegierungen (auf der Ebene der Zertifikate betrachtet) bei den vertrauten Ressourceninhabern (SOAs) liegt, ist es sinnvoll, diese Informationen in deren Hintergrundsystemen aufzubewahren und zu pflegen. Dies wird in Abb. 6-7 durch die Datenhaltungskomponente mit der Bezeichnung „Delegierungsdaten“ verdeutlicht.
Um die Delegierungen, welche zur Ausstellung von Attributszertifikaten führten, festzulegen, wird als Basis dieses Datenbestands die so genannte Delegierungsrelation (DGR) konzipiert (siehe auch [Küh03]).
Diese Relation ist ein Quadrupel *DGR(DG, DN, ZDG, ZDN)* wobei *DG* die Menge der potenziellen Delegierungsgeber und *DN* die Menge der potenziellen Delegierungsnehmer symbolisieren.
ZDG ist die Menge der Zertifikate der Delegierungsgeber, deren Inhalte in Delegierungsvorgängen delegiert wurden. *ZDN* ist die Menge der aufgrund von Delegierungen für die Delegierungsnehmer ausgestellten Zertifikate.
Ein Tupel $delegation(dg, dn, zdg, zdn) \in DGR$ dieser Relation besagt, dass der Delegierungsnehmer *dn* sein Attributszertifikat *zdn* aufgrund eines von *dg* ausgestellten Antragszertifikats erhalten hat. Hierbei wurden die Inhalte des Attributszertifikats *zdg* delegiert, welches zuvor auf Namen von *dg* ausgestellt wurde. Im Fall von X.509-konformen Attributszertifikaten können *zdn* und *zdg* die Seriennummern (SerialNumber) der jeweiligen Zertifikate enthalten. Um *dg* und *dn* eindeutig zu identifizieren, müssen systemweit eindeutige Namen verwendet werden.

Nachfolgend werden verschiedene Aspekte der Pflege dieser Relation diskutiert. Die möglichen Operationen in DGR sind genau das Einfügen und das Löschen einzelner *delegation(dg, dn, zdg, zdn)* Tupel aus der Relation. Bei jedem erfolgreichen Delegierungsvorgang sollte mindestens ein neues Tupel in *DGR* eingefügt werden. Beim Rückzug einer Delegierung, wenn das mit dem Widerruf von r ($r>0$) Attributszertifikaten verbunden ist, müssen mindestens r Tupel aus *DGR* gelöscht werden.
Wie aus der obigen Definition der Delegierungsrelation hervorgeht, beinhaltet diese keine Informationen zu den in den einzelnen Delegierungsvorgängen weitergegebenen Rechten. Sie stellt lediglich den Bezug zwischen den Akteuren und deren Zertifikaten her. Einerseits kann dadurch die Delegierungsrelation unabhängig von der jeweiligen Anwendungssemantik in einem System eingesetzt werden. Andererseits müssen die in Abb. 6-10 gezeigten als nicht trivial anzusehenden Delegierungsvorgänge entsprechend behandelt werden:

- **Delegierung von gleichen Rechten aus mehreren in genau ein Zertifikat**: Ein Delegierungsgeber *dg*, der eine Teilmenge T seiner Rechte R ($T \subseteq R$) an einen Delegierungsnehmer *dn* delegieren möchte, benennt im Antragszertifikat AZ_{dg} neben anderen Parameter T (*Attributes*) und *dn (Target-User)* (siehe das Format der Antragszertifikate weiter oben). Dabei hat *dg* nicht unbedingt Kenntnis darüber, welche seiner insgesamt z Attributszertifikate T beinhalten.

Tritt der Fall auf, dass mehrere, d.h. $1<i\leq z$ seiner Zertifikate genau T enthalten (siehe Teil *a* in Abb. 6-10), so müssen in die Delegierungsrelation entsprechend z neue Tupel $delegation_i\ (dg, dn, zdg_i, zdn)$ eingefügt werden.
Diese Tupel sind bis auf das Attribut $ZDG=zdg_i$ identisch. Die zdg_i Attribute beinhalten jeweils einen Verweis auf eines der z Zertifikate des Delegierungsgebers dg.

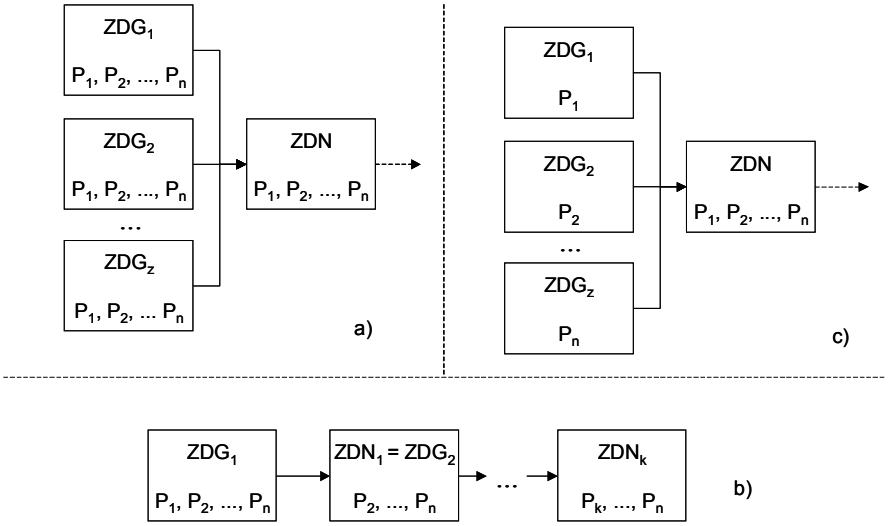


Abb. 6-10: Nicht-triviale Delegierungsvorgänge

- **Delegierung einer Teilmenge der in einem Zertifikat enthaltenen Rechte:** Problematischer ist der Fall, wenn eine Teilmenge $T_{zdg}\subseteq R_{zdg}$ der innerhalb des Zertifikats *zdg* abgelegten Rechte R_{zdg} des Delegierungsgebers *dg* an den Delegierungsnehmer *dn* delegiert werden sollen (siehe Teil *b* in Abb. 6-10). Das hieraus resultierende neue Zertifikat *zdn* an den vorgesehenen Delegierungsempfänger *dn* auszustellen bereitet natürlich kein Problem. Anhand des hierbei in *DRG* eingefügten Tupels *delegation(dg, dn, zdg, zdn)* ist aber nicht eindeutig festzustellen, welche Rechte aus *zdg* delegiert wurden.
  Sollten *dg* die in der obigen Delegierung *nicht* weiterdelegierten Rechte $S_{zdg}=R_{zdg}/T_{zdg}$ später entzogen werden, so müsste ihm ein neues Zertifikat *zdg'* (welches S_{zdg} nicht mehr enthält) ausgestellt und *zdg* entsprechend gesperrt werden. Von dieser Sperrung wäre das vorher für *dn* ausgestellte Attributszertifikat *zdn* nicht betroffen, es bleibt weiterhin gültig.
  Um hier jedoch die Konsistenz der Delegierungsrelation zu bewahren, muss das Tupel *delegation(dg, dn, zdg, zdn)* gelöscht werden. Anschließend muss das neue Tupel *delegation'(dg, dn, zdg', zdn)* eingefügt werden. Es kann natürlich sein, dass von der Sperrung von *zdg* mehrere, d.h. $1<i\leq z$ Attributszertifikate betroffen sind. In diesem Fall müssen z entsprechend neue Tupel in *DGR* eingefügt werden.
  Sollten hingegen den Delegierungsgeber *dg* die früher an den Delegierungsnehmer *dn* delegierten Rechte T_{zdg} entzogen werden, welche ja sowohl in *zdg* als auch in *zdn* enthalten sind, so müssen (z.B. in einem rekursiven Prozess) sämtliche Tupel aus *DGR* gelöscht werden, welche durch das Delegieren von Teilmengen von T_{zdg} entstanden sind. Das Auffinden dieser Tupel in *DGR* wird erschwert, weil die einzelnen Tupel die jeweilig delegierten Rechte (T_{zdg}) nicht enthalten. Eine Abhilfe kann das Limitieren der Kardinalität der in einem Attributszertifikat maximal enthaltenen Rechte R_{zdg} auf genau eins ($|R_{zdg}|=1$) schaffen. Dadurch kann entweder dieses eine Recht ($|T|=1$) oder keins delegiert werden.

Damit können sämtliche von *zdg* ausgehenden Delegierungen wesentlich einfacher aus *DGR* entfernt werden. Bei der Wahl dieser Strategie erhöht sich jedoch die Anzahl der Attributszertifikate in einem System drastisch.

- **Delegierung disjunkter Rechtemengen aus mehreren Zertifikaten:** Sollten Rechte des Delegierungsgebers *dg* an den Delegierungsnehmer *dn* delegiert werden, welche in *z (z>1)* Attributszertifikaten von *dg* verteilt vorliegen (siehe Teil *c* in Abb. 6-10), führt das auch zu nicht-trivialen Schritten. Nach Ausstellung eines neuen Zertifikats *zdn* für *dn* muss *DGR* zunächst um genau *z* neue Tupel *delegation(dg, dn, zdg_i, zdn) ($1<i\leq z$)* erweitert werden.
  Angenommen eines dieser *z* Ursprungs-Zertifikate zdg_{Rev} wird frühzeitig gesperrt. Hierbei kann das jeweilige Delegierungsmodell vorschreiben, dass diese Sperrung mit der Sperrung von *zdn* verbunden ist, da die Grundlage der damaligen Delegierung (zdg_{Rev}) nun nicht mehr besteht.
  Die Sperrung von *zdn* bedeutet zunächst das Entfernen des zuvor eingefügten Tupels *delegation(dg, dn, zdg_{Rev}, zdn)* aus *DGR,* denn es gibt zdg_{Rev} nicht mehr. Außerdem müssen auch die anderen zuvor eingefügten *z-1* Tupel *delegation(dg, dn, zdg_i, zdn)* entfernt werden. Der Grund hierfür ist, dass die in diesen angegebenen zdg_i Attributszertifikate keine „richtigen" Eltern mehr von *zdn* sind. Das Zertifikat *zdn* beinhaltet nämlich mehr Rechte als diese zdg_i Zertifikate zusammen. Ist es erwünscht, dass der Delegierungsempfänger *dn* nach dem Widerruf von zdg_{Rev} die von diesem Widerruf nicht betroffenen Rechte weiterhin behält, so müssen ihm ein neues Zertifikat *zdn'* ausgestellt, und die entsprechenden *z-1* neue Tupel *delegation(dg, dn, zdg_i, zdn')* in *DGR* eingefügt werden.

6.4.2 Betriebliche Auswirkungen der Offline-Delegierung

Im Folgenden werden verschiedene betriebliche Konsequenzen der Offline-Delegierung in einem System diskutiert. Hierbei werden die Vor- und Nachteile der Technik vom Standpunkt der verschiedenen Akteure durchleuchtet.

- **Bedienbarkeit:** Ein Benutzer (z.B. ein „Rechtemakler"), der einen Teil der eigenen Rechte delegieren will, formuliert seine Delegierungswünsche, indem er die jeweiligen Parameter eines Antragszertifikats setzt. Da er den Antrag signiert, entsteht ein eindeutiger Nachweis, dass er dem Delegierungsnehmer Zugang zu den jeweiligen Ressourcen gewähren wollte. Der mit der Antragsstellung verbundene organisatorische und technische Aufwand (Werkzeugunterstützung, sichere Aufbewahrung des eigenen Signierschlüssels, usw.) ist vergleichbar mit dem, der bei der Ausstellung von Attributszertifikaten anfällt.
  Verweigert die jeweilige SOA die Ausstellung des beantragten Zertifikats, so kann der Delegierungsgeber mit Hilfe des Antragszertifikats sowie der daraufhin erhaltenen durch die SOA signierten „Absage-Nachricht" seine Absicht nachweisen. Um hier nachträgliche Fälschungen auszuschließen, kann das Antragszertifikat beispielsweise durch einen Zeitstempeldienst [Sch96] beglaubigt werden.
  Von dem Standpunkt der Delegierungsgeber aus gesehen spielt es eine eher untergeordnete Rolle, wer die Attributszertifikate ausstellt (signiert), die schließlich während der Zugriffskontrolle verwendet werden.
  Der Delegierungsgeber hat nämlich keinerlei Einfluss auf die durch einen autonom agierenden Verifizierer ausgeführten Prozesse, gleich welche Technik zur Abwicklung der Delegierungen verwendet wurde.

Die Delegierungsnehmer erhalten nun Zertifikate, die direkt von den Ressourcenbesitzern ausgestellt werden. Wird ein Zugriffsversuch aufgrund solcher Zertifikate abgelehnt, so kann sich der Zertifikatsbesitzer gleich bei der richtigen Stelle beschweren und evtl. die schnelle Erweiterung seiner Rechte erreichen. Mangels zu speichernder evtl. langer Delegierungspfade vereinfacht sich auch die Aufbewahrung der eigenen Berechtigungen (relevant vor allem in Client-Push-Systemen).

- **Aufwandschätzungen:** Bei der Verwendung der Offline-Delegierung ist insgesamt mit einer Senkung des durchschnittlich zu erwartenden systemweiten Aufwands zu rechnen.
  Die klaren Nutznießer dieser Technik sind die Verifizierer, welche die Zugriffskontrolle realisieren. Sie kommen nach einer einzigen Signaturprüfung gleich zum richtigen Schluss, ob ein vorliegendes Attributszertifikat akzeptiert werden kann.
  Durch die ausfallende Ursprungsprüfung während der Zugriffskontrolle werden die Online-Zertifikatsablagen (Datenbanken) ebenfalls entlastet.
  Die mit der Ursprungsprüfung verbundenen Datenbankanfragen fallen nämlich gänzlich weg, wodurch die durchschnittlichen Antwortzeiten der Datenbanken sinken können. Ebenfalls kann hier insgesamt mit einer kleineren zu speichernden Datenmenge (Anzahl der Zertifikate) gerechnet werden. Die in den Datenbanken abgelegten Zertifikate haben außerdem einen um die Informationen zu diversen Vorgängerzertifikaten geminderten Umfang.
  Ein Zusatzaufwand entsteht jedoch seitens der Ressourcenbesitzer (SOAs). Diese müssen nun sämtliche in die Zugriffskontrolle einfließenden Attributszertifikate ausstellen (signieren) und entsprechend verwalten. Wird hier jedoch der eigens ausgestellte Zertifikatsbestand mittels eines I-CVT verwaltet, entstehen dadurch keine nennenswerten zusätzlichen Berechnungskosten:
  Ein neues aufgrund eines geprüften Antragszertifikats ausgestelltes Zertifikat kann in den jeweiligen Baum hinzugefügt werden, und es wird bei der nächsten Aktualisierung der Wurzelsignatur wirksam. Es ist hier natürlich mit einer erhöhten Größe (Höhe) der einzelnen I-CVTs und folglich mit einer erhöhten erwarteten Länge der Zertifizierungspfade zu rechnen.
  Außerdem müssen SOAs die bei ihnen jeweils eingereichten Antragszertifikate (einmalig) überprüfen. Dies erhöht zunächst die Gesamtkosten seitens der SOAs, verursacht aber keinen erhöhten systemweiten Aufwand, ganz im Gegenteil.
  Ein Antragszertifikat enthält nämlich genau die Berechtigungen, die ansonsten in einem durch den Delegierungsgeber signierten Attributszertifikat veröffentlicht werden müssten. Solche Attributszertifikate müssten während ihrer Gültigkeitsperiode von verschiedenen Verifizierern in der Regel mehr als einmal geprüft werden, um deren vertrauten Ursprung zu finden.
  Auf Seiten der Delegierungsgeber (*Attribute Authority*, AA) entstehen keine erhöhten Berechnungskosten, denn diese müssen nun Antragszertifikate statt Attributszertifikate signieren. Sie bleiben sogar von der Versorgung der Verifizierer mit Prüfdaten zu den nun von den SOAs ausgestellten Zertifikaten verschont.
  Die Benutzer (Delegierungsnehmer) können sich wie schon oben gesagt auf weniger zu speichernde Zertifikate freuen. Dies kann beispielsweise in mobilen Umgebungen, welche auf dem Client-Push-Modell aufbauen, von Vorteil sein, da mobile Endgeräte gewöhnlich über wenig Speicherplatz verfügen.
  Die End-Benutzer eines Server-Pull-Systems können nur indirekt von dieser Technik profitieren, indem ihre Zugriffsversuche schneller bearbeitet werden können.

- **Sicherheitsaspekte**: Die Zugriffskontrolle basiert in einem System, welches den Empfehlungen des X.509-Standards folgt, auf den Entscheidungen der einzelnen Ressourcenbesitzer (SOA). Sie autorisieren potenzielle Zugreifende für die Ressourcennutzung und evtl. für die Weitergabe, d.h. Delegierung der eigenen Rechte. Diese Autorisierungen werden in einem System wirksam, indem die zunächst ausgestellten Attributszertifikate die Verifizierer über verschiedene als unsicher angenommene Vermittlersysteme, d.h. Datenbanken, Netze, usw. erreichen.
  Werden die von einer vertrauten Stelle zertifizierten Rechte ganz ohne Kenntnis dieser Stelle delegiert, entsteht zwangsläufig ein zusätzliches Sicherheitsrisiko. Ein Verifizierer, der ein hierbei ausgestelltes „fremdes“ Attributszertifikat erhält, kann nämlich nicht sicher sein, ob der jeweilige Delegierungsgeber (*Issuer*) tatsächlich im Sinne des vertrauten Ressourcenbesitzers agiert hat.
  Dies gilt auch, wenn die Ursprungsprüfung des Zertifikats rein ablauftechnisch gesehen erfolgreich abläuft. Die jeweiligen Sicherheitsrichtlinien der SOAs können also durch die Delegierungsgeber leicht verletzt werden, was zu einer aus der Sicht der SOAs unsicheren Zugriffskontrolle führen kann. Dieses Problem wird durch die Offline Delegierung gelöst. Hier stammen ja alle während der Zugriffskontrolle ausgewerteten Zertifikate von vertrauten Instanzen, welche bei der Zertifikatsausstellung garantiert in ihrem eigenen Sinne agieren.
  Als ein sicherheitstechnischer Nachteil der Offline-Delegierung kann jedoch die Notwendigkeit zur Bereitstellung einer entsprechenden operativen Schnittstelle zur Empfangnahme der Antragszertifikate seitens der SOAs angesehen werden. Da die Systeme solcher Instanzen möglichst von der Außenwelt abgeschottet, d.h. überwiegend offline arbeiten sollen, verbirgt sich hinter dieser „Öffnung“ ein gewisses Sicherheitsrisiko.
  Aus diesem Grund ist es empfehlenswert, die notwendige Kommunikation mit den Antragstellern eher über so genannte *Out-of-Band*-Mechanismen (z.B. E-Mail, Fax) abzuwickeln. Dies bereitet jedoch Probleme, wenn eine sehr schnelle Umsetzung der Delegierungen erforderlich ist.

7 PAMINA

Im Versuchssystem mit der Bezeichnung *Privilege Administration and Management InfrAstructure* (PAMINA) wurden die in der Arbeit diskutierten Konzepte prototypisch umgesetzt. Die Komponenten des Systems *PAMINA Administration Server*, *Privilege Database* und *Certificate Verifier* (siehe Abb. 7-1) wurden hauptsächlich im Rahmen von Diplom- und Studienarbeiten implementiert [Ebi01] [Oet02] [Boc03] [Küh03] [Vie03].

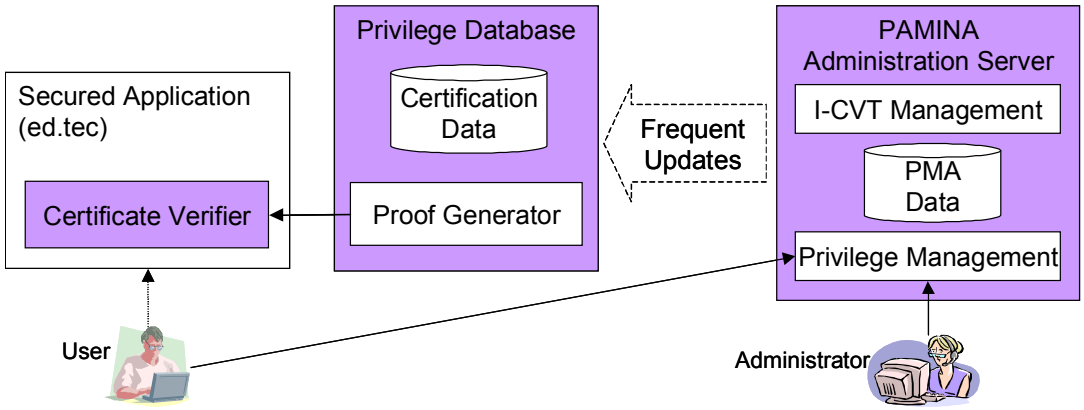


Abb. 7-1: PAMINA im Überblick

Die Einsatzfähigkeit des Systems wurde durch die Umsetzung der Zugriffskontrolle im Wissenstransfersystem *educational technologies* (ed.tec) demonstriert.

7.1 Einsatzszenario: Internet-basierter Wissenstransfer

Die Vorstellung von PAMINA erfolgt anhand des Einsatzszenarios „Internet-basierter Wissenstransfer“. Dort werden verschiedenen Akteuren, d.h. *Autoren*, *Dozenten* und *Lernenden*, innerhalb der (universitären) Aus- und Weiterbildung rechnergestützte Dienste unter Einbezug moderner Internettechnologien angeboten (siehe Abb. 7-2). Um den Anforderungen der genannten Akteure während der Ausübung ihrer Tätigkeiten gerecht zu werden, wurden die jeweiligen Arbeitsabläufe erfasst und analysiert.

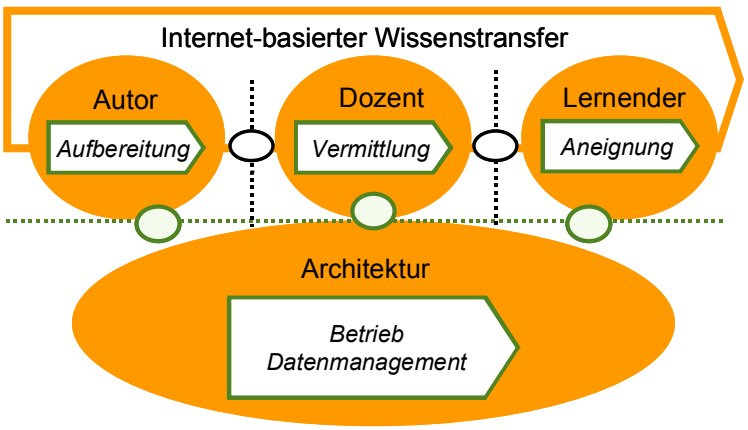


Abb. 7-2: Prozesse im Internet-basierten Wissenstransfer

Erst danach erfolgte eine technische Umsetzung mit Hilfe einer entsprechend entwickelten Lösung [Feu04].

Die im Szenario betrachteten Prozesse können im Hinblick auf die elektronischen Wissensmaterialien wie folgt kategorisiert werden:

- **Aufbereitung:** In diese Kategorie fallen Aktivitäten, welche zur Aufbereitung von Wissensmaterialien im Sinne des jeweiligen Lernziels dienen. Ein Beispiel hierfür ist die Erstellung elektronischer Vorlesungsskripten oder multimedialer Präsentationsmaterialien. Diese Art Aktivitäten werden durch *Autoren*, typischerweise Hochschulprofessoren, durchgeführt.

- **Vermittlung:** Diese Kategorie fast Prozesse zusammen, die während der elektronisch unterstützten, durch *Dozenten* durchgeführten Lernveranstaltungen zum Tragen kommen. Dazu gehört beispielsweise die Live-Übertragung der Stimme eines Dozenten zu einer entfernten Zuhörerschaft einer Vorlesung.

- **Aneignung:** Die vor allem aus pädagogischer Sicht interessanten Prozesse laufen während der Aneignung des elektronisch übermittelten Wissens durch die *Lernenden* ab. Beispiele hierfür sind die Ergänzung der Lernmaterialien durch eigene Notizen oder das wiederholte Abspielen von Bild- und Tonsequenzen, um das Gesagte besser zu verstehen.

7.1.1 ed.tec

Das ed.tec-System dient zur integrierten Unterstützung der skizzierten Prozesslandschaft. Das hinter ed.tec stehende, verteilte und komponentenbasierte Softwaresystem besteht aus den folgenden funktionalen Schichten (siehe Abb. 7-3):

- **Datenhaltung:** Diese Schicht sorgt für die persistente Ablage elektronischer Wissensmaterialien beliebiger Art. Zudem kann hier nach Daten mit gewissen Attributen gesucht werden. Die Funktionalität dieser Schicht kann z.B. durch eine relationale Datenbank oder ein Dateisystem erbracht werden. Durch die heute zur Verfügung stehenden Schnittstellentechnologien, wie z.B. ODBC, kann der Zugriff auf die hier abgelegten Daten über Rechnergrenzen hinweg erfolgen.

- **Datenzugriff:** In dieser Schicht werden die von der Datenhaltungsschicht angebotenen Funktionen auf eine festgelegte Schnittstelle abgebildet. Damit wird das Ziel verfolgt, beliebige Systeme mit unterschiedlichsten Schnittstellen in der Datenhaltungsschicht einsetzen zu können. Die Funktionen der Datenzugriffsschicht werden im *Content Information Server* (CIS) gekapselt angeboten. Der CIS wurde auf Basis der Webservice-Technologie [BHM+04] umgesetzt.

- **Geschäftslogik:** Die Komponenten dieser Schicht erbringen die Kernfunktionalität von ed.tec, d.h. die serverseitige Unterstützung der eigentlichen Geschäftsprozesse. Dabei werden unter anderem die Pflege, Suche und Zusammenstellung der angebotenen Wissensmaterialien unterstützt.

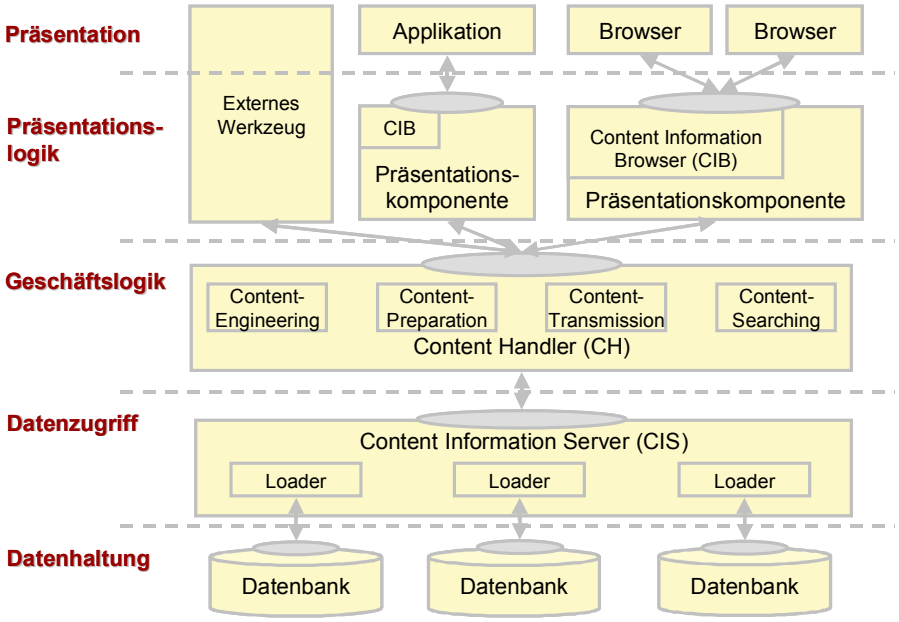


Abb. 7-3: Schichtenmodell von ed.tec

- **Präsentationslogik:** Hier werden Funktionen für die grafische Aufbereitung der für Menschen sichtbaren Daten sowie Steuerungsfunktionen erbracht. Eine wichtige Komponente dieser Schicht ist der *Content Information Browser* (CIB).

- **Präsentation:** Diese Schicht verarbeitet die von der Präsentationslogikschicht generierten Informationen und stellt diese grafisch dar. Dies erfolgt in der Regel mithilfe eines Internet-Browsers.

Die Funktionsweise der Architektur wird anhand eines Beispiels vorgestellt: Der Studierende *Bob* möchte Wissensmaterialien zum Kurs „*Internet Systeme und Web-Applikationen*“ (kurz *ISWA*) erhalten. Insbesondere interessiert ihn das im ed.tec abgelegte Skriptum zum Kapitel „*Architekturgrundlagen*“.
Er startet also seinen Internet-Browser und gibt den URL zum speziell für Studierende vorbereiteten Einstieg ins ed.tec ein. Die im System zum gewählten Thema vorhandenen Materialien kann er über die grafische Navigationskomponente CIB finden. Mit Hilfe des CIB können die strukturiert abgelegten Wissensmaterialien über Abstiege in einem Verzeichnisbaum erreicht werden. *Bob* findet den Weg *ISWA→Architekturgrundlagen* zum gewünschten Thema leicht und bekommt Informationen zu den zur Verfügung stehenden Wissensmaterialien. Er entscheidet sich für das Skript mit dem Titel „*Architekturgrundlagen*“, möchte dieses Online lesen und wählt hierfür den entsprechend gekennzeichneten Link aus.
Auf sein Kommando hin aktiviert der CIB die entsprechende Funktionalität der Geschäftlogikschicht. Diese besteht in diesem Fall darin, *Bob*s Wunsch in eine Form zu bringen, die der für die Datenzugriffe zuständige CIS „versteht“.
Der CIS selbst verfügt über als *Loader* bezeichnete, dynamisch geladene Module (siehe Abb. 7-3), die jeweils einer der unterstützten Datenhaltungsmethoden angepasst sind. Da in unserem konkreten Fall eine Datei aus einem Verzeichnisdienst angefordert werden muss, wird der hierfür zuständige Loader gestartet. Um die gewünschte Datei in der Ablage tatsächlich aufzufinden, muss sie richtig adressiert werden.

Dafür wird eine Kennung (ID) vom Typ *LOADERID/DATABASEID/CONTENTID* benötigt. Die anvisierte Datei kann in der ed.tec-Ablage beispielsweise unter dem ID

LOADERID="EDTEC001"
DATABASEID="ISWA"
CONTENTID="C:\AREAS\ISWA\COURSEWARE\01.VORLESUNG
01-01.ARCHITEKTUR-GRUNDLAGEN\L.ARCHITEKTUR-GRUNDLAGEN.PDF"

repräsentiert sein. Kann der aktivierte Loader diese Datei im Verzeichnis finden, so wird diese bis zur Präsentationsschicht „durchgereicht" und erscheint schließlich im inzwischen automatisch gestarteten Viewer-Modul in *Bobs* Internet-Browser.

7.1.2 Zugriffskontrolle für ed.tec

Angesichts der möglichen Sensibilität der durch ed.tec verwalteten Informationen bedarf es der Etablierung einer Zugriffskontrolle im System. Naheliegende Beispiele für sensible Daten sind Prüfungsfragen, Musterlösungen, Prüfungsergebnisse aber auch hochwertige Wissensmaterialien, die alle nur bestimmten (registrierten) Benutzern zugänglich gemacht werden sollten. Beim Entwurf von ed.tec wurden die hiermit zusammenhängenden Probleme vorerst nicht betrachtet. Im Vordergrund stand vielmehr die Erfüllung funktionaler Anforderungen.

Eine sicherheitstechnische Erweiterung von ed.tec wurde in der Diplomarbeit [Boc03] vorgenommen. Dabei wurden vor allem Maßnahmen für die Authentifizierung der Benutzer, die Verschlüsselung der übertragenen Daten, sowie für die Durchsetzung von Zugriffsentscheidungen getroffen und prototypisch umgesetzt.
Die entstandene Lösung beinhaltet eine passwortbasierte Authentifizierung registrierter Benutzer. Die hierfür notwendige Anmeldung (Abmeldung) der Benutzer erfolgt über in die grafische Oberfläche von ed.tec integrierte Angabemasken, und wird durch eine Benutzerdatenbank unterstützt. Um die übertragenen und serverseitig geprüften Passwörter sowie die in den späteren Arbeitsschritten transferierten Daten (Lernmaterialien, Benutzereingaben, etc.) gegen unbefugte Zugriffe zu schützen, wurde HTTPS [Res00] [FKK96] eingesetzt. Ebenfalls wurde in Form eines integrierten Sitzungsmanagements dafür gesorgt, dass die Identität eines zugreifenden Benutzers stets auf seine initiale Authentifizierung zurückzuführen ist. Weitere konzeptionelle und technische Details zu diesen Themen sind in [Boc03] nachzulesen.
Im Folgenden werden Aspekte der Autorisierung sowie das Treffen und Durchsetzen von Zugriffsentscheidungen im ed.tec diskutiert.

7.1.2.1 Das ed.tec-Autorisierungsmodell

Zunächst wird das Modell zur Verwaltung und Auswertung der im ed.tec verwendeten Autorisierungsdaten charakterisiert. Die dem Modell folgenden Autorisierungsdaten in Form von X.509-Attributszertifikaten stellt PAMINA bereit.

- **Eigentümerprinzip:** Bei der Vergabe von Zugriffsrechten im ed.tec gilt grundsätzlich das Eigentümerprinzip (siehe Kap. 2.1.3): Zugang zu einer bestimmten Menge von Wissensmaterialien kann primär deren Eigentümer gewähren. Als Eigentümer gelten normalerweise die Autoren der Materialien sowie die Administratoren des Systems.

- **Erlaubte Zugriffswege mit rekursiver Geltung:** Autorisierungsdaten für das ed.tec-System beinhalten ausschließlich erlaubte Zugriffswege, d.h. (*Subjekt, Objekt, Zugriffstyp*)-Tripel (siehe auch Kap. 2.1). Dies impliziert, dass alle anderen nicht erfassten Zugriffswege per Definition verboten sind.
  Um die Zugriffswege im ed.tec festzulegen, d.h. diese zu administrieren, muss die Menge der zugreifenden Subjekte (Benutzer), Objekte (Wissensmaterialien), sowie der möglichen Zugriffstypen identifiziert werden. Bei der Analyse der Vorgänge innerhalb von ed.tec wurden die atomaren Zugriffstypen *Erzeugen*, *Lesen*, *Schreiben* und *Löschen* identifiziert. Diese können sich auf die aus Sicht der Benutzer als *Kurs*, *Kapitel* oder *Datei* klassifizierten hierarchisch angeordneten Wissensmaterialien beziehen.
  Im oben behandelten Beispielfall muss entschieden werden, ob *Bob* den Zugriff *Lesen* am Objekt *Kurs=ISWA, Kapitel=Architekturgrundlagen, Datei=Architekturgrundlagen.pdf* ausführen darf.
  Da die Wissensmaterialien im ed.tec in einer hierarchischen Anordnung repräsentiert und verwaltet werden, gelten die mit einer Hierarchieebene assoziierten Autorisierungsdaten ebenfalls für die untergeordneten Ebenen (Rekursivität). Das heißt, die für einen Benutzer auf einer übergeordneten Ebene ausführbaren Aktionen (Erzeugen, Lesen, Schreiben, Löschen) sind auch auf den untergeordneten Ebenen ausführbar.
  Kann beispielsweise *Bob* auf den Kurs *ISWA* lesend (schreibend, usw.) zugreifen, so kann er sämtliche Kapitel und Dateien innerhalb dieses Kurses lesen (schreiben, usw.). Diese Rekursivität kann allerdings auf den untergeordneten Ebenen durch entsprechende Angaben aufgehoben werden. Durch die Anwendung rekursiv geltender Autorisierungsdaten wird der Umfang der zu administrierenden und zu speichernden Autorisierungsdaten reduziert.

- **Verwendung von Fähigkeitslisten:** Die für einen Benutzer erlaubten Zugriffswege werden in Form von Fähigkeitslisten (siehe Kap. 2.1.3) verwaltet und ausgewertet. Die Fähigkeitslisten eines Benutzers werden in für diesen Benutzer (*Holder*) ausgestellten Attributszertifikaten festgehalten.

- **Verwendung von Rollen:** Es ist grundsätzlich möglich, Rollen (siehe Kap. 2.1.3) mit Hilfe von Rollenzertifikaten (siehe Kap. 2.3.1) zu definieren und diesen jeweils mehrere Benutzer zuzuordnen. Die aktuellen Rollenzugehörigkeiten eines Benutzers lassen sich anhand von für diesen Benutzer ausgestellten Rollenzuordnungszertifikaten (siehe Kap. 2.3.1) feststellen.

- **Delegierung:** Es besteht die Möglichkeit zur Dezentralisierung der Autorisierung. Dies erfolgt durch Delegierungen, welche initial die Eigentümer, d.h. typischerweise die Autoren, der Wissensmaterialien vornehmen.

- **Server-Pull-Modell:** Die ed.tec-Autorisierungsdaten werden serverseitig, in der PAMINA-Komponente *Privilege Database* gespeichert. Während der Zugriffskontrolle wird dementsprechend auf diesen Datenbestand zurückgegriffen. Das bedeutet, dass autorisierte ed.tec-Benutzer ihre Autorisierungsdaten nicht selber verwalten und während der Zugriffskontrolle vorzeigen müssen.

7.1.2.2 Einordnung und Ablauf der Zugriffskontrolle im ed.tec

Zugriffsentscheidungen im ed.tec werden in der Datenzugriffsschicht von der Verifizierungskomponente *Privilege Evaluator* getroffen, die eine Erweiterung des *Content Information Server* (CIS) ist. Dabei wird festgestellt, ob die Rechte (präziser: die Fähigkeiten) eines Zugreifenden ausreichen, um die jeweilig gewünschte Aktion im System durchzuführen. Unterstützt wird der *Privilege Evaluator* durch die Komponenten *Certificate Verifier* und *Privilege Cache* (siehe Abb. 7-4).

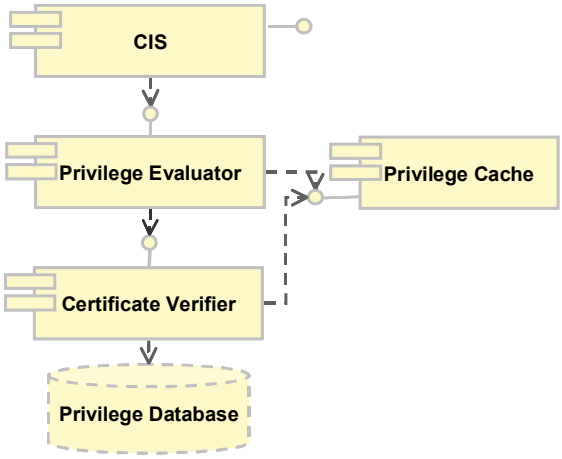


Abb. 7-4: Komponenten zur Realisierung der ed.tec Zugriffskontrolle

Im *Privilege Cache* werden die bereits mit Hilfe des *Certificate Verifier* geprüften, d.h. gültigen, Benutzerrechte zwischengespeichert, um die Zugriffskontrolle zu beschleunigen. Dieser Cache enthält für jeden angemeldeten ed.tec-Benutzer einen eigenen Bereich (Benutzerprofil). Die in diesem liegenden Daten werden periodisch aktualisiert, so lange der Benutzer bzw. seine Sitzung aktiv ist. Ein anfangs leeres Benutzerprofil wird durch den *Privilege Evaluator* erzeugt, nachdem sich der jeweilige Benutzer erfolgreich angemeldet hat.
Die aktuellen Zugriffsrechte des Benutzers werden mit Hilfe von PAMINA ermittelt. Dabei werden die in der jeweiligen Situation relevanten Attributszertifikate ermittelt und anschließend geprüft. Involviert in diesen Prozess sind die *Privilege Database* und der *Certificate Verifier*. Die Funktionsweise und das Zusammenspiel dieser beiden Komponenten werden in Kap. 7.3 respektive in Kap. 7.4 ausführlicher diskutiert.

Der Ablauf der durch den *Privilege Evaluator* durchgeführten Zugriffskontrolle wird anhand des oben beschriebenen Zugriffsfalls geschildert: Wählt *Bob* die ed.tec-Einstiegseite, so wird er zur Angabe seines Benutzernamen und Passworts aufgefordert. Diese werden über die inzwischen aufgebaute verschlüsselte Verbindung an den ed.tec-Webserver übertragen. *Bobs* Angaben werden mit Hilfe einer Benutzerdatenbank geprüft, der genauere Ablauf der Identifizierung und Authentifizierung ist in [Boc03] nachzulesen. War die Anmeldung erfolgreich, so wird für *Bob* eine Sitzung geöffnet.
Anschließend wird durch den *Privilege Evaluator* ein leeres Benutzerprofil im *Privilege Cache* erzeugt. Nach dieser Phase ist *Bob* also noch nicht in der Lage, auf die geschützten ed.tec-Ressourcen zuzugreifen. Hierfür muss noch sein Benutzerprofil mit Autorisierungsdaten gefüllt werden. Dabei fordert der *Privilege Evaluator* den *Certificate Verifier* auf, sämtliche für *Bob* ausgestellten Fähigkeiten auszuhändigen. Die von der *Privilege Database* (siehe Kap. 7.3) erhaltenen Daten werden nach Überprüfung durch den *Certificate Verifier* (siehe Kap. 7.4) im Cache abgelegt.

Sie bilden die Grundlage der weiteren auf *Bob* bezogenen Zugriffsentscheidungen innerhalb dieser Sitzung.
Möchte nun *Bob* wie im obigen Beispiel auf das ed.tec-Objekt (Datei) *ISWA/Architekturgrundlagen/Architekturgrundlagen* lesend zugreifen, so wird in seinem Benutzerprofil nach den (rekursiv geltenden) Fähigkeiten *{Objekt=ISWA, Zugriffstyp=Lesen}*, *{Objekt=ISWA/Architekturgrundlagen, Zugriffstyp=Lesen}* und schließlich nach *{Objekt=ISWA/Architekturgrundlagen/Architekturgrundlagen, Zugriffstyp=Lesen}* gesucht.
Ist eine dieser Fähigkeiten vorhanden, so wird der Zugriff erlaubt und durch den CIS bzw. durch das aktivierte Loader-Modul ausgeführt. Ansonsten erhält *Bob* eine Fehlermeldung.
Die Vergabe und Pflege der auf diese Art und Weise ausgewerteten Autorisierungsdaten für ed.tec erfolgt mit Hilfe des nachfolgend vorgestellten *PAMINA Administration Servers*.

7.2 PAMINA Administration Server

Der *PAMINA Administration Server* (PAS) ist ein System zur Unterstützung der Arbeitsabläufe innerhalb einer Autorisierungs- bzw. Zertifizierungsstelle, die im X.509-Kontext als *Source of Authority* bzw. *Attribute Authority* bezeichnet wird (siehe Kap. 2.2.4). Diese Art von Instanzen werden in PAMINA zusammengefasst als *Privilege Management Authorities* (PMA) bezeichnet. Im Fall von ed.tec können z.B. einzelne Institute oder Fakultäten einer Universität als PMA auftreten.

Der PAS bietet seinen Benutzern, das heißt vor allem Administratoren eines durch PAMINA gesicherten Systems, Dienste zur Vergabe und durchgehenden Pflege von Autorisierungsdaten an. Er wird mittels grafischer Werkzeuge bedient, die in gängigen Internet-Browsern laufen können. In Abhängigkeit der jeweiligen administrativen Vorgänge erfolgt „im Hintergrund“, also den PAS-Benutzern unsichtbar, eine Abbildung der jeweilig manipulierten Autorisierungsdaten auf Attributszertifikate, die durch die I-CVT-Technik geschützt werden.

Das Anwendungsfalldiagramm in Abb. 7-5 fasst die den Anwendern sichtbaren Funktionen des PAS zusammen. Wie man sieht, wird auf der Anwenderseite grundsätzlich zwischen „Endbenutzern“ sowie „Administratoren“ unterschieden:

- **Endbenutzer** sind Anwender, die primär auf ein durch PAMINA abgesichertes System, wie z.B. ed.tec, zugreifen sollen. Sie können ihre aktuellen (serverseitig gespeicherten) Berechtigungen in einer benutzerfreundlichen Form einsehen (siehe den Anwendungsfall „*Einsehen von Zugriffsrechten*“). Somit können sie stets über ihre aktuellen Zugriffsmöglichkeiten im jeweiligen System informiert werden. Außerdem, falls es im jeweiligen Anwendungskontext vorgesehen ist, können Endbenutzer ihre eigenen Berechtigungen delegieren („*Delegierung von Zugriffsrechten*“).

- **Administratoren** sind die Haupt-Zielgruppe des PAS. Sie sind primär für die Durchführung der Autorisierung der Endbenutzer eines abgesicherten Systems zuständig und bilden damit die „technische Belegschaft“ einer PMA.

Sie können für die Endbenutzer Fähigkeiten anlegen und diese pflegen („*Verwaltung von Fähigkeiten*“). Ebenfalls werden sie bei der Erzeugung und Pflege von Rollen („*Verwaltung von Rollen*“) sowie der Zuordnung der Benutzer zu existierenden Rollen („*Rollenzuordnung von Benutzern*“) durch den PAS unterstützt.

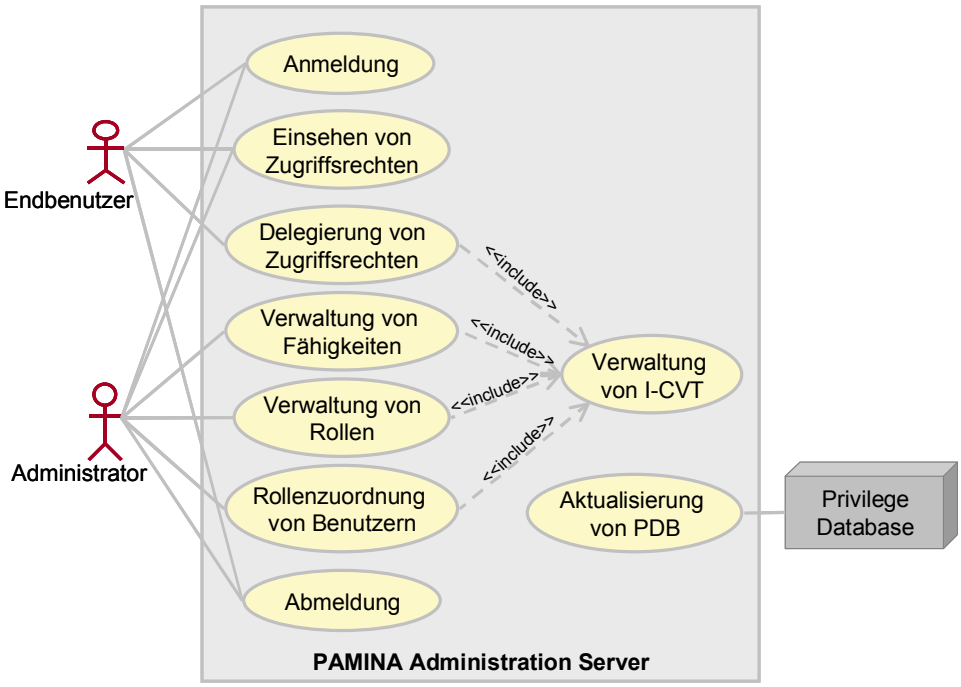


Abb. 7-5: Anwendungsfalldiagramm PAMINA Administration Server

Für beide Arten von Anwendern gilt, dass sie sich beim PAS anmelden („*Anmeldung*“) müssen, um die jeweiligen Funktionen abzurufen. Ein bereits angemeldeter Anwender kann sich nach Beendigung der Arbeit abmelden („*Abmeldung*“).

Die Ergebnisse der verschiedenen administrativen Tätigkeiten werden den abgesicherten Systemen über die in Kap. 7.3 vorgestellte PAMINA-Komponente *Privilege Database* (PDB) erreichbar gemacht. Der PDB-Datenbestand wird durch den PAS in einem automatisierten Prozess, d.h. ohne Benutzerinteraktionen, periodisch aktualisiert („*Aktualisierung von PDB*“). Hierbei erhält die PDB seit der letzten Aktualisierung neu hinzugekommenen Zertifikate sowie Instruktionen und Daten zur Aktualisierung des zugehörigen I-CVT.

Da der PAS die Brücke zwischen den menschlichen Benutzern und dem Rest des weitgehend automatisiert laufenden PAMINA-Systems schlägt, müssen die hier über die grafische Benutzeroberfläche angebotenen Funktionen dem jeweiligen Anwendungskontext und dem Autorisierungsmodell angepasst werden. Die derzeitige PAS-Implementierung wurde den Bedürfnissen des ed.tec-Systems entsprechend entwickelt.

7.2.1 Bedienung und Zertifizierungsvorgänge im Überblick

Bei der Entwicklung vom PAS wurde auf die intuitive Bedienung des Systems Wert gelegt. Dabei gilt, dass sich die Anwender den Aufgaben im Zusammenhang mit der Autorisierung widmen können, ohne mit den dabei tatsächlich entstehenden oder eben aus dem System entfernten Attributszertifikaten konfrontiert zu werden.

Die über die grafische Benutzeroberfläche erreichbaren Funktionen wurden in einem einfachen Menü gruppiert, den in Abb. 7-5 dargestellten Anwendungsfällen entsprechend. Die wichtigsten Funktionen werden hier kurz vorgestellt. Eine detailliertere Beschreibung der Funktionen, die passenden UML-konformen Aktivitätsdiagramme sowie weitere Bildschirmabzüge sind in [Küh03] und [Boc03] zu finden.

- **Einsehen von Zugriffsrechten:** Diese Funktionalität wird sowohl angemeldeten Endbenutzern als auch Administratoren angeboten. Um hier eine bestmögliche thematische und technische Integration in das ed.tec-Szenario zu erreichen, wurde bei der Visualisierung der Zugriffsrechte auf die oben erwähnte ed.tec-Komponente *Content Information Browser* (CIB) zurückgegriffen: Der CIB wurde dabei um eine Ansicht der aktuellen Rechte des jeweilig gefragten Benutzers zum jeweilig angezeigten ed.tec-Objekt erweitert. Die hier angezeigten Autorisierungsdaten werden aus dem (lokal) vorhandenen Zertifikatsbestand der PMA ermittelt (siehe weiter unten).
  Ein Beispiel für das Erscheinen der eigenen Berechtigungen *Lesen* (*Read, R*) und *Schreiben* (*Write, W*) eines am PAS angemeldeten ed.tec-Benutzers innerhalb der Ablage mit dem Namen „*courseware*“ zeigt Abb. 7-6.

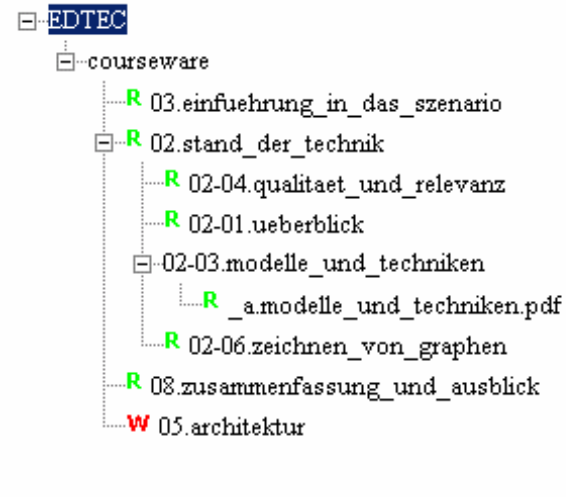


Abb. 7-6: Im erweiterten CIB angezeigte Zugriffsrechte eines ed.tec-Benutzers

Aufgrund der auf dieser Weise visualisierten Informationen können die weiteren administrativen Schritte getätigt werden, welche schließlich zum Ausstellen oder eben Entfernen von Zertifikaten führen:

- **Verwaltung von Fähigkeiten:** Primär administrative Aufgaben sind das Anlegen und Pflegen von Fähigkeiten („Rechten“) für die (ed.tec-)Endbenutzer.
  Beim Anlegen einer neuen Fähigkeit wählt der Administrator mit Hilfe des oben gezeigten erweiterten CIB die jeweilige ed.tec-Ressource aus und verknüpft diesen mit einem aus einer Benutzerliste ausgewählten Benutzernamen, wie z.B. *Bob*, sowie dem gewünschten Zugriffstypen, wie z.B. *Lesen*. Soll die angelegte Fähigkeit zeitlich befristet sein, so kann deren Ablaufdatum durch eine eingebaute Kalenderfunktion bestimmt werden. Ebenfalls kann bestimmt werden, ob der Benutzer die neue Fähigkeit delegieren darf.
  Nach Anlegen einer Fähigkeit wird ein neues Attributszertifikat für den jeweiligen Benutzer ausgestellt und im Datenbestand der PMA abgelegt. Ein Beispielzertifikat (*ACInfo*) zeigt Abb. 7-7.

Aus diesem Zertifikat wird für die ed.tec-Zugriffskontrolle ersichtlich, dass *Bob* (*Holder*) im angegebenen Zeitraum (*AttrCertValidityPeriod*) sämtliche Wissensmaterialien unterhalb der Hierarchieebene */ISWA/Architekturgrundlagen/* lesen und dass er dieses Recht grundsätzlich delegieren darf *(Authority:TRUE)*.

```
ACInfo:
        Holder:
                EntityName: Bob
        Issuer:
                IssuerName: PMA-Telematik
        SerialNumber:
                0x42
        AttrCertValidityPeriod:
                NotBeforeTime: Dec 24 09:27:16 CET 2003
                NotAfterTime: Dec 24 19:27:16 CET 2005
        Attributes: 1
                Object: /ISWA/Architekturgrundlagen/
                Access: Read
        Extensions:
                BasicAttConstraints:
                        Authority: TRUE
```

Abb. 7-7: Attributszertifikat mit Benutzerrechten für Bob

Vergleicht man dieses Zertifikat mit dem X.509-Format (siehe Abb. 2-5), so stellt man das Fehlen bestimmter Felder, wie z.B. *Version* und *Signature* fest. Diese Felder, die ja im gesamten Zertifikatsbestand einer Zertifizierungsstelle identisch ausfallen, sind Bestandteil des die Zertifikate signierenden I-CVT. Dies folgt dem in Kap. 4.2 diskutierten Konzept zur Redundanzsenkung der zertifizierten Daten einer Zertifizierungsstelle (siehe dazu auch Abb. 4-3).

Eine vorhandene Fähigkeit kann geändert oder gelöscht werden. Jegliche Änderungen an den Angaben zu einer Fähigkeit, wie z.B. das Neusetzen des Ablaufdatums, gehen mit dem Entfernen des bereits vorhandenen sowie dem Ausstellen eines neuen Zertifikats einher. Sollte ein Benutzer eine bestimmte Fähigkeit nicht mehr besitzen, so wird das jeweilige Zertifikat aus dem Bestand ohne Nachfolger entfernt.

- **Verwaltung von Rollen:** Die Werkzeugunterstützung für die rollenbasierte Zugriffskontrolle beinhaltet Funktionen zum Anlegen, Ändern und Entfernen von Rollen im ed.tec-System.
  Beim Anlegen einer neuen Rolle müssen deren Name und die mit der Rolle verknüpften Fähigkeiten, also Objekt-Zugriffstyp-Paare, spezifiziert werden. Bei Auswahl bzw. Erzeugung der gewünschten Fähigkeiten für eine Rolle wird ebenfalls auf die oben erwähnte erweiterte CIB-Komponente zurückgegriffen.
  Eine neu angelegte Rolle erscheint in Form eines Rollenzertifikats im Datenbestand der PMA. In Abb. 7-8 ist ein Rollenzertifikat zur Spezifizierung der Rolle „*Informatiker*“ gezeigt.

```
ACInfo:
        Holder:
                RoleName: Informatiker
        Issuer:
                IssuerName: PMA-Telematik
        SerialNumber:
                0x27
        AttrCertValidityPeriod:
                NotBeforeTime: Dec 24 09:27:16 CET 2003
               NotAfterTime: Dec 24 19:27:16 CET 2010
        Attributes: 1
              Object: /ISWA/
              Access: Write
```

Abb. 7-8: Beispiel für ein ed.tec-Rollenzertifikat

Die mit dieser Rolle über Rollenzuordnungszertifikate (siehe weiter unten) verknüpften Benutzer dürfen im ed.tec sämtliche Objekte unterhalb der Hierarchieebene *ISWA* schreiben (siehe das Feld *Attributes*).
Bei Änderungen an einer Rolle wird das betroffene Rollenzertifikat aus dem Datenbestand der PMA entfernt, und ein neues Rollenzertifikat mit den geänderten Daten wird angelegt. Sollte eine Rolle ohne Nachfolger gelöscht werden, so wird das diese repräsentierende Zertifikat aus dem Datenbestand entfernt.

- **Rollenzuordnung von Benutzern:** Rollen müssen natürlich Benutzer zugeordnet werden. Dieser Vorgang wird im PAS durch die bereits erwähnten einfachen Auswahllisten unterstützt. Dabei werden die Benutzer sowie die existierenden Rollen in jeweils einer Liste dargestellt (siehe den Bildschirmabzug in Abb. 7-9).

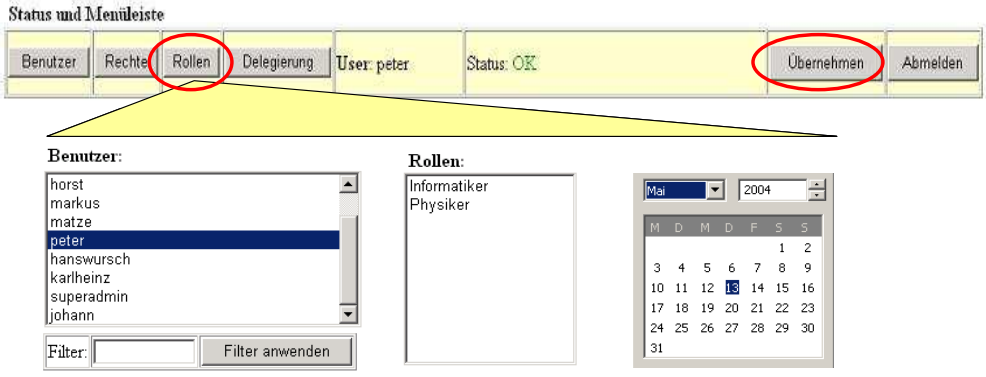


Abb. 7-9: Oberflächenelemente für die Rollenzuordnung von Benutzern

Auch hier kann die gewünschte Dauer der Rollenzugehörigkeit festgelegt werden. Nach Festlegen einer Rollenmitgliedschaft wird dem jeweiligen Benutzer ein neues Attributszertifikat, genauer ein Rollenzuordnungszertifikat, ausgestellt. Dieses wird aus dem Datenbestand wieder entfernt, falls der Benutzer die Rollenmitgliedschaft verliert.
In Abb. 7-10 ist ein Rollenzuordnungszertifikat dargestellt, welches für den Benutzer (*Holder*) *Bob* ausgestellt wurde.

```
ACInfo:
        Holder:
                EntityName: Bob
        Issuer:
                IssuerName: PMA-Telematik
        SerialNumber:
                0x47
        AttrCertValidityPeriod:
                NotBeforeTime: Dec 20 09:27:16 CET 2003
               NotAfterTime: Dec 24 19:27:16 CET 2005
        Attributes: 0
        Extensions:
              RoleSpecCertIdentifier:
                              RoleName: Informatiker
                              RoleCertIssuer: PMA-Telematik
                             RoleCertSerialNumber: 0x27
```

Abb. 7-10: Rollenzuordnung von Bob zur Rolle Informatiker

Das Zertifikat bescheinigt ihm die Rollenzugehörigkeit „*Informatiker*“ im von der *PMA-Telematik* (*Issuer*) vorgesehenen Zeitraum (*AttrCertValidityPeriod*).
Die unter dem Feld *Extensions* untergebrachten Daten *RoleName*, *RoleCertIssuer* und *RoleCertSerialnumber* identifizieren das entsprechende Rollenzertifikat (siehe Abb. 7-8), aus welchem die hiermit verknüpften Fähigkeiten entnommen werden können.

- **Delegierung von Zugriffsrechten:** Die durch den PAS unterstützte Rechtedelegierung läuft gemäß der Technik der Offline-Delegierung ab, die in Kap. 6.4 ausführlich diskutiert wurde. Ein als Delegierungsgeber auftretender Benutzer muss dabei einen Delegierungsantrag mit Hilfe der grafischen Oberfläche des PAS „zusammenklicken“ und diesen nach Bestätigung geltend machen:
  Die zu delegierenden Rechte gibt er durch Auswählen des entsprechenden CIB-Knotens an, den Delegierungsnehmer wählt er aus der Liste der bekannten Benutzer. Sollte die Delegierung zeitlich befristet sein, so kann das Ablaufdatum durch die bereits erwähnte Kalenderfunktionalität bestimmt werden.
  Nach Versenden des Delegierungsantrags wird zunächst geprüft, ob der Delegierungsgeber berechtigt ist, die gewählten Rechte zu delegieren. Dabei wird das Ergänzungsfeld *Authority* des jeweiligen Ursprungszertifikats überprüft. Hat dieses den Wert *TRUE* (wie z.B. das Zertifikat in Abb. 7-7), so werden für den Delegierungsnehmer die Zertifikate ausgestellt. Die während einer Delegierung festgelegten Parameter können im Laufe der Zeit geändert werden. Hierfür werden dem Benutzer sämtliche durch ihn vorher getätigten Delegierungen in einer Listenform angezeigt. Nach der Durchführung der gewünschten Änderungen wird das zugehörige Zertifikat, welches die Delegierung festhielt, aus dem Datenbestand der PMA entfernt. Anschließend wird ein neues Zertifikat mit den geänderten Angaben erzeugt.
  Es ist ebenfalls möglich, eine Delegierung (frühzeitig) zu stornieren. Nach Auswahl der zu widerrufenden Einträge entfernt der PAS die hiervon betroffenen Zertifikate aus dem PMA-Datenbestand.

Die geschilderten Autorisierungsvorgänge sowie deren Abbildung auf die verschiedenen Zertifikatsarten fasst Abb. 7-11 zusammen.

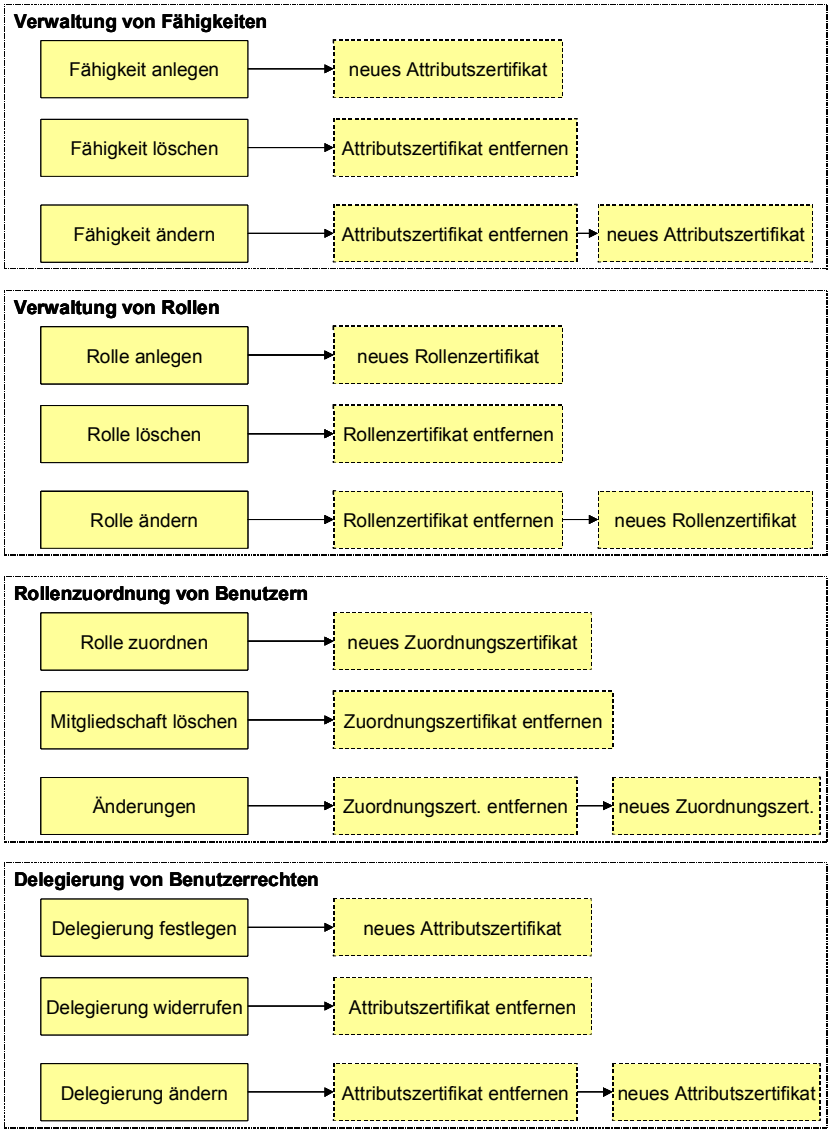


Abb. 7-11: Autorisierung und Abbildung auf den Zertifikatsbestand einer PMA

- **Verwaltung von I-CVT:** Die im Rahmen der Autorisierungsvorgänge entstandenen Zertifikate werden mit Hilfe der I-CVT-Technik gesichert: Jede Instanz von *PAMINA Administration Server* (PAS) pflegt einen I-CVT, welcher regelmäßig aktualisiert und im Namen der PMA signiert wird.
  Der PAMINA-Prototyp verwendet hierfür die in Kap. 5.3 behandelte B^+-Baum-basierte Konstruktion. Der eigene I-CVT einer PMA erfasst, d.h. indiziert und signiert, sämtliche durch diese PMA erzeugten Attributs-, Rollen- und Rollenzuordnungszertifikate.
  Beim Aufsetzen einer neuen PAS-Instanz wird zunächst ein leerer Baum angelegt, dessen Wurzel gleich signiert wird. Der neue Baum wird persistent im Datenbestand der PMA gesichert. Danach werden dem Baum, den am (anfangs ebenfalls leeren) Zertifikatsbestand getätigten Änderungen entsprechend, Angaben zu neuen Zertifikaten hinzugefügt oder zu bestehenden Zertifikaten gelöscht.

Dabei wird die Struktur des I-CVT entsprechend modifiziert: Zunächst erfolgt das Einfügen (Löschen) des jeweiligen Suchschlüssels und Hashwerts des betroffenen Zertifikats in den I-CVT.
Als Suchschlüssel dient im Fall von ed.tec die Zertifikatsfeld-Sequenz *Holder||SerialNumber*, was ihrerseits die in Kap. 5.3 behandelten Beispiele motiviert hat. Nach jeder Einfüge- oder Löschoperation erfolgt auch eine Balancierung des Baums. Nach diesem Schritt werden die in den betroffenen Baumknoten abgelegten Hashwerte neu berechnet. Dies geschieht gemäß der in Def. 5-6 angegebenen Berechnungsformel.
In regelmäßigen Abständen wird die neue Wurzelsignatur unter Einbezug des privaten Schlüssels der PMA erzeugt. In die Berechnung dieser Signatur fließen der zum Zeitpunkt des Signierens vorliegende aktuelle Wurzelhashwert und die als *CVT-Infos* bezeichneten Daten (siehe Beispiel in Abb. 7-12) ein.

```
CVT-Infos:
        Version: 2
        Issuer:
            IssuerName: PMA-Telematik
        Signature: SHA1_RSA
        AttrCertValidityPeriod:
            NotBeforeTime: Dec 24 09:27:16 CET 2003
            NotAfterTime: Dec 24 19:27:16 CET 2003
```

Abb. 7-12: Beispiel für durch die Wurzelsignatur erfasste CVT-Infos

Diese *CVT-Infos* enthält im Wesentlichen die Zertifikatsfelder, die für sämtliche Zertifikate einer PMA identisch ausfallen und deshalb nur einmalig gespeichert werden müssen (siehe das Konzept zur Redundanzsenkung der zertifizierten Daten in Kap. 4.2). Insbesondere wird bei jeder Aktualisierung der Wurzelsignatur die in *CVT-Infos* gespeicherte Gültigkeitsperiode (*AttrCertValidityPeriod*) des gesamten Baums auf einen neuen Wert gesetzt.
Nach erfolgreicher Durchführung dieser Vorgänge wird der neue I-CVT im Datenbestand der PMA abgelegt. Anschließend wird mit der Aktualisierung der PDB (siehe Kap. 7.3) begonnen.

7.2.2 Interner Aufbau

Die oben geschilderten Vorgänge werden durch die in Abb. 7-13 gezeigten, in der Programmiersprache *C#* implementierten PAS-Komponenten erbracht.

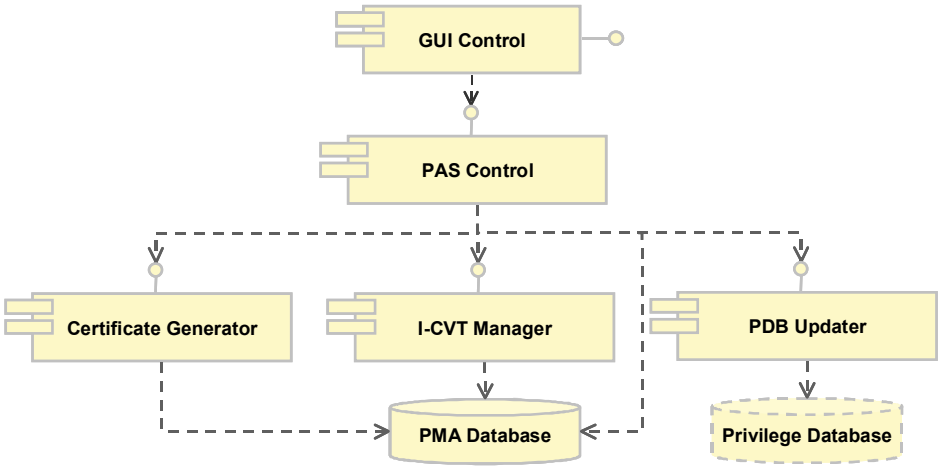


Abb. 7-13: Komponenten des PAMINA Administration Server

Im Folgenden wird auf die Aufgabe und Funktionsweise dieser Komponenten eingegangen:

- **GUI Control:** Diese Komponente ist zuständig für die dynamische Generierung und grafische Gestaltung (Layout) der Oberflächenelemente des Systems. Im Wesentlichen bedeutet dies das Zusammenstellen von HTML-basierten Seiten, welche die jeweilig erforderlichen Steuerungselemente (Menüleisten, Auswahllisten, usw.) sowie statischen Inhalte (Textfelder, Bilder, Fehlermeldungen, usw.) enthalten.
  Diese Seiten werden den PAS-Anwendern mit Hilfe eines „vorgeschalteten" Webservers übermittelt. Ebenfalls sorgt dieser Webserver für das Weiterreichen der durch einen Benutzer gewählten Aktionen, wie z.B. das Anklicken eines Menüpunkts, an den *GUI Control*.
  Die Konfiguration der jeweilig angezeigten Seiten ist Aufgabe der Komponente mit der Bezeichnung *PAS Control* (siehe unten): Diese bestimmt, je nach Situation, welche Daten (Fehlermeldungen, Listeninhalte, etc.) in den Seiten eingeblendet werden sollten.
  Die technologische Basis vom *GUI Control* bildet die *ASP.NET*-Technologie von Microsoft. Die notwendige Webserver-Funktionalität wird im Prototypen durch das Produkt *Internet Information Server* vom selbigen Hersteller erbracht.

- **PAS Control:** Diese Komponente instanziiert und steuert die anderen PAS-Komponenten in Abhängigkeit der durch die Anwender getätigten Aktionen.
  Die wichtigste funktionale Aufgabe dabei ist die Abbildung der Autorisierungsvorgänge auf Attributszertifikate (siehe das Schema hierfür in Abb. 7-11). Dabei steuert der *PAS Control* die für die Formatierung der auszustellenden Zertifikate zuständige Komponente *Certificate Generator* (siehe unten). Außerdem sorgt der *PAS Control* für die Konsistenzhaltung des Datenbestands (*PMA Database*) der PMA insbesondere wenn dort Zertifikate entfernt werden müssen.

Sollten beispielsweise Änderungen an einer vorhandenen Rolle geltend gemacht werden, so gibt der *PAS Control* die Ausstellung eines neuen Rollenzertifikats dem *Certificate Generator* durch den Aufruf der hierfür vorgesehenen Methode in Auftrag. Das neue Rollenzertifikat wird durch den *Certificate Generator* in der *PMA Database* abgelegt. Anschließend wird das dort noch vorhandene ursprüngliche Rollenzertifikat durch den *PAS Control* entfernt.
Da diese Operationen im I-CVT reflektiert werden müssen, ruft der *PAS Control* den *I-CVT Manager* (siehe unten) auf, um den Hashwert und Suchschlüssel des neuen Zertifikats in den I-CVT einzufügen bzw. diese Angaben zum gelöschten Zertifikat aus dem Baum zu entfernen. Wie schon oben erwähnt, entscheidet der *PAS Control* auch über die in einer gegebenen Situation anzuzeigenden Informationen, welche dem *GUI Control* übermittelt werden.
Im Fall von anzuzeigenden Autorisierungsdaten, d.h. Rollenzugehörigkeiten und Rechten eines gegebenen Benutzers oder den Inhalten einer Rolle, stützt sich der *PAS Control* auf den Zertifikatbestand in der *PMA Database*. Sollten beispielsweise die aktuellen Rechte des Benutzers *Bob* angezeigt werden (siehe Anwendungsfall „*Einsehen von Zugriffsrechten*“), so holt sich der *PAS Control* sämtliche für *Bob* ausgestellten Attributszertifikate sowie die mit diesen evtl. verknüpften Rollenzertifikate aus der *PMA Database*.
Anschließend extrahiert er die hierin enthaltenen relevanten Daten (Rechte, Gültigkeitsintervall, Delegierbarkeit, usw.) und reicht diese dem *GUI Control* weiter. Dieser kann nun die mit diesen Angaben parametrisierte Navigationskomponente (erweiterten CIB) einblenden.
Eine weitere Aufgabe des *PAS-Control* ist das periodische Starten des Prozesses, der auf der Seite einer PAS-Instanz durch den *PDB Updater* ausgeführt wird, und dessen Ergebnis aktualisierte Daten seitens der PDB sind. Der Vorgang hierfür wird in Kap. 7.3 genauer erklärt.

- **Certificate Generator:** Diese Komponente funktioniert als eine „Zertifikatsfabrik“ innerhalb einer PAS-Instanz. Getriggert wird sie durch den *PAS Control*, dessen Aufrufe zur Zertifikatsausstellung er mit Hilfe von Zertifikatsvorlagen ausführt. Als Eingabe erwartet der *Certificate Generator* die zu zertifizierenden „Rohdaten“, d.h. die Werte für die künftigen Zertifikatsfelder, wie z.B. *Holder*, *AttrCertValidityPeriod* und *Attributes*.
  Die Ausstellung eines neuen Zertifikats geht stets mit der Vergabe einer neuen, d.h. vorher noch nicht vergebenen, Seriennummer (*SerialNumber*), einher. Hierfür inkrementiert der *Certificate Generator* einen Zähler, dessen aktueller Wert die größte bereits vergebene Seriennummer angibt.
  Nach der erfolgreichen Erzeugung eines neuen Zertifikats gibt der *Certificate Generator* dem *PAS Control* eine Erfolgsmeldung. Abschließend legt er das neue Zertifikat in der *PMA Database* ab.

- **PMA Database:** Die *PMA Database* dient als Datenablage von Daten, die eine PAS-Instanz im Laufe der administrativen Tätigkeiten erzeugt. Die wichtigsten Daten sind die (unsignierten) Attributs- Rollen- und Rollenzuordnungszertifikate. Diese werden als Tupel der entsprechenden in Abb. 7-14 gezeigten Relationen *Attr_Cert*, *Role_Spec_Cert* und *Role_Member_Cert* abgelegt. Die einzelnen Attribute dieser Relationen deuten jeweils auf ein Zertifikatsfeld hin.

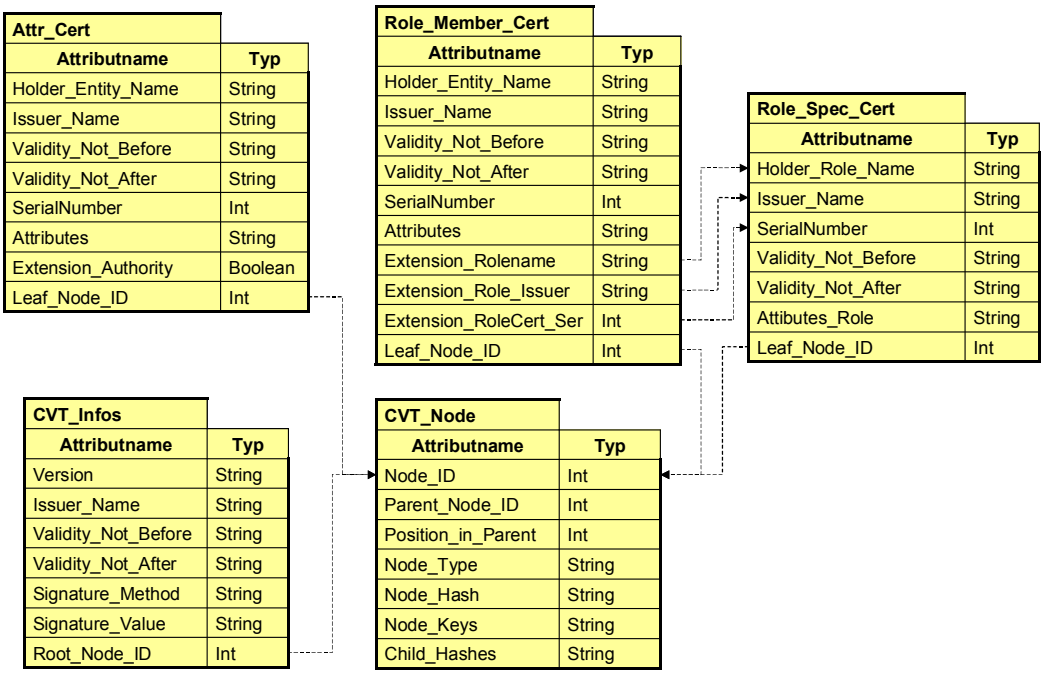


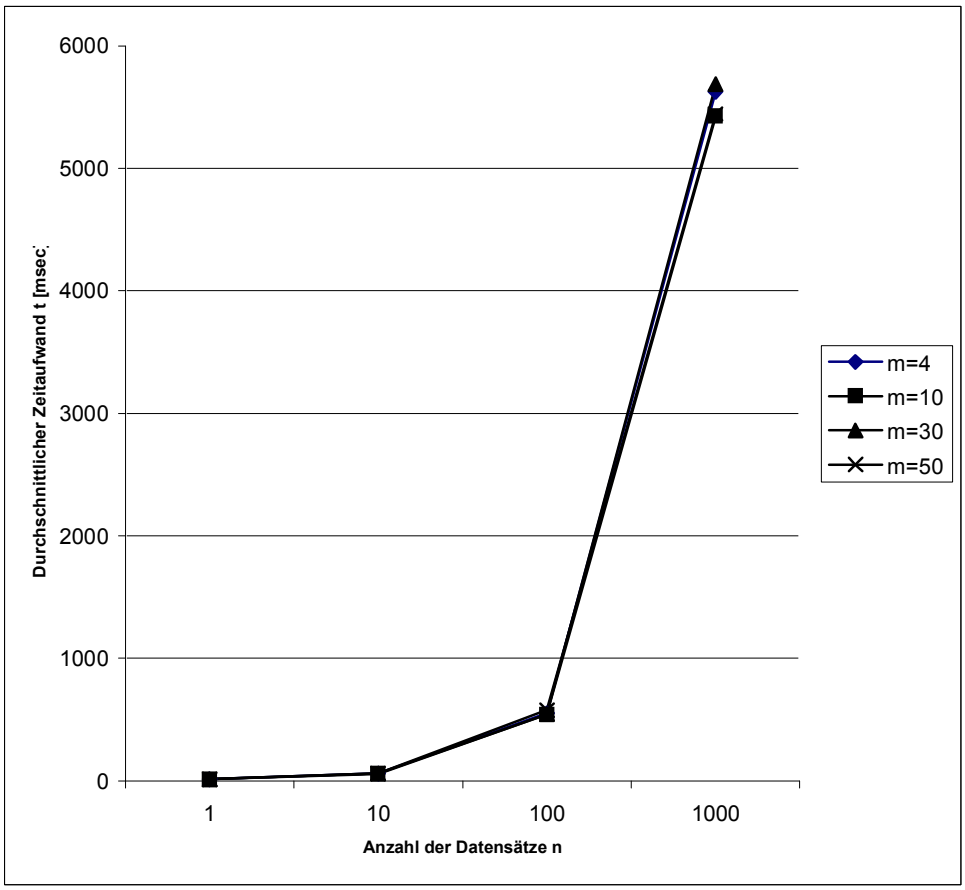
Abb. 7-14: Zertifikate und I-CVT als Relationen in der PMA Database

Neben diesen Angaben enthalten diese Relationen ein Attribut namens *Leaf_Node_ID*. Dieses stellt eine (redundante) Verbindung zu dem ebenfalls in der *PMA Database* abgelegten (aktuellsten) I-CVT her.
Dieser wird mit Hilfe der Relationen *CVT_Node* und *CVT_Infos* (siehe Abb. 7-14) persistent gespeichert. Anhand von *CVT_Node* lässt sich die Struktur des I-CVT sowie der einzelnen Knoten des Baums rekonstruieren, was Aufgabe des *I-CVT Manager* ist (siehe unten). In der Relation *CVT_Infos* werden hingegen Daten gespeichert, die in die Berechnung der ebenfalls hier abgelegten Wurzelsignatur (*Signature_Method*) einfließen.
Die technische Basis der *PMA Database* bildet das Produkt *Microsoft SQL Server 2000*.

- **I-CVT Manager:** Diese Komponente implementiert mit Hilfe entsprechender Algorithmen die Verwaltung des I-CVT der jeweiligen PMA. Die wichtigsten Schritte sind hierbei das Einfügen von neuen und das Löschen von bestehenden Einträgen, die Balancierung des Baums und das Signieren der Wurzel.
  Der *I-CVT Manager* führt diese Operationen an einer als *C#*-Objekt repräsentierten Instanz der aktuellen I-CVT durch. Diese wird beim Starten des PAS während der Initialisierung des *I-CVT Manager* erzeugt. Der Aufbau dieses „objektorientierten" Baums erfolgt anhand von der in der *PMA Database* abgelegten Relation *CVT_Node*: In einem rekursiven Vorgang ausgehend von den Blättern (also Tupeln mit *Node_Type="Leaf"*) und mit Hilfe der in den Knoten abgelegten Verweise auf den jeweiligen Vaterknoten (*Parent_Node_ID*) sowie der Angaben zur Verzweigung (*Position_in_Parent*) wird der Baum erzeugt.
  Das dabei entstehende I-CVT-Objekt verfügt über Methoden zum Einfügen/Löschen von Zertifikaten, zum Balancieren des Baums, und auch zur Berechnung der Hashwerte (*Node_Hash*) mit der Hashfunktion SHA-1 (*SHA1CryptoServiceProvider*) [NIST95]. Diese Funktionen werden in Abhängigkeit der weiter oben geschilderten Benutzeraktionen aufgerufen. Das Einfügen/Löschen der Zertifikate in den I-CVT wird in Log-Einträgen festgehalten.

Ein Log-Eintrag enthält die jeweils getätigte Aktion und die (lokal) eindeutige Kennung (*Serialnumber*) des betroffenen Zertifikats. Diese Log-Einträge finden ihre Anwendung während der Aktualisierung der PDB (siehe Kap. 7.3).
Wie schon gesagt, wird die Wurzelsignatur in periodischen Abständen regelmäßig, mit Hilfe des RSA-Verfahrens [RSA78], neu berechnet. Dabei setzt der *I-CVT Manager* auch den neuen Wert der Gültigkeitsperiode für den gesamten Baum. Nach erfolgreicher Aktualisierung dieser Angaben speichert der *I-CVT Manager* die Knoten des neuen I-CVT wieder in Form von Tupeln der Relation *CVT_Node* in der *PMA Database* ab. Ebenfalls wird die Relation *CVT_Infos* mit den neuen Werten von *Signature_Value*, *Validity_Not_Before* und *Validity_Not_After* überschrieben.

Das in Abb. 7-15 gezeigte Diagramm und die zugehörige Tabelle geben den am Versuchssystem ermittelten durchschnittlichen Zeitaufwand *t* gemessen in Millisekunden zum Aufbau von I-CVTs durch den *I-CVT Manager* an. Betrachtet wurden dabei anfangs leere B^+-Bäume der Ordnung *m={4, 10, 30, 50}* (siehe Def. 5-5), in welche jeweils *n={1, 10, 100, 1000}* Datensätze nacheinander eingefügt wurden (siehe Achse *n*).



m/n	*1*	*10*	*100*	*1000*
4	15.0	62.0	550.8	5624.7
10	15.0	60.4	541.6	5423.0
30	15.0	58.8	544.7	5687.0
50	14.5	62.0	577.3	5443.3

Abb. 7-15: Zeitaufwand in msec zum Einfügen von Zertifikaten in I-CVTs

Diese Datensätze sind fiktive Zertifikate (semantikfreie Zeichenketten) mit der einheitlichen Länge von 4 Kilobyte. Diesen Zertifikaten wurden für die Zwecke der Messungen Suchschlüssel des Typs {*Holder*||*SerialNumber*} im jeweiligen Baum zugeordnet. Dabei wurde jeder verwendete Wert für den Schlüsselteil *Holder* maximal *n/2*-fach vergeben.
Dargestellt sind Durchschnittswerte von zehn Messungen für jede betrachtete Baumordnung und -größe. Erfasst wurden die durchschnittliche Summe der Ausführungszeiten von folgenden Operationen an den I-CVTs:

- Berechnung der Hashwerte der einzufügenden (fiktiven) Zertifikate,
- Generierung von neuen (d.h. noch nicht vergebenen) Suchschlüsseln,
- Einfügen der Suchschlüssel und der Hashwerte in den jeweiligen Baum, wobei das Anlegen oder Spalten von Blättern, Balancieren des Baums sowie (Neu-) Berechnen der betroffenen Hashwerte erfolgen.

Zwischen den einzelnen Wiederholungen erfolgte dabei eine Neuinitialisierung des *I-CVT Managers* und damit der bearbeiteten I-CVT-Objekte. Als Laufzeitumgebung für diese sowie für alle anderen weiter unten noch behandelten Zeitmessungen diente ein PC-System mit einem 1,8 GHz Intel Pentium Prozessor und 1 GB Hauptspeicher. Das eingesetzte Messverfahren basiert auf der Funktion *GetTickCount()* der Windows API. Diese Funktion liefert zum Zeitpunkt deren Aufrufens die seit dem letzten Systemstart vergangene Zeit gemessen in Millisekunden. Die Zeitmessungen erfolgten durch die Differenzbildung der Rückgabewerte zwischen zwei Aufrufen von *GetTickCount()*.
Anhand von Abb. 7-18 lässt sich feststellen, dass die ermittelten Werte zur Generierung von kompletten Bäumen nur unwesentlich von der jeweiligen Baumordnung abhängen. Eine mögliche Erklärung hierfür ist, dass die einzelnen I-CVT-Objekte im Hauptspeicher des Versuchsrechners gehalten und manipuliert wurden. Bei dieser Speicherungsart scheint das Tiefen-Breite-Verhältnis der Bäume eine eher marginale Rolle zu spielen.

- **PDB Updater:** Wie der Name schon sagt, führt diese Komponente die PAS-seitigen Aktivitäten während der Aktualisierung der PDB. Ihre Aufgabe ist dabei die Kommunikation zwischen dem PAS und der PDB (nach einer initialer Anmeldung) abzuwickeln und dafür zu sorgen, dass die PDB stets die korrekten Daten erreichen. Diese Daten sind im Wesentlichen eine Befehlssequenz zur PDB-seitigen strukturbedingten Aktualisierung des jeweiligen I-CVT, die neue Wurzel des Baums (inklusive *CVT-Infos* und Wurzelsignatur) sowie die neuen Zertifikate, die in der PDB abgelegt werden müssen. Die Befehlsequenz wird anhand der oben erwähnten Log-Einträge erzeugt, welche die zuletzt am I-CVT durchgeführten Operationen dokumentieren.
  Die Übertragung dieser Daten an die PDB erfolgt mit Hilfe eines SOAP-basierten Webservice. Dabei ruft der *PDB Updater* die entsprechende Funktion (siehe Anwendungsfall „*Aktualisierung von Daten*“ in Kap. 7.3) auf der Seite der PDB auf.

7.3 Privilege Database

Die *Privilege Database* (PDB) dient hauptsächlich als Anlaufstelle für die durch PAMINA abgesicherten Anwendungen. Sie speichert die von den Zertifizierungsstellen (PMA) hier abgelegten und regelmäßig aktualisierten I-CVTs und evtl. auch die zugehörigen Zertifikate (dies ist auch der Fall im ed.tec-Szenario). Eine weitere operative Aufgabe der PDB ist die Aushändigung angeforderter Zertifikate an die hierfür berechtigten Systeme sowie die Generierung und Auslieferung der passenden Beweise anhand des zugehörigen I-CVT.
In Abb. 7-16 wird das Anwendungsfalldiagramm der PDB gezeigt. Wie schon erwähnt, erreichen Zertifikate und die zur Aktualisierung der I-CVTs benötigten Daten die PDB über Instanzen des *PAMINA Administration Server* (PAS) der einzelnen Zertifizierungsstellen (siehe Anwendungsfall „*Aktualisierung von Daten*“). Die Konsumenten der Zertifikate sowie der anhand der I-CVTs generierten Beweise sind hingegen die *Certificate Verifier*, welche diese Daten hier herunterladen und anschließend überprüfen können („*Herunterladen von Zertifikaten*“ und „*Herunterladen von Beweisen*“). Beide Arten von Dienstnehmern müssen sich bei der PDB authentifizieren, bevor die vorgesehene Funktionalität zugänglich gemacht wird („*Anmeldung*“, „*Abmeldung*“).

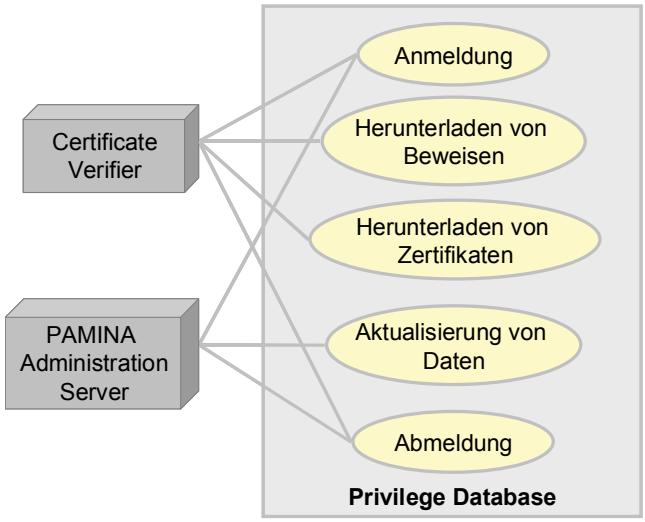


Abb. 7-16: Anwendungsfalldiagramm Privilege Database (PDB)

- **Aktualisierung von Daten:** Während der Aktualisierung der PDB durch eine bereits angemeldete PAS-Instanz wird durch die PAS-Komponente *I-CVT Updater* eine Befehlsequenz gesendet. Diese bestimmt die strukturbedingten Änderungen am aktuell in der PDB vorliegenden I-CVT, welcher von der jeweiligen PMA stammt. Anhand der enthaltenen (entsprechend kodierten) Befehle, wie z.B. „*Lösche Zertifikat mit Seriennummer 42*“ oder „*Füge Zertifikat mit Seriennummer 69 ein*“, berechnet die PDB den neuen I-CVT und legt diesen im eigenen Datenbestand ab. Das hiermit verfolgte Ziel ist, die (aufwändige) Übertragung kompletter Bäume zu vermeiden.
  Neben den Befehlen zur Änderung des I-CVT werden die seit der letzten Aktualisierung neu hinzugekommenen Zertifikate sowie die neue Wurzel (insbesondere die neue Gültigkeitsperiode und Wurzelsignatur) ebenfalls an die PDB geschickt und dort gespeichert. Diese Angaben kann die PDB ja nicht selbstständig errechnen.

- **Herunterladen von Zertifikaten:** Anwendungen, die nach dem Server-Pull-Prinzip aufgebaut sind, können durch die PDB mit (unsignierten) Zertifikaten versorgt werden. Dies ist auch im ed.tec der Fall: Die PDB stellt die hier abgelegten Attributs-, Rollen- und Rollenzuordnungszertifikate für angemeldete *Certificate Verifier* zur Verfügung. Die derzeitige Implementierung der PDB sieht zwei Arten von Anfragen vor:

  1. Es können sämtliche Attributs-, und Rollenzuordnungszertifikate eines bestimmten ed.tec-Benutzers über die hierfür vorgesehene Schnittstelle abgefragt werden. Hierfür müssen der Benutzername (*Holder*) sowie der Name der jeweiligen vertrauten PMA (*Issuer*) angegeben werden. Anhand der daraufhin erhaltenen Zertifikate wird das weiter oben erwähnte ed.tec-Benutzerprofil erzeugt (siehe dazu mehr in Kap. 7.4)

  2. Es können Zertifikate einzeln abgefragt werden, vorausgesetzt ihre eindeutige Kennung ist der Anwendung bekannt. Im Fall von ed.tec wird diese Funktionalität ausschließlich bei der Ermittlung von Rollenzertifikaten gebraucht. Ihre Kennung wird den Rollenzuordnungszertifikaten des gefragten Benutzers entnommen.

- **Herunterladen von Beweisen:** Die Aushändigung von Zertifikaten geht normalerweise mit der Generierung und Auslieferung von Beweisen einher, die anhand der jeweilig relevanten I-CVT durch die PDB generiert werden. Für das ed.tec-System werden Beweise im Einklang mit den oben geschilderten Anfragen bereitgestellt:

  1. Die Beantwortung der Anfrage nach allen für einen Benutzer (*Holder*) ausgestellten Zertifikaten kann mit Hilfe eines Vollständigkeitsbeweises überprüft werden. Dieser wird ausgehend vom I-CVT der jeweiligen PMA generiert, deren Knoten (Blätter) durch den Suchschlüssel *Holder||SerialNumber* (entnommen aus den gleichnamigen Zertifikatsfeldern) indiziert wurden.

  2. Werden hingegen Anfragen nach einzelnen Zertifikaten an die PDB gestellt, so generiert diese „einfache“ Zertifizierungspfade, die entweder die Existenz oder eben das Fehlen des gefragten Zertifikats bestätigen.

Die Prüfung dieser Beweise erfolgt mit Hilfe der PAMINA-Komponente *Certificate Verifier* (siehe Kap. 7.4).

7.3.1 Interner Aufbau

Die PDB wurde hauptsächlich im Rahmen der Arbeiten [Oet02] [Küh03] und [Vie03] (weiter-)entwickelt. Ihre in Abb. 7-17 dargestellten funktionalen Komponenten wurden in der Programmiersprache *C#* implementiert.

- **PDB Control:** Diese Komponente steuert die Vorgänge innerhalb der PDB. Sie sorgt auch für die Instanziierung und Beendigung der anderen oben abgebildeten Komponenten. Der *PDB Control* wird hauptsächlich durch den *PMA Service* und den *Verification Service* getriggert. Diese bilden die Schnittstellen der PDB für die angemeldeten PAS-Instanzen respektive für die abgesicherten Anwendungen (*Certificate Verifier*).

Die dabei erwartete Funktionalität wird durch den *Data Manager* bzw. den *Proof Generator* erbracht. Das Zusammenspiel der PDB-Komponenten wird anhand der oben skizzierten Anwendungsfälle „*Aktualisierung der Daten*“ und „*Generierung von Beweisen*“ dargestellt:

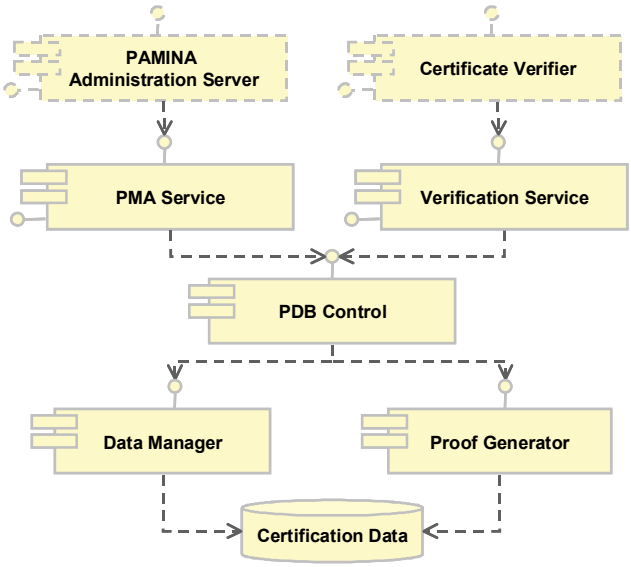


Abb. 7-17: Komponenten der PDB

Meldet sich eine PAS-Instanz bei der PDB über den *PMA Service* zwecks Aktualisierung der (eigenen) Daten an, so erzeugt der *PDB Control* eine neue Instanz der Komponente *Data Manager*. Dieser ist exklusiv für die Aktualisierungen des Datenbestands, d.h. I-CVT und Zertifikate, der jeweiligen PMA zuständig. Nach Abmeldung der PAS-Instanz wird der zugehörige *Data Manager* durch den *PDB Control* zerstört.
Ähnliches gilt für eine über den *Verification Service* erfolgreich angemeldete Anwendung (z.B. ed.tec): Der *PDB Control* instanziiert für jede zu bedienende Anwendung einen eigenen *Proof Generator*, welcher anhand eines im Hauptspeicher aufgebauten I-CVT die jeweilig erforderlichen Beweise generiert. Ein *Proof Generator* bleibt so lange „am Leben“ erhalten bis sich die Anwendung wieder abgemeldet hat.
Eine wichtige Aufgabe des *PMA Control* ist die Überwachung der Datenkonsistenz: Nach der Aktualisierung eines I-CVT fordert er die betroffenen *Proof Generators* auf, den im eigenen Speicherbereich liegenden I-CVT mit der aktualisierten Version zu ersetzen.

- **PMA Service:** Der *PMA Service* bildet die Kommunikationsschnittstelle zu den PAS-Instanzen hin. Er sorgt für deren (passwortbasierte) Authentifizierung. Dies ist wichtig, um unberechtigte Änderungen an den Daten (Zertifikaten, I-CVTs) zu verhindern. Des Weiteren erledigt diese Komponente die Weiterleitung der von einer PMA erhaltenen Aktualisierungsdaten an den zuständigen *Data Manager*. Der *PMA Service* wurde als ein SOAP-basierter Web-Service implementiert. Er ist in der Lage, gleichzeitig mehrere *PMA Server* zu bedienen, d.h. ihre Aktualisierungswünsche entgegenzunehmen. Der *PMA Service* läuft im Prozessraum eines Web-Servers, im Prototypen wurde hierfür der *Internet Information Server* der Firma Microsoft verwendet.

- **Data Manager:** Jede Instanz der Komponente *Data Manager* pflegt jeweils die einer PMA (*Issuer*) zugeordneten Relationen innerhalb der Datenablage *(Certification Data)* der PDB. Die Pflege der Daten geschieht anhand der durch den zugeordneten *PAS* gesendeten Einfüge- oder Löschbefehle, Zertifikate sowie weiteren Aktualisierungsangaben zum I-CVT der PMA. Erhaltene (neue) Zertifikate bildet der *Data Manager* auf die oben in Abb. 7-14 bereits gezeigten Relationen ab und speichert diese im Datenbestand *Certification Data*. Zu löschende Zertifikate (angegeben durch ihre eindeutige Kennung) werden aus dem Datenbestand entfernt.
  Des Weiteren erfolgt hier die Berechnung des aktualisierten I-CVT. Dabei führt der *Data Manager* sehr ähnliche Schritte aus, wie die oben diskutierte Komponente *I-CVT Manager*: Die Manipulation des I-CVT (Einfügen oder Löschen von Zertifikaten, Balancierung, Hashwertberechnungen) erfolgt mit Hilfe eines I-CVT-Objekts, das zunächst ausgehend von den im PDB-Datenbestand vorhandenen Relationen aufgebaut wird. Ein wesentlicher Unterschied zum Ablauf innerhalb des *I-CVT Managers* im PAS ist, dass hier keine Wurzelsignatur berechnet wird. Statt dessen wird die von der jeweiligen PAS-Instanz erhaltene Wurzelsignatur in der Datenbank (als Tupel der Relation *CVT_Infos,* siehe Abb. 7-14) abgelegt.
  Nach der Durchführung der Berechnungen werden auch hier die Knoten des neuen I-CVT in Relationen transformiert und im Datenbestand der PDB gespeichert. Die Fertigstellung der Aktualisierung wird dem *PDB Control* gemeldet, der dann dafür sorgt, dass die Zugriffskontrolle auf der Anwenderseite anhand der neuen Daten erfolgt.

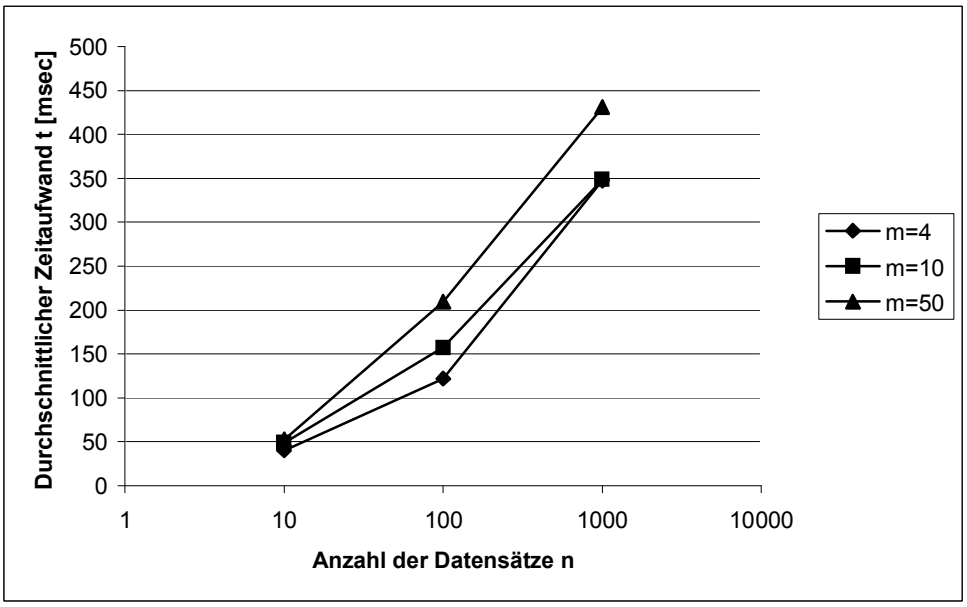
- **Certification Data:** Als Folge der Aktualisierungen befindet sich in diesem Datenbestand eine Replizierung sämtlicher aus Sicht der abgesicherten Anwendungen relevanter Daten von allen registrierten PMAs. Diese Daten sind im Fall von ed.tec die Attributs-, Rollen-, und Rollenzuordnungszertifikate sowie die diese signierenden I-CVTs. Die diese speichernden Relationen sind identisch mit den in Abb. 7-14 gezeigten Relationen. Dieser Komponente liegt eine Instanz des RDBMS *Microsoft SQL Server 2000* zugrunde. Die hierin enthaltenen Daten werden dem entsprechend mittels in den Programmcode des *Data Managers* eingebetteten SQL-Befehlen verwaltet.

- **Verification Service:** Diese Komponente bildet die Schnittstelle, über welche angemeldete Anwendungen, d.h. *Certificate Verifier*, ihre oben beschriebenen Anfragen nach Zertifikaten und den passenden Beweisen bei der PDB absetzen können. Ein Ziel der Authentifizierungen ist auszuschließen, dass Unbefugten Zertifikate ausgehändigt werden, aus denen sie ansonsten Benutzernamen (*Holder*), Berechtigungen (*Attributes*) etc. erfahren könnten. Diese Komponente wurde ebenfalls als ein über SOAP angebotener Web-Service realisiert und ist mit Hilfe des oben genannten Web-Servers für die Anwendungen erreichbar.

- **Proof Generator:** Der *Proof Generator* generiert ausgehend von einem vorliegenden aktuellen I-CVT Vollständigkeits- bzw. Existenzbeweise je nach Bedarf (siehe noch Kap. 7.4.1). Im Fall von ed.tec werden Beweise zur Sicherung von Intervallsuchen benötigt. Die Vorgehensweise und die hierbei durch den *Proof Generator* verwendeten Algorithmen hierfür wurden anhand von Beispielen in Kap. 5.3 im Detail behandelt und werden hier nicht wiederholt.

Die Basis der Beweisgenerierung ist ein im Hauptspeicher residierendes I-CVT-Objekt. Dieses wird anhand der in der PDB (*Certification Data*) vorliegenden Relationstupeln (*CVT_Node*, *CVT-Infos*) durch den *Proof Generator* nach jeder Aktualisierung des Baums aufgebaut. Dies wird durch das gleiche Programmmodul realisiert, das auch im *I-CVT Manager* (siehe Kap. 7.2.2) seine Dienste leistet.
Neben der Beweisgenerierung ist der *Proof Generator* auch für die Auslieferung angefragter (unsignierter) Zertifikate zuständig, angenommen sie sind in der PDB abgelegt worden. Hierfür wird auf den PDB-Datenbestand zurückgegriffen und dort gemäß der Anfrage nach passenden Daten gesucht. Anfragende Anwendungen, wie z.B. ed.tec, erreichen die schließlich gefundenen Zertifikate über den *Verification Service*.

Das in Abb. 7-18 gezeigte Diagramm und die zugehörige Tabelle geben den am Versuchssystem ermittelten durchschnittlichen Gesamtzeitaufwand *t* gemessen in Millisekunden für die Aushändigung von Zertifikaten sowie zur Generierung von Vollständigkeitsbeweisen durch den *Proof Generator* an. Die Beweise wurden ausgehend von B^+-Bäumen der Ordnung *m={4, 10, 50}* (siehe Def. 5-5) generiert, wobei die Bäume jeweils *n={10*, *100*, *1000}* Datensätze signieren. Betrachtet wurden dabei bereits im Hauptspeicher der PDB vorliegende I-CVTs, die zuvor abgelegte (fiktive) Attributszertifikate mit Hilfe von Suchschlüsseln vom Typ {*Holder*||*SerialNumber*} erfassen. Die erzeugten Beweise zielten jeweils auf *k=[0; n/2]* Zertifikate eines Benutzers (*Holder*), die (den geltenden Sortierregeln entsprechend) Intervalle der Breite *k* auf der Blattebene der einzelnen Bäume bildeten.



m/n	*10*	*100*	*1000*
4	40.1	121.8	347.6
10	49.0	157.3	348.6
50	52.5	209.5	430.8

Abb. 7-18: Zeitaufwand zur Generierung von Vollständigkeitsbeweisen

Anhand von Abb. 7-18 lässt sich erkennen, dass größer gewählte Baumordnungen (zumindest bei den hier betrachteten Baumgrößen) zu einem höheren Zeitaufwand führen können.

Bemerkenswert ist außerdem die große Abweichung (stellenweise bis zu 100%) der in den Versuchen ermittelten Einzelwerte von den hier dargestellten Durchschnittswerten. Dies ist neben den unterschiedlich ausfallenden Beweisgrößen auch auf das instabile Laufzeitverhalten des PDB-Prototyps zurückzuführen.

7.4 Certificate Verifier

Die PAMINA-Komponente mit der Bezeichnung *Certificate Verifier* (im Weiteren als *Verifier* genannt) wurde entwickelt, um abgesicherte Anwendungen mit gültigen, d.h. überprüften, Benutzerrechten zu versorgen.
Ein Ziel war dabei Anwendungen bzw. Anwendungsentwicklern eine möglichst einfache Nutzung von PAMINA insbesondere deren in Kap. 7.3 diskutierten Komponente *Privilege Database* (PDB)zu ermöglichen. Erprobt wurde dies im Fall von ed.tec [Boc03] [Vie03]. Der dabei entstandene *Verifier*-Prototyp kann leider keineswegs eine universell einsetzbare Komponente genannt werden.

Abb. 7-19: Anwendungsfalldiagramm zum Verifier

Die Realisierung des zunächst generisch klingenden Anwendungsfalls „*Ermittlung von gültigen Benutzerrechten*“ (siehe Abb. 7-19) ist nämlich stark vom in Kap. 7.1.2.1 beschriebenen ed.tec-Autorisierungsmodell geprägt:

- **Ermittlung von gültigen Benutzerrechten:** Es wurde für die Zugriffskontrolle im ed.tec die möglichst schnelle und sichere Zusammenstellung eines vollständigen Benutzerprofils angestrebt, sobald sich ein Benutzer erfolgreich am System angemeldet hatte (siehe Kap. 7.1.2.2).
  Dieses Profil wird im weiter oben beschriebenen *Privilege Cache* abgelegt und bildet die Grundlage der Zugriffsentscheidungen, die durch die Komponente *Privilege Evaluator* getroffen werden (siehe auch Abb. 7-4).
  Hierzu werden im Fall von ed.tec die jeweilig relevanten Attributs-, Rollen- sowie Rollenzuordnungszertifikate von der zuständigen PDB-Instanz angefordert.
  Dabei werden zuerst sämtliche Attributs- und/oder Rollenzuordnungszertifikate heruntergeladen, für die gilt, dass sie für den jeweilig Zugreifenden (*Holder*) durch eine bestimmte der Zugriffskontrolle vertraute Stelle (*Issuer*) ausgestellt wurden.
  Zu den ausgelieferten Zertifikaten (evtl. keine!) wird bei der PDB ein Vollständigkeitsbeweis angefordert. Dieser wird dort ausgehend vom I-CVT der jeweiligen Zertifizierungsstelle (*Issuer*) generiert und dem *Verifier* ausgehändigt.
  Anhand dieses Beweises kann der *Verifier* zum Einen die Zertifikate auf ihre aktuelle Gültigkeit überprüfen, welche er auf die obige Anfrage schließlich erhalten hat. Zum Anderen kann er feststellen, ob die (evtl. korrupte) PDB-Instanz aus Sicht des entstehenden Benutzerprofils relevante Zertifikate zurückgehalten hat. Ist dies der Fall, so wird der Vorgang abgebrochen. (In einer realen Betriebsumgebung sollte sich der *Verifier* an dieser Stelle die Zertifikate bei einer anderen PDB-Instanz anfordern.)

Gibt es unter den zuvor erhaltenen Zertifikaten ein oder mehrere Rollenzuordnungszertifikate, die also Rollenmitgliedschaften des gefragten Benutzers festlegen, so wird die PDB aufgefordert, die passenden Rollenzertifikate auszuliefern.
Dies erfolgt durch Angabe die Namen des Ausstellers (*Issuer*) und der Rolle (*Holder*) sowie die Seriennummer (*SerialNumber*) des Rollenzertifikats.
Diese Daten können aus den einschlägigen Feldern des jeweiligen Rollenzuordnungszertifikats entnommen werden (siehe auch das Beispiel in Abb. 7-10).
Da es sich hierbei um „Einzelzertifikate" handelt, wird durch die PDB zu jedem angeforderten Zertifikat je nach Erfolg der Suche ein Existenz- bzw. Nicht-Existenz-Beweis generiert und ausgeliefert. Läuft die Prüfung dieser Beweise problemlos ab, d.h. ohne Verdacht auf jegliche Manipulationen, so wird der Vorgang fortgesetzt.
Dabei werden aus den erhaltenen und für gültig befundenen Zertifikaten die für das ed.tec-System (genauer für die Komponente *Privilege Evaluator*) interpretierbaren Benutzerrechte extrahiert. Diese findet der *Verifier* unter dem Zertifikatsfeld *Attributes*.
Außerdem wird noch der (erwartete) Zeitpunkt der nächsten Aktualisierung des jeweiligen I-CVT zurückgeliefert, was seinerseits die Wiederholung des gesamten Vorgangs (und die Erneuerung des *Privilege Cache* im ed.tec) erforderlich macht. Es steht jedoch in der Verantwortung der abgesicherten Anwendung, diesen Vorgang in Kenntnis des nächsten erwarteten Aktualisierungszeitpunkts der Daten erneut zu starten.

7.4.1 Interner Aufbau

Der interne Aufbau sowie die „Außenbeziehungen" des *Verifiers* werden in Abb. 7-20 gezeigt. Er besteht aus den funktionalen Komponenten *Verifier Control*, *PDB Client* und *Proof Validator*, die in der Programmiersprache *C#* geschrieben wurden [Vie03]. Diese können nach entsprechenden syntaktischen (z.B. Portierung auf andere Programmiersprachen) sowie semantischen (z.B. Unterstützung anderer Autorisierungsmodelle) Anpassungen in weiteren Anwendungssystemen eingesetzt werden.

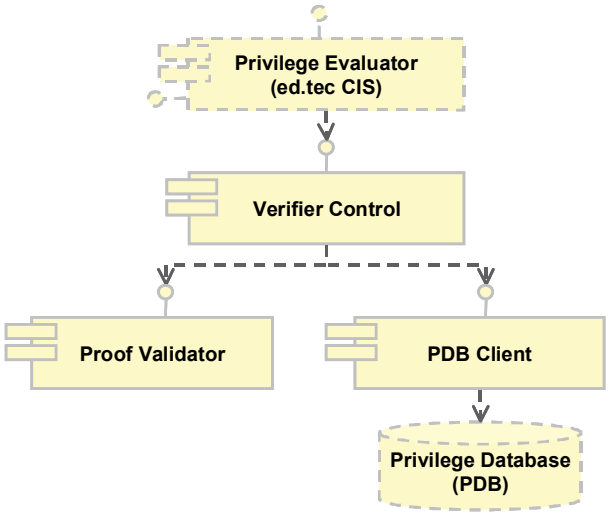


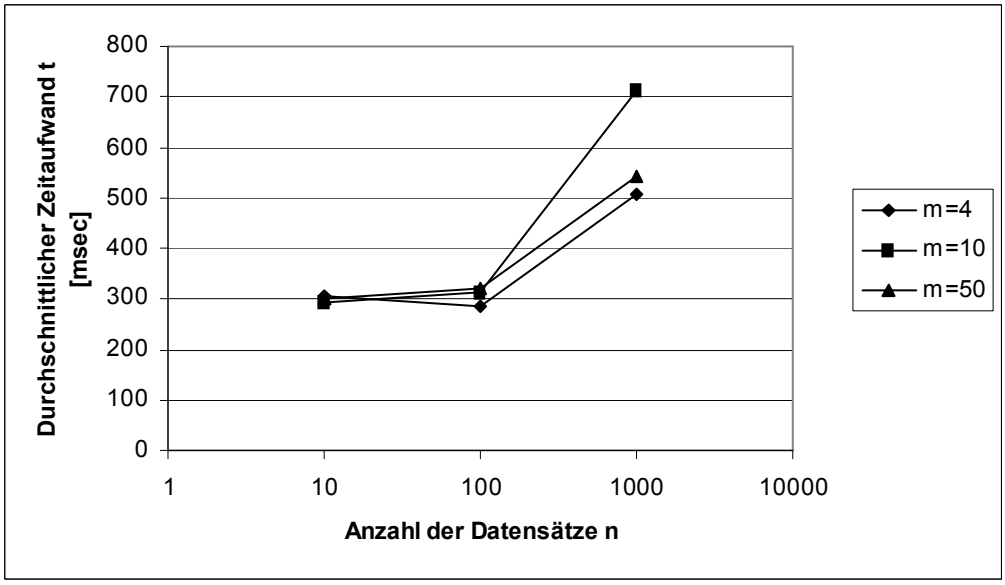
Abb. 7-20: Komponenten des Certificate Verifier

- **Verifier Control:** Aufgabe dieser Komponente ist die Steuerung der internen Abläufe beziehungsweise der beiden anderen Komponenten des *Verifiers* während der Ausführung des oben geschilderten Anwendungsfalls „*Ermittlung von gültigen Benutzerrechten*“.
  Der *Verifier Control* wird durch die jeweilig bediente Anwendung erzeugt und getriggert, in dem er zur Auslieferung der Benutzerrechte eines bestimmten Benutzers aufgefordert wird. Im Fall von ed.tec ist für das Absetzen dieser Anforderungen die Komponente des erweiterten CIS mit der Bezeichnung *Privilege Evaluator* zuständig (siehe Abb. 7-20 oben).
  Sobald den *Verifier* die erste Anfrage erreicht, erzeugt der *Verifier Control* jeweils eine Instanz der Komponenten *PDB Client* und *Proof Validator*.
  Ersterer ist für das Absetzen von spezifischen Zertifikatsanfragen an die *Privilege Database* (PDB) und die Abwicklung der hierfür notwendigen Kommunikation zuständig.
  Letztere Komponente hingegen erbringt die vom *Verifier* erwartete Sicherheitsfunktionalität, nämlich die Überprüfung der erhaltenen Zertifikate mit Hilfe der zugehörigen Vollständigkeitsbeweise.
  Im Fall eines fehlerfreien Durchlaufs der durch diese beiden Komponenten realisierten Vorgänge und Algorithmen ermittelt der *Verifier Control* die in den vorliegenden geprüften Zertifikaten enthaltenen Benutzerrechte, d.h. Fähigkeiten. Diese werden in Form einer Liste an die anfragende Anwendung (ed.tec) ausgegeben.

- **PDB Client:** Diese Komponente bildet den Gegenpart zum auf der Seite der PDB laufenden SOAP-basierten *Verification Service* (siehe Kap. 7.3.1). Nach seiner Instanziierung durch den *Verifier Control* wickelt der *PDB Client* zunächst den initialen Verbindungsaufbau und die anschließende passwortbasierte Anmeldung bei der PDB ab.
  Die weiter oben geschilderten Anfragen an die PDB führt der *PDB Client* (auf Kommando des *Verifier Control* hin) durch Aufrufen und Parametrisieren der entsprechenden Methoden der PDB-Komponente *Verification Service* aus (siehe auch die PDB-Anwendungsfälle „*Herunterladen von Zertifikaten*“ und „*Herunterladen von Beweisen*“ in Kap. 7.3.1).
  Beim Herunterladen von Zertifikaten gibt der *PDB Client* stets den Namen der ihm vertrauten Stelle (*Issuer*) als Eingabeparameter an. Diesen Namen entnimmt er einer Konfigurationsdatei. Beim Ermitteln sämtlicher durch diese Stelle an einen bestimmten Benutzer ausgestellter Zertifikate gibt er zusätzlich dessen Namen (*Holder*) an. Anhand dieser Informationen kann die PDB eine Suche nach den passenden Zertifikaten des Benutzers im Datenbestand durchführen. Außerdem kann sie den jeweiligen I-CVT finden (aufbauen), der vom *Issuer* stammt. Da der gefragte *Issuer* den eigenen I-CVT zuvor mit Hilfe des Suchschlüssels *Holder||SerialNumber* erstellt bzw. sortiert hat, kann die PDB (*Proof Generator*) nun einen Vollständigkeitsbeweis für die Intervallsuche nach allen s_i Suchschlüsseln mit $Holder||min(SerialNumber) \leq s_i \leq Holder||max(SerialNumber)$ generieren. Dieser Beweis (dessen Strukturierung in Kap. 5.3.4 behandelt wurde) wird nach Erhalt der Zertifikaten (evtl. keine) durch den *PDB Client* angefordert.
  Beim Herunterladen von „Einzelzertifikaten“, d.h. im Fall von ed.tec Rollenzertifikaten, übergibt der *PDB Client* der PDB neben dem Namen der vertrauten PMA (*Issuer*) auch den Rollennamen (*Holder*) und die Seriennummer (*SerialNumber*) des gefragten Zertifikats.

Daraufhin erwartet der *PDB Client* genau ein Zertifikat sowie im Anschluss den entsprechenden Existenz-Beweis, oder ggf. einen Nicht-Existenz-Beweis.
Dieser Beweis bestätigt die aktuelle Gültigkeit oder eben das Fehlen des Zertifikats mit dem Suchschlüssel *Holder=Rollenname||SerialNumber=Seriennummer* eindeutig. Die von der PDB erhaltenen Zertifikate und die zugehörigen Beweise gibt der *PDB Client* zur weiteren Bearbeitung dem *Verifier Control* weiter.

- **Proof Validator:** Der *Proof Validator* ist für die Prüfung der durch den *PDB Client* bei der PDB abgeholten Beweise zuständig. Er arbeitet gemäß der in Kap. 5.3 detailliert behandelten Algorithmen zur Prüfung von Vollständigkeits- bzw. Existenz-Beweisen. Dabei werden die folgenden (groben) Schritte ausgeführt:
  Zunächst wird anhand der in den Beweisen enthaltenen Hashwerte und Suchschlüssel der Wurzelhashwert des Baums rekonstruiert. Während dessen wird in jedem betrachteten Knoten des Baums festgestellt, ob dort Hinweise (Suchschlüssel) auf relevante aber durch die PDB verschwiegene Zertifikate enthalten sind. Ist das der Fall, so wird (wie oben bereits erwähnt) der Prüfvorgang abgebrochen.
  Der ermittelte Wurzelhashwert ermöglicht zusammen mit den in den Beweisen enthaltenen *CVT-Infos* (siehe Beispiel in Abb. 7-12) die Prüfung der Wurzelsignatur, die ebenfalls Bestandteil des geprüften Beweises sein muss.
  Für die Signaturprüfung wird der öffentliche Schlüssel der jeweiligen Zertifizierungsstelle benötigt, der im *Verifier*-Prototypen einer Konfigurationsdatei entnommen wird. Ist ein Beweis rechnerisch gesehen gültig, so wird festgestellt, ob dieser Beweis noch aktuell genug ist, um ihn zu akzeptieren. Dies erfolgt durch Überprüfung des Felds *AttrCertValidityPeriod* innerhalb von *CVT-Infos*. Ist die aktuelle Uhrzeit (Systemuhr) innerhalb der hier angegebenen Periode, so wird der Beweis akzeptiert, was wiederum dem *Verifier Control* gemeldet wird.



m/n	*10*	*100*	*1000*
4	304.2	285.5	505.9
10	295.0	314.5	713.7
50	302.2	321.9	544.2

Abb. 7-21: Zeitaufwand zur Prüfung von Vollständigkeitsbeweisen

Abb. 7-21 zeigt Mittelwerte des Zeitaufwands gemessen in Millisekunden zur Prüfung der bereits in Kap 7.3.1. erwähnten Test-Beweise, welche jeweils auf *k=[0; n/2]* Zertifikate eines Benutzers zielten. Dabei steht *n={10, 100, 1000}* für die Gesamtanzahl der durch den jeweiligen Ausgangsbaum der Ordnung *m={4, 10, 50}* erfassten Zertifikate. Diese Messwerte deuten darauf hin, dass unter den für die durchgeführten Messungen gewählten Bedingungen die Ordnung *m=4* und somit eher tiefere Bäume von Vorteil waren.

Die Abhängigkeit der in diesem Abschnitt beschriebenen Verifier-Komponenten vom betrachteten Anwendungskontext liegt auf der Hand:

- Die Abläufe während der Ermittlung der Benutzerrechte wurden zur Bedienung des ed.tec-Systems aufgebaut. Dies erschwert den Einsatz des den Ablauf steuernden *Verifier Control* in einem anderen Anwendungskontext.

- Die Ermittlung der jeweilig benötigten Zertifikate durch den *PDB Control* ist ebenfalls den Zielen des ed.tec untergeordnet: Die implementierten Datenbankanfragen sind zur Anforderung von Zertifikatsarten geeignet, welche im ed.tec eingesetzt werden. (Gleiches gilt natürlich für die PDB-Komponente *Verification Service*, welche in der derzeitigen Ausbaustufe die aus ed.tec-Sicht passenden Methoden anbietet.)

- Die Implementierung des *Proof Validators* ist durch die im ed.tec-Kontext verwendeten Struktur der I-CVTs geprägt. Diese sind ja B^+-Bäume, welche aus Sicht der Beweisprüfung semantiktragende Suchschlüssel enthalten. Hinzu kommt, dass der *Proof Validator* genau die von der *PDB Client* angeforderten Vollständigkeitsbeweise, d.h. Beweise für die Intervallsuche über die jeweiligen Suchschlüssel überprüfen kann. Anders aufgebaute bzw. indizierte Suchbaumkonstruktionen und/oder von den derzeitigen abweichend strukturierte Beweise würden die faktische Neuimplementierung des *Proof Validators* (sowie des *Proof Generators* auf Seiten der PDB) erforderlich machen.

Die genannten Punkte deuten auf eine Schwäche der I-CVT-basierten Absicherung bzw. Aktualisierung von Zertifikaten hin: Scheinbar einfache Änderungen haben eine massive Auswirkung auf die Realisierung der betroffenen Systemteile. Dies konnte insbesondere bei der Entwicklung von PAMINA festgestellt werden.

8 Ausblick

In dieser Arbeit wurden Aspekte der zertifikatsbasierten Zugriffskontrolle diskutiert. Als Ausgangspunkt diente ein Prozessmodell, das den meisten heute bekannten Zugriffskontrollmechanismen und –produkten zugrunde liegt.
Die effektive, d.h. sichere, Zugriffskontrolle setzt unter anderem voraus, dass die dabei ausgewerteten Autorisierungsdaten möglichst aktuell sind und die Zugriffskontrolle in einer authentischen und unveränderten Form erreichen. In der Praxis werden hierfür immer häufiger digital signierte Attributszertifikate verwendet. Diese ermöglichen die eindeutige Zuordnung von Autorisierungsdaten zu einer vertrauenswürdigen Stelle, die diese ursprünglich ausgestellt und signiert hat. Mit Hilfe von Attributszertifikaten können zwischenzeitlich erfolgte Manipulationen an den Autorisierungsdaten zuverlässig entdeckt werden. Dadurch können die mit deren Transport und Zwischenspeicherung in nicht vertrauenswürdigen Instanzen verbundene Risiken gemindert werden. Ein in diesem Umfeld bedeutender Standard ist X.509 [ITU01], der Formatierungs- sowie Interpretationsvorschriften für die verschiedenen Zertifikatsfelder enthält.
Diese Art Absicherung der Autorisierungsdaten führt in einem System zwangsläufig zu erhöhten betrieblichen Kosten. Diese machen sich hauptsächlich als zusätzlicher Zeitaufwand sowohl bei der Pflege als auch während der Auswertung der Autorisierungsdaten bemerkbar.

Nach einer Analyse verschiedener Vorgehensweisen wurde in der Arbeit vorgeschlagen, Attributszertifikate mit einer kurzen Gültigkeitsperiode auszustellen und diese mit einer entsprechender Häufigkeit zu aktualisieren. Dies macht vor allem die in den meisten Umgebungen erwartete hohe Änderungsdynamik der Autorisierungsdaten erforderlich.
Eine effiziente Realisierungsmöglichkeit bieten hierfür Authentifizierungsbäume [Mer89] an, die in diesem Kontext als *Certification Verification Trees* (CVT) [GGM00] funktionieren können. Sie erlauben im Wesentlichen das gleichzeitige Signieren von praktisch beliebig vielen Attributszertifikaten in regelmäßigen (kurzen) Abständen. Eine Voraussetzung zu deren Nutzung ist die Aufstellung einer den CVT speichernden Instanz (Datenbank). Diese kann die aktuelle Gültigkeit eines gefragten evtl. ebenfalls in dieser Datenbank gespeicherten Zertifikats anhand des Baums bescheinigen.
Ein (mit Beispielen belegtes) Problem dabei ist, dass diese Datenbank ansonsten durch den Baum signierte, d.h. gültige, Zertifikate unentdeckt verschweigen kann. Dies kann unmittelbar zur Ablehnung ursprünglich erlaubter bzw. zur Genehmigung ursprünglich verbotener Zugriffe in einem System führen.
In der Arbeit wurden zur Behebung dieses Problems verschiedene Konzepte vorgestellt und bewertet. Die vorgeschlagenen Vorgehensweisen machen die durch die Datenbank ausgeführte Suche nach bestimmten Zertifikaten für die Zugriffskontrolle nachvollziehbar und somit überprüfbar. Als Ergebnis gelten die Antworten der jeweiligen Datenhaltungsinstanz in Abhängigkeit der Suchergebnisse als sichere Vollständigkeitsbeweise bzw. im Spezialfall Existenz- oder Nicht-Existenz-Beweise.

Die Generierung derartiger Nachweise setzt entweder das Einfügen zusätzlicher Daten in einen CVT oder den Einbezug einer sicheren externen Indexstruktur oder aber Änderungen an der ursprünglichen Baumstruktur voraus. Detaillierter wurde der letztere Weg behandelt, wobei in Anlehnung an [BLL02] eine Datenstruktur mit der Bezeichnung *Improved Certification Verification Tree* entstand. Dieser Baum kann verschiedene Ausprägungen bezüglich seiner internen Struktur haben, die ausgearbeiteten Algorithmen zur Erstellung und Prüfung von verschiedenen Beweisen sowie die behandelten Beispiele setzen (in der Datenbanktechnik heute etablierte) B^+-Bäume [BM70] voraus.
Eine nahe liegende Idee ist die Verwendung der I-CVT-Struktur zur Indizierung von Daten beliebiger Semantik. Eine für den Anfragenden nachvollziehbare Beantwortung von Datenbankanfragen kann in den verschiedensten Szenarien auch außerhalb der in der Arbeit betrachteten Zugriffskontrolle ausgenutzt werden: Man denke z.B. an ein Online-Shop zum Verkauf von Reiseangeboten, wobei ein die Suchkriterien festlegender Kunde das preiswerteste Angebot aus der ihm vorgelegten Angebotsliste in Anspruch nehmen möchte.
Eine performante Implementierung derartiger Datenbanken auf Basis heutiger Datenbankmanagementsysteme bedarf tief greifender Änderungen an deren Aufbau: Die heute meistens automatisiert aufgebauten und verwalteten Indexbäume zu den in einem Datenbestand abgelegten Daten müssen nun um sicherheitsrelevante Daten (Hashwerte und Signaturen), die sich zudem dynamisch ändern, ergänzt werden. Diese Daten kann eine Datenbankinstanz nur teilweise selbständig ermitteln. Außerdem kann anhand einer bestimmten Indexstruktur (I-CVT) nur die „passende“ Anfrageart zufrieden stellend beantwortet werden, nämlich in Abhängigkeit von den im Baum enthaltenen und in die Berechnungen einfließenden Suchschlüsseln.
Der Gegenstand künftiger Untersuchungen sollte die konzeptionelle Klärung dieser Fragen sowie deren Implementierung möglicherweise durch die Erweiterung eines bestehenden (Open-Source) Datenbankmanagementsystems sein.

Im Prototypen mit der Bezeichnung PAMINA wurde der wesentlich bequemere Weg zur Realisierung der I-CVT-gestützten Zugriffskontrolle begangen: Hier werden sowohl die Zertifikate verschiedener Zertifizierungsstellen als auch die diese sichernden I-CVT in einer relationalen Datenbank (*Privilege Database*) abgelegt bzw. gepflegt (*PAMINA Administration Server*). Dadurch kommen die suchtechnischen Vorteile der in Form von Relationstupeln abgelegten B-Bäume nicht zum Tragen. Sie ermöglichen lediglich eine Konfiguration der durchschnittlich erwarteten Länge der Vollständigkeitsbeweise.
Neben der Betrachtung der Möglichkeiten zur häufigen Aktualisierung und Überprüfung von Attributszertifikaten wurde einer deren bedeutendsten Anwendungsfälle, nämlich die Delegierung von Rechten, im Detail behandelt. Da Attributszertifikate die Überprüfung der Identität der diese signierenden Instanzen ermöglichen, kann ihre Entstehungsgeschichte über verschiedene Delegierungsgeber und –Nehmer hinweg sicher dokumentiert werden.
Hierfür sehen gängige Standards (darunter der betrachtete X.509-Standard) in die Zertifikate eingefügte Rückverweise vor. Diese referenzieren die Zertifikate, aus welchen die im jeweiligen Zertifikat enthaltenen Rechte stammen.
Die Verfolgung dieser Referenzen bis hin zu Zertifikaten, die durch den Besitzer der betroffenen Ressourcen ausgestellten wurden, erweist sich vor allem wegen der mehrfach auszuführenden Signaturprüfungen und der Suche nach den jeweilig erforderlichen Ursprungszertifikate als kostenintensiv.

Zur Behebung dieser Probleme wurden ebenfalls mehrere Vorgehensweisen vorgeschlagen und bewertet. Diese sehen teilweise eine Erweiterung der standardisierten Zertifikatsformate vor und sind deshalb nicht anstrebenswert.
Das Konzept der „Offline Delegierung“ ist hingegen als eine rein betriebliche Maßnahme zu betrachten. Dabei können bereits zertifizierte Benutzer an die jeweilige Zertifizierungsstelle (Ressourcenbesitzer), die ihnen diese Rechte ursprünglich bescheinigt hat, Anträge zwecks Delegierung einer Teilmenge der eigenen Berechtigungen stellen. Als Folge entstehen keine während der Zugriffskontrolle zu prüfenden oder eben zwischenzeitlich zu reduzierenden Delegierungsnetzwerke mehr.
Die durchschnittlich erwarteten Kostenvorteile dieser Technik während der Zertifikatsprüfung können anhand der in der Arbeit durchgeführten Analyse nachvollzogen werden. Fraglich bleibt jedoch, wie sich dieses Konzept in realen Betriebs- bzw. Geschäftsumgebungen etablieren lässt. Dies sollte in Zukunft im Rahmen von entsprechenden Experimenten bzw. Feldversuchen erforscht werden.

9 Literaturquellen

[AFS01] M. Mar Albà, J. Domingo-Ferrer, F. Sebé: *Asynchronous Large-Scale Certification Based on Certificate Verification Trees*, In Proc. IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues, Darmstadt, Mai 2001.

[AL99] C. Adams, S. Lloyds: *Understanding Public Key Infrastructures: Concepts, Standards, and Deployment Considerations*, Macmillan Publisher, 1999.

[ALO98] W. Aiello, S. Lodha, R. Ostrowsky: *Fast digital identity revocation*. Advances in Cryptology - CRYPTO '98, LNCS, Bd. 1462, S. 137-152. Springer, 1998

[Amo94] E. Amoroso: *Fundamentals of Computer Security Technology*, Prentice Hall, 1994.

[ASZ+01] C. Adams, P. Sylvester, M. Zolotarev, R. Zuccherato: *Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols,* IETF Request for Comments 3029, Februar 2001.

[Aur98a] T. Aura: *Fast access control decisions from delegation certificate databases*, In Proc. of 3rd Australasian Conference on Information Security and Privacy ACISP '98, LNCS Bd. 1438, S. 284-295, Springer, Juli 1998.

[Aur98b] T. Aura: *On the structure of delegation networks*, In Proc. of 11th IEEE Computer Security Foundations Workshop, Rockport, Massachusetts, June 1998, S. 14-26, IEEE Computer Society Press 1998.

[Aur99] Tuomas Aura: *Distributed Access-Rights Management with Delegation Certificates*. In Secure Internet Programming -- Security Issues for Distributed and Mobile Objects, LNCS, Bd. 1603, S. 211-235. Springer, 1999.

[AZ98] C. Adams, R. Zuccherato: *A General, Flexible Approach to Certificate Revocation*, White Paper, Entrust Technologies, 1998.

[BGG+99] A. Berger, A. Giessler, P. Glöckner, W.Schneider: *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV*, Interoperabilitätsspezifikation SigI, August 1999.

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard: *Web Services Architecture*, W3C Working Group Note, Februar 2004.

[BHS92] D. Bayer, S. Haber, W.S. Stornetta: *Improving the Efficiency and Reliability of Digital Time-Stamping*, Sequences ’91, Methods in Communnication, Security, and Computer Science, S. 329-334, Springer, 1992.

[BL73] D. E. Bell, L. J. LaPadula: *Secure Computer Systems: Mathematical Foundations and Model*. Technical Report M74-244, MITRE Corp., Mai 1973

[Bla99] B. Blakley: *Corba Security: An Introduction to Safe Computing with Objects*, Addison Wesley, 1999.

[BLL00] A. Buldas, P. Laud, H. Lipmaa: *Accountable Certificate Management using Undeniable Attestations*. Proc. of the 7th ACM Conference on Computer and Communication Security, S. 9-17, November 2000.

[BLL02] A. Buldas, P. Laud, H. Lipmaa: *Eliminating Counterevidence with Applications to Accountable Certificate Management*, Journal of Computer Security, 10(3), S. 273-296, 2002.

[BLS00] A. Buldas, H. Lipmaa, B. Schoenmakers: *Optimally Efficient Accountable Time-Stamping*. In Public Key Cryptography '2000, LNCS Bd. 1751, S. 293-305, Springer, Januar 2000.

[BM70] R. Bayer, E. M. McCreight: *Organization and Maintenance of Large Ordered Indexes*. In Proc. SIGFIDET Workshop, S. 107-141, 1970.

[BM72] R. Bayer, E. M. McCreight: *Organization of large ordered indexes*, Acta Inform., 1, S. 173–189, 1972.

[Boc03] H. Bock: *Sicherheitskonzepte für ein Internet-basiertes Wissenstransfersystem*, Diplomarbeit, Universität Karlsruhe, März 2003.

[BR93] M. Bellare, P. Rogaway: *Random Oracles are Practical: a Paradigm for Designing Efficient Protocols*, 1st Conference on Computer and Communication Security, ACM, S. 62-73, 1993.

[BS00] E. Barka, R. Sandhu: *Framework for Role-Based Delegation Models*, In. Proc. of 16th Annual Computer Security Applications Conference (ACSAC), S. 168-180, Dezember 2000.

[CFS+03] S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu: *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework,* IETF Request for Comments: 3647, November 2003.

[Cho96] S. Chokhani: *A Security Flaw in the X.509 Standard* , In 19th National Information Systems Security Conference Proceedings, S. 463-471, 1996.

[CO02] D. Chadwick, O. Otenko: *The PERMIS X.509 Role Based Privilege Management Infrastructure*, In Proc. 7th ACM Symposium On Access Control Models And Technologies, SACMAT'02, S. 135-140, Juni 2002.

[Coc01] C. Cocks: *An Identity Based Encryption Scheme Based on Quadratic Residues*, in Proc. Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, Dezember 17-19, 2001, LNCS 2260, S. 360-364, Springer Verlag, 2001.

[Com79] D. Comer: *The Ubiquitous B-Tree*, ACM Computing Surveys, 11(2), S. 121-128, Juni 1979.

[CS03] D. W. Chadwick, M. V. Sahalayev: *LDAP Schema for X.509 Attribute Certificates*, IETF Internet Draft, Februar 2003.

[DBP96] H. Dobbertin, A. Bosselaers, B. Preneel: *RIPEMD-160: A Strengthened Version of RIPEMD*, Fast Software Encryption, LNCS Bd. 1039, S. 71-82, Springer, 1996.

[DE02] S. Dohrmann, C. Ellison: *Public-key Support for Collaborative Groups*, In Proc. 1st Annual PKI Research Workshop, April 2002.

[Den82] D. Dening: *Cryptography and Data Security,* Addison Wesley, 1982.

[DG+01] P. Devanbu, M. Gertz, C. Martel, S. G. Stubblebine: *Authentic Third-party Data Publication*, Proc. IFIP TC11/ WG11.3 14th Annual Working Conference on Database Security: Data and Application Security, Development and Directions, S. 101-112, November 2001.

[DG97] C. I. Dalton, J. F. Griffin: *Applying Military Grade Security to the Internet*. Journal Computer Networks and ISDN Systems, Vol. 29 (15), S. 1799-1808, 1997.

[DH76] W. Diffie and M.E. Hellman: *New Directions in Cryptography*, IEEE Transactions on Information Theory, v. IT-22, n. 6, Nov, S. 644-654. 1976.

[Dob96] H. Dobbertin: *Cryptanalysis of MD4*, Fast Software Encryption, Third International Workshop, LNCS 1039, S. 53-69, Springer, 1996.

[DR00] J. Daemen, V. Rijmen: *The Block Cipher Rijndael,* In Smart Card Research and Applications, LNCS Bd. 1820, S. 288-296, Springer, 2000.

[Ebi01] P. Ebinger: *Konzeption und prototypische Implementierung einer Infrastruktur zum zertifikatsbasierten Privilegien-Management*, Diplomarbeit, Universität Karlsruhe, November 2001.

[EFL+99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen: *SPKI Certificate Theory*, IETF Request for Comments 2693, September 1999.

[ElG85] T. ElGamal: *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Transactions on Information Theory, IT-31 S. 469-472, 1985.

[Ell99] C. Ellison: *SPKI Requirements*, IETF Request for Comments 2692, September 1999.

[Feu04] D. Feuerhelm: *Metaoperationen und Metadaten im Internet-basierten Wissenstransfer*, Dissertation, Universität Karlsruhe, 2004.

[FH02] S. Farrell, R. Housley: *An Internet Attribute Certificate Profile for Authorization*, Request for Comments 3281, April 2002.

[FKK96] A. O. Freier, P. Karlton, and P. C. Kocher: *The SSL protocol version 3.0*. IETF Network Working Group, Internet draft, November 1996.

[FP03] C. Francis, D. Pinkas: *Attribute Certificate Policy Extension*, PKIX Working Group, Internet Draft, April 2003.

[GGM00] I. Gassko, P. Gemmell, P. MacKenzie: *Efficient and Fresh Certification*. Public Key Cryptography '2000, LNCS, Bd. 1751, S. 342-353, Springer, 2000.

[Goo97] G. Goos: *Vorlesungen über Informatik, Band 1, Grundlagen und funktionales Programmieren*, S. 313-318, Springer, 1997.

[GS02] U. Gallizzi, A. Seiler: Attributszertifikate, Projektarbeit, Zürcher Hochschule Winterthur, Juli 2002.

[HAN99] H. G. Hegering, S. Abeck, B. Neumair: *Integrated Management of Networked Systems, Concepts, Architectures, and Their Operational Application*, Morgan Kaufmann, 1999.

[HFP+99] R. Housley, W. Ford, W. Polk, D. Solo: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, IETF Request for Comments 2459, Januar 1999.

[HM01] M. Hartmann, S. Maseberg: *Fail-Safe-Konzept für FlexiPKI,* In Kommunikationssicherheit im Zeichen des Internet, S. 128-138, Vieweg, 2001.

[HM01] J.Y. Halpern, R. van der Meyden: *A logical reconstruction of SPKI*, In Proc. 14th IEEE Computer Security Foundations Workshop, S. 59-70, 2001.

[HMM+00] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, Y. Ravid: *Access control meets public key infrastructure, or: Assigning roles to strangers*. In Proc. 2000 IEEE Symposium on Security and Privacy, S. 2–14., Mai 2000.

[HP+02] R. Housley, W. Polk, W. Ford, D. Solo: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, IETF Request for Comments 3280, April 2002.

[HP01] V. Hammer, H. Petersen: *Aspekte der Cross-Zertifizierung,* in Tagungsband des 7. Deutschen IT-Sicherheitskongreß des BSI, SecuMedia Verlag, Ingelheim, S. 281-296, 2001.

[HRU76] Michael A. Harrison , Walter L. Ruzzo , Jeffrey D. Ullman: *Protection in operating systems*, Communications of the ACM, v.19 n.8, S. 461-471, August 1976.

[Inc02] D. Ince: *Developing Distributed and E-commerce Applications*, Addison-Wesley, 2002.

[ISO89] International Organization for Standardization: *Information Technology - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture*, ISO/IEC 7498-2, 1989.

[ITU01] ITU-T Recommendation X.509. *Information Technology - Open Systems Interconnection - The directory: Public-Key and Attribute Certificate Frameworks*, ISO/IEC 9594-8. Februar 2001.

[ITU02] ITU-T Recommendation, X.680-X.683: *Abstract Syntax Notation One (ASN.1) X.690-X.693: ASN.1 encoding rules*, 2002.

[ITU93] ITU-T Recommendation *X.500: Overview of Concepts, Models, and Services,* ISO/IEC 9594-1:1993, 1993.

[ITU95] ITU-T Recommendation X.812: *Information Technology – Open Systems Interconnection – Security Frameworks for Open Systems: Access Control Framework*, ISO/EC 10181-3, 1995.

[JH94] K.M. Jackson und J. Hruska (Hrsg.): *Computer Security Reference Book*, Butterworth-Heinemann Ltd, Oxford, 1994.

[Kal92] B. Kaliski: *The MD2 Message-Digest Algorithm*, Network Working Group, Request for Comments 1319, April 1992.

[KHS00] Y. Kortesniemi, T. Hasu, J. Särs: *A revocation, validation and authentication protocol for SPKI based delegation systems*, In Proc. Network and Distributed System Security Symposium (NDSS), 2000.

[Knu98] D. E. Knuth: *The Art Of Computer Programming*, Bd. 3, Sorting and Searching. Zweite Ausgabe. Addison-Wesley. 1998.

[Koc98] P. Kocher: *On certificate revocation and validation*, in Proc. Financial Cryptography - FC'98, LNCS, Bd. 1465, S. 172-177., Springer, 1998.

[Koh78] L. Kohnfelder. *Towards a practical public-key cryptosystem*. Bachelor Thesis. MIT. 1978.

[Kor02] Y. Kortesniemi: *SPKI Performance and Certificate Chain Reduction*, Beitrag zum Workshop „*Credential basierte Zugriffskontrolle*“ in Tagungsband „*Informatik bewegt*“ zur 32. Jahrestagung der GI in Dortmund, Oktober 2002.

[KPS95] C. Kaufman, R. Perlman, M. Spencer: *Network Security – PRIVATE Communication in a PUBLIC World,* Prentice Hall, 1995.

[Küh03] K. Kühn: *Weiterentwicklung eines Systems für die Verwaltung von Attributszertifikaten,* Studienarbeit, Universität Kalrsruhe, Juni 2003.

[Lai92] X. Lai: *On the design and security of block ciphers*, In ETH Series in Information Processing, Vol. 1, Hartung-Gorre Verlag, 1992.

[Lam71] Butler W. Lampson: *Protection*, Proc. 5th Princeton Conf. on Information Sciences and Systems, S. 437-446, Princeton, 1971.

[Lee88] T. M. P. Lee: *Using Mandatory Integrity to Enforce "Commercial" Security*, Proc. of the 1988 IEEE Symposium on Security and Privacy, S. 140-146, April 1988.

[Lip82] S. B. Lipner: *Non-Discretionary Controls for Commercial Applications*. Proc. IEEE Symposium on Security and Privacy, S. 2-10, April 1982.

[LN99] J. Linn, M. Nyström: *Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments*, Proc. of the 4th ACM Workshop on RBAC, S. 121-130, 1999.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams: „*X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*“ IETF Request for Comments 2560, Juni 1999.

[McL94] J. McLean: *Security Models*, Encyclopedia of Software Engineering, John Wiley and Sons, 1994.

[Mer78] R. C. Merkle: *Secure communication over insecure channels*, Communications of the ACM 21, 4 S. 294-299, April 1978.

[Mer90] R. C. Merkle: *A Certified Digital Signature*. Advances in Cryptology: CRYPTO '89, LNCS, Bd. 0435, S. 218-238. Springer. 1990.

[MGP+01] I. Mavridis, C. Georgiadis, G. Pangalos, M. Khair: *Access Control based on Attribute Certificates for Medical Intranet Applications*, Journal of Medical Internet Research; 3(1):e9 März, 2001.

[MH78] R. Merkle, M. Hellman: *Hiding information and signatures in trapdoor knapsacks*, IEEE Transactions on Information Theory, IT-24 S. 525-530, September 1978.

[MHF03] A. Malpani, R. Housley, T. Freeman: *Simple Certificate Validation Protocol (SCVP)*" IETF Internet Draft, Juni 2003.

[Mic96] S. Micali: *Efficient Certificate Revocation*. Tech. Memo MIT/LCS/TM-542b. 1996.

[Mic97] S. Micali: *Efficient Certificate Revocation*, In Proc. RSA Data Security Conference, San Francisco, Januar, 1997.

[MMP02] M. Myers, A. Malpani, D.Pinkas: *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol, version 2,* IETF Internet Draft, Dezember 2002.

[MNM+00] C. Mayerl, Z. Nochta, M. Müller, M. Schauer, A. Uremovic, S. Abeck: *Specification of a Service Management Architecture to Run Distributed and Networked Systems*, In Trends Towards a Universal Service Market (USM'2000), München, 2000.

[MOV96] Alfred J. Menezes, Paul van Oorschott und Scott Vanstone: *Handbook of Applied Cryptography*. CRC Press, Oktober 1996.

[MR00] P. McDaniel, A. Rubin: *A Response to `Can We Eliminate Certificate Revocation Lists?'*. In Proc. Financial Cryptography 2000. International Financial Cryptography Association (IFCA), Februar 2000.

[Myr00] P. H. Myrvang: *An Infrastructure for Authentication, Authorization and Delegation*, Cand. Scient. Thesis in Computer Science, Univ. Tromso, 2000.

[NA+01a] Z. Nochta, G. Augustin, M. Becker, M. Friedmann, S. Abeck: *Sicherheitskonzept für eine durch Kunden steuerbare Dienstmanagement-Architektur*. Kommunikation in Verteilten Systemen (KIVS), Informatik Aktuell, S. 103-113, Springer, Februar 2001.

[NA+01b] Z. Nochta, G. Augustin, M. Becker, D. Feuerhelm, M. Friedmann: *Integration von Public-Key-Infrastrukturen in CORBA-Systeme*, Kommunikationssicherheit im Zeichen des Internet. Grundlagen, Strategien, Realisierung und Anwendungen, Vieweg, März 2001.

[NA02a] Z. Nochta, S. Abeck: *Ein vertrauenswürdiger webbasierter Zugriffskontrolldienst*, Von e-Learning bis e-Payment: Das Internet als sicherer Marktplatz, Tagungsband der Leipziger Informatik Tage (LIT'2002), S. 201-208, Leipzig, September 2002.

[NA02b] Z. Nochta, S. Abeck: *Sichere und effiziente Zugriffskontrolle mit PAMINA*, Beitrag zum Workshop „Credential basierte Zugriffskontrolle" in Tagungsband „*Informatik bewegt*" zur 32. Jahrestagung der GI in Dortmund, Oktober 2002.

[NEA02] Z. Nochta, P. Ebinger und S. Abeck: *PAMINA: a Certificate Based Privilege Management System*, in Proc. Network & Distributed System Security Symposium, S. 167-179, San Diego, Februar 2002.

[Nes00] I. Nestlerode: *Implementing EFECT*, Master Thesis, Massachusetts Institute of Technology, 2000.

[Nik99] P. Nikander: *An Architecture for Authorization and Delegation in Distributed Object-Oriented Agent Systems*. Doctoral dissertation, Helsinki University of Technology, März 1999.

[NIST01] U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2: *Secure Hash Standard* (Draft), 2001.

[NIST94] U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology: *Public-Key Infrastructure Study*, Gaithersburg, USA, 1994.

[NIST95] U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-1: *Secure Hash Standard*, April 1995.

[NN00] M. Naor and K. Nissim: *Certificate revocation and certificate update,* IEEE Journal on Selected Areas in Communications, 18(4), S. 561-570, April 2000.

[NOA03] Z. Nochta, J. Oetting, S. Abeck: *Management von X.509-Attributszertifikaten für die Zugriffskontrolle in verteilten Systemen*, In Praxisberichte zur 13.ITG/GI Fachtagung Kommunikation in verteilten Informationssystemen (KIVS), S. 143-155, VDE Verlag, Februar 2003.

[Oet02] J. Oetting: *Sicherheitskonzepte für Web-Services*, Diplomarbeit, Universität Karlsruhe, Juli 2002.

[OMG02] Object Management Group: *Security Services Specification* v. 1.8, März 2002.

[PBD97] B. Preneel, A. Bosselaers, and H. Dobbertin: *The cryptographic hash function RIPEMD-160*, CryptoBytes, Vol. 3, No. 2, S. 9-14, 1997.

[PH02] D. Pinkas, R. Housley: *Delegated Path Validation and Delegated Path Discovery Protocol Requirements*, IETF Request for Comments 3379, September 2002.

[PHB02] W. Polk, R. Housley, L. Bassham: *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, IETF Request for Comments 3279, April 2002.

[Res00] E. Rescorla: *HTTP Over TLS*, IETF Network Working Group, Request for Comments 2818, Mai 2000.

[Riv92] R. Rivest *The MD5 Message-Digest Algorithm*, Network Working Group Request for Comments 1321, April 1992.

[Riv98] R. Rivest: *Can We Eliminate Certificate Revocation Lists?*. In Proc. Financial Cryptography, LNCS, Bd. 1465, S. 178-193. Springer, 1998.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, v. 21, n. 2, S. 120-126, Februar 1978.

[Sam02] P Samarati: *Enriching Access Control to Support Credential-Based Specifications*, Beitrag zum Workshop „*Credential basierte Zugriffskontrolle*“ in Tagungsband „*Informatik bewegt*“ zur 32. Jahrestagung der GI in Dortmund, S. 114-119, Oktober 2002.

[Sär00] Jonna Särs: *Analysis and Application of Accountable Certificate Management*, Seminar on Network Security. Mobile Security ’2000, Helsinki University of Technology, Dezember 2000, (siehe www.nixu.fi/~jonna/paper/netsec00_certmgmt.html)

[SC+96] R. Sandhu, E. Coyne, H. Feinstein, C. Youman: *Role-Based Access Control*, in Proceedings of 10th Annual Computer Security Applications Conference, 1996.

[Sch00] B. Schneier: *Secrets & Lies: Digital Security in a Networked World*, John Wiley & Sons; 1st edition, 2000.

[Sch96] B. Schneier: *Applied Cryptography: Protocols, Algorithms and Source Code in C*, 2nd edition. John Wiley & Sons, 1996.

[Sha79] A. Shamir: *How to Share a Secret,* Communications of the ACM, 22(11), S. 612- 613, November 1979.

[SS94] R. S. Sandhu, P. Samarati: *Access Control: Principles and Practice*, IEEE Communications, Vol. 32(9), S. 40-48, September 1994.

[Sti02] D. Stinson: *Cryptography Theory and Practice*, Second Edition, CRC Press März, 2002.

[Sum97] R. C. Summers: *Secure Computing: Threats and Safeguards*, McGraw-Hill, 1997.

[SWS+01] G. Stoker, B. S. White, E. Stackpole, T.J. Highley, M. Humphrey: *Toward Realizable Restricted Delegation in Computational Grids*, In Proc. International Conference on High Performance Computing and Networking Europe, S. 32-41, 2001.

[Tan96] A. S. Tanenbaum. *Computer Networks*. Dritte Ausgabe. Prentice-Hall, Upper Saddle River, New Jersey. 1996.

[Vie03] A. Viehl: *Realisierung eines Systems für die effiziente und flexible Überprüfung von Attributszertifikaten*, Studienarbeit, Universität Karlsruhe, Mai 2003.

[WLM00] R. N. Wright, P. Lincoln, J. K. Millen: *Efficient fault-tolerant certificate revocation*, In Proc. ACM Conference on Computer and Communications Security, S. 19-24, 2000.

[Woh01] P. Wohlmacher: *Gültigkeitsprüfung von Zertifikaten*, In Kommunikationssicherheit im Zeichen des Internet. Grundlagen, Strategien, Realisierung und Anwendungen, Vieweg, März 2001.