Verteiltes Datenverwaltungssystem für Authentifizierungs- und Autorisierungsinfrastrukturen: *P2P-ZuSI*

-

Formales Modell und Sicherheitsanalyse

Dissertation zur Erlangung des Grades
eines Doktors der Wirtschaftswissenschaft

Eingereicht an der
Wirtschaftswissenschaftlichen Fakultät
der Universität Regensburg

Vorgelegt von Wiebke Dresp

Berichterstatter:

Prof. Dr. Peter Lory
Prof. Dr. Günther Pernul

Tag der Disputation: 31.01.2008

Danksagung

Vielen Dank an meinen Doktorvater Prof. Dr. Peter Lory, der mich über die gesamte Arbeit hinweg motiviert und unterstützt hat. Die zahlreichen Gespräche und Diskussionen haben zur Lösung vieler Probleme beigetragen.
Ein herzliches Dankeschön auch an meinen Zweitbetreuer Prof. Dr. Günther Pernul für die interessanten Denkanstöße, die mich angeregt haben, über den Tellerrand zu schauen.

Weiterhin gilt mein Dank der Studienstiftung des Deutschen Volkes für die ideelle und finanzielle Förderung meiner Arbeit.

Dr. Thomas Wölfl danke ich für die vielen Fachgespräche, Stefan Dürbeck und Christian Franzke für ihren konzentrierten Einsatz bei der Abschlusskorrektur und meinem Bruder Henning Dresp für die Unterstützung bei der graphischen Aufarbeitung. Ihr habt was gut bei mir!

Besonderen Dank an meinen Mann Alexander Koenen-Dresp für die nicht nur moralische, sondern auch praktische Unterstützung bei der Promotion, die endlosen Korrekturzyklen und die ausführlichen fachlichen Diskussionen. Ich weiß all das sehr zu schätzen.

Danke auch an meine Eltern - für die nötige Rückendeckung und den Glauben an meinen Erfolg.

All we have to decide is what to do with the time that is given to us.
(Gandalf the Grey, J.R.R. Tolkien)

Zusammenfassung

Das netzwerkbasierte Angebot von Diensten in der IT verlagert sich vom bisher dominierenden Client-Server-Ansatz immer stärker in Richtung verteilter Systeme, welche durch Flexibilität, Skalierbarkeit und nicht zuletzt geringe Kosten eine attraktive Alternative zu wartungsintensiven und teuren Serverlösungen darstellen. Dieser Trend geht mit der organisatorischen Dezentralisierung und Flexibilisierung von Verwaltungsstrukturen in Unternehmen und Behörden einher.
In diesem Umfeld der sich auflösenden klaren Organisationsstrukturen kommt der Authentifizierung und Autorisierung von Entitäten in dezentralisierten Systemen steigende Bedeutung zu.
Authentifizierungs- und Autorisierungsinfrastrukturen (AAI) bilden in Form von Regeln, Protokollen, Daten und Software den Rahmen für die Realisierung dieser beiden Dienste. Während die Ausstellung von Berechtigungen und Identitätsnachweisen ebenso wie deren Prüfung bereits heute in einigen AAI in dezentralisierter Form möglich ist, ist für die Speicherung und Verwaltung von Daten ausnahmslos ein zentrales Verzeichnis notwendig - ein Faktum, welches dem Paradigma der Dezentralisierung widerspricht und das Angebot von Authentifizierung und Autorisierung in durchgängig dezentralisierten Umgebungen verhindert.

Die vorliegende Dissertation setzt daher an diesem Punkt an und konzipiert ein verteiltes Peer-to-Peer-Verzeichnis, in welchem Daten zertifikatbasierter AAI veröffentlicht, verwaltet und angefragt werden können. Hierfür wird zunächst das vom Standpunkt der Sicherheit her optimale strukturierte Peer-to-Peer-Regelwerk - Kademlia - ausgewählt und die Datenverfügbarkeit in einem darauf basierenden Verzeichnis stochastisch formalisiert und umfassend analysiert.
Weiterhin wird eine geeignete Lösung für die Rückrufproblematik in zertifikatbasierten AAI gewählt, die mit dem Faktum der verteilten Datenverwaltung auf nicht notwendigerweise vertrauenswürdigen Knoten korrespondiert. Im Rahmen der Definition des Regelwerks für das Datenverwaltungssystem für Zertifikate und Statusinformationen, *P2P-ZuSI*, wird ein Schichtenmodell präsentiert, welches von diesem Regelwerk und dem darunterliegenden Peer-to-Peer-Netzwerk für die AAI-Anwendungsschicht abstrahiert und somit die flexible, transparente Umsetzung verteilter Datenverwaltung in beliebigen konkreten AAI-Systemen ermöglicht.
Eine AAI mit einem verteilten Verzeichnis unter Verwendung von *P2P-ZuSI* wird rollenbasiert in Form eines hierarchischen gefärbten Petrinetzes modelliert. Die Sicherheitseigenschaften von *P2P-ZuSI* hinsichtlich der neben der Verfügbarkeit relevanten Sicherheitsziele Vertraulichkeit und Integrität werden anhand dieses Modells nachgewiesen.
Ein Ausblick auf mögliche Einsatzszenarien im Internet ebenso wie in geschlossenen Netzwerken von Unternehmen, Universitäten oder Behörden zeigt die Praxisrelevanz der Arbeit.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

In den letzten Jahren zeichnet sich in der Informationstechnik ein Trend zu mehr Dezentralisierung in netzwerkbasierten Anwendungen ab. Traditionelle zentralisierte Server-Systeme, welche in der Vergangenheit hauptsächlich eingesetzt wurden, werden in vielen Anwendungsbereichen von verteilten Systemen, in denen Dienste von mehreren Instanzen gemeinschaftlich angeboten werden, abgelöst. Das Spektrum reicht von verteilten Serverlösungen bis hin zu selbstorganisierenden Peer-to-Peer-Netzwerken.
Insbesondere Peer-to-Peer-basierte Anwendungen im Internet profitieren immer stärker von der aktiven Einbindung der freien Ressourcen leistungsstarker Endbenutzer-Systeme. Dadurch entstehen gegenüber Client-Server-Systemen deutliche Vorteile hinsichtlich der Betriebskosten, Ausfallsicherheit und Skalierbarkeit für verteilt realisierbare Dienste wie Dateisysteme, Distributed Web Caches oder Dokumentmanagement.
Zudem ist organisatorische Dezentralisierung im Geschäfts- und Privatbereich auf dem Vormarsch. Sowohl in Unternehmen als auch im wissenschaftlichen Umfeld werden zentralistisch angelegte Strukturen wie beispielsweise feste Fakultäten, Institute oder Abteilungen zunehmend durch flexible Organisationsformen wie abteilungsübergreifende Arbeitsgruppen oder Kollaborationen zwischen Universitäten, Forschungseinrichtungen und Unternehmen ergänzt oder ersetzt.

Authentifizierung und Autorisierung (A&A) sind für dezentralisierte Systeme und Strukturen kritische und unverzichtbare Infrastrukturdienste. Unter Authentifizierung versteht man die Verifikation der von einer Entität vorgegebenen Identität. Bei der Autorisierung werden die Berechtigungen einer Entität zur Durchführung einer bestimmten Aktion hergeleitet oder geprüft.
Beispiele für die Notwendigkeit von A&A sind die Zugangskontrolle zu kostenpflichtigen oder anderweitig beschränkten Diensten oder die Vergabe von Lese- und Schreibberechtigungen auf vertrauliche Daten.

Authentifizierungs- und Autorisierungsinfrastrukturen (AAI) stellen in Form von Regeln, Protokollen, Daten und Software den Rahmen bereit, in welchem A&A realisiert werden. Die Konzeptionierung und Weiterentwicklung von Authentifizierungs- und Auto-

risierungsinfrastrukturen ist ein wichtiger Bestandteil aktueller Forschung und Entwicklung.
Es existieren bereits AAI-Systeme, welche die *dezentrale* Ausübung von Authentifizierung und Autorisierung ermöglichen. Beispielsweise bieten in SDSI/SPKI [ACM02] und AKENTI [AKENT] dezentralisierte Entitäten die Dienste an. Sie operieren dabei alle auf einer einheitlichen Datenbasis, welche aber bis heute zentrale Server voraussetzt und damit dem Paradigma der Dezentralisierung zuwiderläuft.
Dieser Paradigmenbruch erschwert den konistenten Ausbau von Authentifizierungs- und Autorisierungsdiensten für dezentralisierte Umgebungen und erzeugt hohen Aufwand für den Betrieb weiterhin notwendiger zentraler Server.

Diese Dissertation soll die Anwendung von AAI in vollständig dezentralisierten Systemen ermöglichen, um durch die entstehende Selbstorganisation, organisatorische Unabhängigkeit und hardwaretechnische sowie finanzielle Effizienz einen starken Anreiz zu bieten, dieses Konzept praktisch einzusetzen. Sie leistet damit einen Beitrag zur Auflösung des Paradigmenbruches zwischen zentralisierter Datenverwaltung und dezentralisiertem Dienstangebot in Authentifizierungs- und Autorisierungsinfrastrukturen.

1.2 Ziele der Arbeit

Obwohl AAI bereits seit einigen Jahren intensiv erforscht und auf dem wissenschaftlichen Gebiet weiterentwickelt werden, sind ihre Einsatzbereiche bisher stark beschränkt. Es existieren nur wenige Softwaresysteme, die überhaupt AAI-Funktionen bereitstellen. Insbesondere auf dem Feld der zertifikatbasierten AAI sowie deren Bausteinen, den Public-Key-Infrastrukturen (Authentifizierung) und Privilege-Management-Infrastrukturen (Autorisierung) liegen - sogar für den in der Wissenschaft dominierenden X.509-Standard - kaum Lösungen vor.

Die vorliegende Dissertation präsentiert eine umfassende Systematik für die Speicherung, Verwaltung und Bereitstellung von Daten einer zertifikatbasierten AAI in einer dezentralisierten Umgebung. Dieses verteilte Verzeichnis wird auf Basis eines strukturierten Peer-to-Peer-Netzwerks umgesetzt.
Es wird dazu ein Regelwerk konzipiert, welches Speicher- und Anfragemethoden für das Verzeichnis festlegt. Diese nutzen Funktionalitäten eines existierenden Peer-to-Peer-Systems und stellen eine standardisierte Schnittstelle für die handelnden Entitäten der AAI, also z.B. für Zertifikataussteller, bereit. Die Kombination aus diesem Regelwerk und dem konkreten Peer-to-Peer-System bildet gemeinsam das verteilte Verzeichnis und wird als *P2P-ZuSI* (Peer-to-Peer Zertifikat- und Status-Informationssystem) bezeichnet.

Ein weiteres Ziel ist die formale Modellierung von *P2P-ZuSI* in Kombination mit einer abstrakten AAI, welche auf diesem verteilten Verzeichnis operiert. Das Modell bildet das konzipierte System in eindeutiger, visueller und simulierbarer Form ab.

Durch eine quantitative stochastische Analyse des gewählten Peer-to-Peer-Systems und die qualitative Untersuchung des formalen Modells werden die Stärken und Schwächen

hinsichtlich Angriffen durch einzelne oder kollaborierende Teilnehmer belegt und somit der Grad der Systemsicherheit determiniert.

1.3 Aufbau

Das Forschungsthema ist am Schnittpunkt der zwei großen Forschungsbereiche Peer-to-Peer (P2P) und AAI angesiedelt. Daher werden zunächst diese beiden Themenkomplexe eingeführt (Kapitel 2 und 3).
Um ein verteiltes Verzeichnis konzipieren zu können, ist zunächst die Auswahl eines konkreten strukturierten P2P-Regelwerks notwendig. Dieses bestimmt die Grundparameter des Systems wie z.B. die Effizienz der Such- und Speichermethoden für Datensätze, deren Replikation und somit auch die zu erwartende Datenverfügbarkeit.
Die präsentierten P2P-Regelwerke werden anhand von Kriterien, die insbesondere aus den Sicherheitsanforderungen verteilter Datenverwaltungssysteme hergeleitet werden, qualitativ verglichen und anhand dieser Gegenüberstellung eine begründete Entscheidung getroffen. Anschließend folgt die quantitative Untersuchung der Datenverfügbarkeit in einem auf die gewählte Technik aufbauenden Datenverwaltungssystem unter Verwendung stochastischer Methoden (Kapitel 4).
Auf dieser Grundlage wird das detaillierte Regelwerk für *P2P-ZuSI* hergeleitet (Kapitel 5) und das Gesamtsystem in Form eines gefärbten hierarchischen Petrinetzes formal modelliert (Kapitel 6).
Danach erfolgt die Vervollständigung der bereits mit der Verfügbarkeitsanalyse eingeleiteten Sicherheitsevaluation in Form der qualitativen Untersuchung der Sicherheitsziele Vertraulichkeit und Integrität. Zugeschnitten auf verschiedene Angriffsszenarien wird das Petrinetzmodell um zusätzliche Elemente oder Initialbelegungen erweitert und dann mittels Zustandsraumanalyse auf Schwachstellen untersucht (Kapitel 7).
Die Analyse ermöglicht die nachfolgende Herleitung von praktischen Einsatzszenarien für *P2P-ZuSI* (Kapitel 8). Fazit und Ausblick auf weitere Forschungs- und Entwicklungsthemen (Kapitel 9) schließen die Arbeit ab.

Kapitel 2

Peer-to-Peer: Ein neues Paradigma

Ein P2P-Netzwerk ist ein flexibler Zusammenschluss von gleichberechtigten Entitäten, welche gemeinsam einen bestimmten Dienst anbieten. In diesem Sinne dominierten in den Anfängen des Internet Peer-to-Peer-basierte Strukturen wie beispielsweise DNS und das Usenet bzw. FidoNet (vgl. [Ora01]). Der eigentliche Siegeszug von Peer-to-Peer begann allerdings erst 1999 mit der lauffeuerartigen Verbreitung des durch zentrale Indexserver unterstützten Peer-to-Peer-Systems Napster [NAPST]. Im Jahr 2000 folgte unmittelbar Gnutella [GNUTE], das ganz ohne privilegierte Entitäten auskommt.
Seither sind Peer-to-Peer-Netzwerke Gegenstand intensiver wissenschaftlicher Forschung und Weiterentwicklung. Einen wesentlichen Meilenstein stellt dabei die Entwicklung strukturierter Peer-to-Peer-Netzwerke dar, in denen Metriken und Regeln die Zuordnung von Datensätzen zu Teilnehmern und dadurch das effizientere Auffinden von Daten ermöglichen. Diese werden inzwischen auch vermehrt praktisch angewandt, beispielsweise in verteilten Dateisystemen und Filesharing-Diensten.

Kapitel 2 bietet eine Einführung zum Thema Peer-to-Peer. Zunächst werden die zugrunde liegenden Konzepte vorgestellt. Ein Vergleich von unstrukturierten und strukturierten Peer-to-Peer-Techniken wird hinsichtlich ihrer Eignung als Basis für ein Datenverwaltungssystem für AAI-Daten durchgeführt und eine Empfehlung für ein strukturiertes System für diesen Anwendungsfall ausgesprochen.

2.1 Grundlagen

P2P-Systeme sind eine spezielle Form verteilter Systeme gemäß folgender Definition:

Definition 2.1 ***Verteiltes System:***
Ein verteiltes System ist ein Verbund von autonomen Entitäten, welche über ein Netzwerk kommunizieren. Das Ziel eines verteilten Systems ist die Bündelung von Ressourcen der partizipierenden Entitäten zur Generierung eines kohärenten Gesamtsystems.

In einem Peer-to-Peer-System werden diese autonomen Entitäten auch als Knoten oder Peers (Gleichgestellte) bezeichnet. Jeder Knoten kann sowohl als Nutzer (Client) als auch

als Anbieter (Server, Provider) des Dienstes auftreten. Es existiert also keine A-Priori-Klassifizierung in Anbieter und Konsumenten.

Definition 2.2 ***Knoten, Peers:***
Knoten sind Entitäten des Peer-to-Peer-Netzwerks. Sie sind untereinander durch logische Verbindungen verknüpft und können Daten anfragen, Nachrichten empfangen, beantworten und weiterleiten sowie Datensätze speichern.
Auf technischer Ebene ist ein Peer ein Computersystem, welches mit anderen Systemen über ein Netzwerk verbunden ist. Es steht unter der Kontrolle eines Knotenbetreibers, z.B. eines Computerbenutzers.

P2P-Systeme zeichnen sich durch ihre Anpassungsfähigkeit und vollständige Dezentralisierung aus: sie operieren gänzlich ohne privilegierte Entitäten. Kommunikationsverbindungen werden ohne zentralisierte Koordination als logische Punkt-zu-Punkt-Verbindungen[1] zwischen den Knoten etabliert. Das Netzwerk, in welchem die Entitäten kollaborieren, ist in der Regel transient, d.h. die Teilnehmer wechseln häufig und in kurzen Zeitabständen.
Da ungenutzte Ressourcen der Peers eingesetzt werden, entstehen beinahe keine zusätzlichen Fixkosten und nur geringer variabler Aufwand für jeden Teilnehmer. Es werden von vornherein nur geringe Anforderungen an die Beschaffenheit und Ausstattung der Peers gestellt. Existierende P2P-Netzwerke setzen sich in der Regel zum großen Teil aus privaten Computern, meist PCs, zusammen.
In Client-Server-Systemen hingegen - sowohl in solchen, in denen der Dienst vollständig zentralisiert angeboten wird als auch in solchen, in denen mehrere Server den Dienst als verteiltes System realisieren - werden hohe Anforderungen an die Server gestellt, beispielsweise hinsichtlich Hardware, Software, Verfügbarkeit und Wartung.

Definition 2.3 ***Peer-to-Peer-System (angelehnt an [AnS04]):***
Ein Peer-to-Peer-System ist ein verteiltes System von untereinander verbundenen Knoten, die in der Lage sind sich selbst zum Zweck der gemeinsamen Nutzung von Ressourcen, d.h. Daten, CPU-Leistung, Speicherplatz und Bandbreite in Netzwerktopologien zu organisieren. Es ist in der Lage mit auftretenden Fehlern und einem transienten Netzwerkzustand umzugehen, während die Performanz und Stabilität des Knotenverbundes erhalten bleibt.
Ein Peer-to-Peer-System ist selbstorganisierend und benötigt keine Vermittlungsdienste vom einem globalen zentralen Server oder einer anderen privilegierten Entität.

Derzeit wird diese Netzstruktur vor allem für die Realisierung von Filesharing- und Publikationsdiensten (z.B. Gnutella) eingesetzt, ist aber auch Gegenstand der Forschung als alternative Technologie für verteilte Dateisysteme wie PAST [RoD01b] oder das steganographische Dateisystem Mnemosyne [HaR02], Distributed Web Caches wie Squirrel [IRD02], Instant-Messaging-Anwendungen wie ICQ [ICQ] oder Systeme zum verteilten Rechnen (Distributed Computing) wie Seti@Home [Ora01].
Im Folgenden liegt der Betrachtungsschwerpunkt auf Peer-to-Peer-Systemen zur Verwaltung, Speicherung und Bereitstellung von Datensätzen.

[1] Auf der Ebene der Netzwerkkommunikation, z.B. über TCP/IP im Internet, werden diese Verbindungen in der Regel nicht unmittelbar zwischen den Maschinen aufgebaut.

2.2 Klassifizierung von Peer-to-Peer-Systemen

In unstrukturierten P2P-Systemen wie z.B. Gnutella [GNUTE] verbleiben alle Daten zunächst auf dem publizierenden Knoten; es existieren keine Regeln, welche die Verteilung und Positionierung der Daten innerhalb des Netzwerks festlegen. Dadurch entsteht kein Verwaltungsaufwand, wenn Knoten hinzukommen (Join) oder ausfallen (Leave), d.h. das Netzwerk ist höchst effizient skalierbar. Verlässt ein Knoten das Netz, gehen die bei ihm gespeicherten Daten allerdings für andere Nutzer verloren.
Beliebte Datensätze werden in der Regel auf den Knoten, welche sie anfragen, repliziert (Pufferung, Caching[2]). Somit stehen sie auch nach Ausfall des Knotens, welcher sie initial publiziert hat, weiter zur Verfügung. Zudem verringert das Caching den Aufwand für die Suche nach Datensätzen und verteilt die Last der Bereitstellung auf multiple Knoten.
Durch das Fehlen eines Regelwerks für die Zuordnung von Datensätzen zu Peers kann eine gesuchte Information theoretisch auf jedem Knoten des Netzes - oder aber auf gar keinem - existieren. A priori verfügt also ein Teilnehmer, der einen bestimmten Datensatz anfragt, über keinerlei Informationen hinsichtlich dessen Speicherknoten.
Die Suche ist daher ein ungerichteter Prozess, bei dem jeder Knoten, der eine Suchanfrage erhält, diese - wenn er sie nicht selbst beantworten kann - an alle ihm bekannten Peers weiterleitet (siehe Abbildung 2.1). Dieser Vorgang wird als Flooding bezeichnet, da das Netzwerk mit Anfragen „geflutet“ wird.

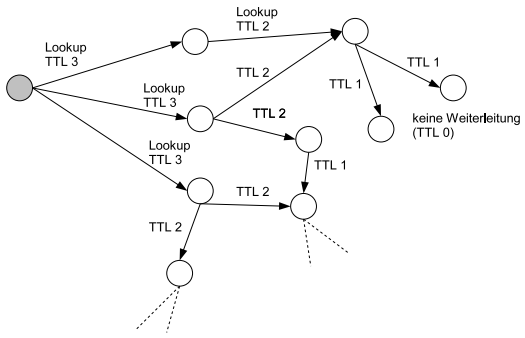


Abbildung 2.1: Weiterleitung von Suchanfragen in einem unstrukturierten Netzwerk für TTL $t = 3$

Die Anzahl und Beschaffenheit der Kontakte[3] unterliegt ebenfalls keinen Regeln. So kann es leicht passieren, dass dadurch ganze Teilnetze von einer Suchanfrage gar nicht erreicht werden, dafür aber andere Knoten die gleiche Anfrage mehrfach erhalten.
Die maximale Lebensdauer, d.h. die maximale Anzahl t an Weiterleitungsschritten für eine Suchanfrage (Time To Live, TTL), muss zudem eingeschränkt werden. Ansonsten

[2] In Version 2 von Gnutella wird das Caching so weit vorangetrieben, dass nur festgelegte „Super Peers“ die gepufferten Daten bereitstellen und den einzigen Zugang zu ihren Subnetzen repräsentieren. Dies widerspricht allerdings der Forderung der Gleichberechtigung zwischen Knoten und der Dezentralisierung, ist aber vor allem ein Risiko, da ein Super Peer die Verbreitung von Datensätzen, die aus seinem Subnetz kommen, unterbinden kann.

[3] Gnutella: jeder Knoten verfügt über ungefähr 5 Kontakte

würde nicht nur die Netzlast durch das Flooding zu hoch, sondern es würden auch unzumutbare Rücklaufzeiten auftreten. Im schlimmsten Fall würde sogar die Suche gar nicht terminieren.
Die Suche bricht nun also ab, sobald die Anfrage t-mal weitergeleitet wurde[4]. Die Suche bleibt damit stets lückenhaft, insbesondere in Netzwerken mit vielen Teilnehmern.
Für zuverlässige Datenspeicherung ist es aber unbedingt notwendig, dass *alle* im Netz vorhandenen Daten schnell und effizient gefunden werden können. Außerdem muss auch die *Nicht*existenz von gesuchten Datensätzen zweifelsfrei feststellbar sein.
Dies leisten nur strukturierte Peer-to-Peer-Systeme wie z.B. Chord [SML01], Pastry [RoD01a] oder Kademlia [MaM02]. Sie verfügen über ein Regelwerk, nach dem jeder Datensatz einem oder mehreren Peers zugeordnet wird.
Dafür wird jedem Datensatz zu Beginn seiner Existenz ein Bezeichner zugewiesen, welcher in einem systemweit festgelegten Bezeichnerbereich (Adressraum) liegt. Dieser Adressraum ist unter den Peers aufgeteilt, so dass jeder Knoten für ein bestimmtes Segment zuständig ist. Dieses ist in der Regel variabel, d.h. es ändert sich bei Reorganisation des Netzwerks.
Ein Datensatz mit Bezeichner (FileID) b verbleibt nicht auf dem ihn publizierenden Knoten. Stattdessen wird unter Zuhilfenahme eines rekursiven Suchalgorithmus die Menge $\mathcal{N}_b$ der zuständigen Knoten ermittelt ($|\mathcal{N}_b| \geq 1$) und der Datensatz dorthin übertragen.
Strukturierte Peer-to-Peer-Systeme werden auch als DHTs (Distributed Hash Table, verteilte Hashtabelle) bezeichnet: Jedes Bezeichner-Datensatz-Paar ist Element einer über alle Knoten verteilt gespeicherten Hashtabelle, ebenso jede Zuordnung von Peer-Bezeichner zur Netzwerkadresse (IP-Adresse) eines Knotens. Jeder Peer speichert in einer sogenannten Routing-Tabelle Verweise auf einen Teil der existierenden Knoten, um Nachrichten weiterleiten zu können.
Diese regelbasierte Umschichtung von Datensätzen hat den Vorteil, dass jeder Knoten des Netzwerks mit Hilfe des gleichen Zuordnungsalgorithmus sowohl ermitteln kann, wo ein Datensatz gespeichert werden muss, als auch, welche Knoten derzeit für einen gesuchten Datensatz zuständig sind, und eine Anfrage nach dessen Bereitstellung direkt an diese stellen kann.

Definition 2.4 ***Overlay-Netzwerk und Overlay-Topologie:***
Ein Overlay-Netzwerk bzw. Overlay im Sinne des Paradigmas strukturierter Peer-to-Peer-Systeme[5] *ist ein logisches Netzwerk, welches auf ein anderes aufsetzt. Es implementiert ein Regelwerk und eine Struktur für die Anordnung und Interaktion von Peers. Die Methoden, Strukturen und Protokolle des darunter liegenden Netzwerks werden für die Kommunikation zwischen Knoten genutzt. Die Routenwahl im Overlay hängt nicht von den Routingregeln des zugrunde liegenden Netzwerks ab.*
Die Topologie des Overlay-Netzwerks bezeichnet die Struktur, in welcher die Entitäten logisch untereinander verbunden sind.

Teil des Regelwerks sind bespielsweise die Kriterien für die bei jedem Knoten vorgehaltenen Informationen über andere Peers und die Protokolle für die Kontaktaufnahme

[4] Gnutella: $t = 7$, d.h. maximal $\sum_{i=1}^{7}(5^i) = 97655$ Knoten könnten theoretisch erreicht werden, wenn kein Knoten mehrfach kontaktiert würde; dieses Maximum ist allerdings wegen des fehlenden Regelwerks für die Weiterleitung von Suchanfragen praktisch nicht erreichbar.

[5] Es existieren auch Nicht-P2P-Overlays, beispielsweise VPNs (Virtual Private Networks).

zwischen Knoten. Die darunter liegende Transportschicht implementiert in der Regel TCP oder UDP. Damit wird die konkrete Kommunikation zwischen Teilnehmern des Overlay-Netzwerks realisiert.

Verschiedene Overlay-Topologien sind in Abbildung 2.2 dargestellt: die Ringtopologie wie in Chord, die baumförmige Topologie wie in Kademlia und die multidimensionale Torustopologie wie in CAN (Content-Addressable Network, [RFH01]).

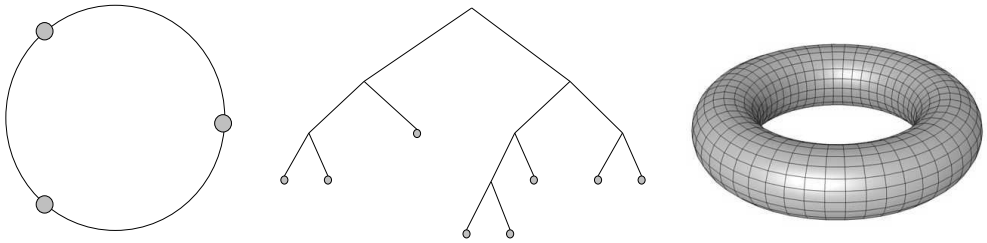


Abbildung 2.2: Overlay-Topologien: Ring, Binärbaum, dreidimensionaler Torus

Zur Realisierung von Kommunikationsverbindungen zwischen Knoten, insbesondere mit dem Ziel der Weiterleitung (Routing) von Nachrichten, hält jeder Knoten Informationen über andere Teilnehmer im Netzwerk und deren Zuständigkeitsbereiche in einer Tabelle vor. Jede deckt dabei nur einen kleinen Ausschnitt des gesamten Netzwerks ab: der Netzzustand ist an keiner Stelle vollständig erfasst, sondern findet sich verteilt in der Gesamtheit der Routing-Tabellen aller Knoten.
Es existieren systemweite Regeln, welche die in einer Routing-Tabelle zu speichernden Referenzen auf andere Knoten determinieren und damit sicherstellen, dass eine Nachricht durch schrittweise Weiterleitung an jeden beliebigen Knoten auch außerhalb der Menge der eigenen Kontakte übermittelt werden kann.
Die Zuteilung des Zuständigkeitsbereichs für jeden Knoten wird in der Regel dynamisch realisiert um die Fluktuation durch Joins und Leaves auszugleichen.
Replikation, also die kontrollierte Erzeugung verschiedener identischer Exemplare eines Datensatzes auf verschiedenen Knoten ist ein wichtiges Mittel zur Verfügbarkeitssicherung bei unangekündigten Leaves oder in Protokollen, die beim Leave keinen Transfer der gespeicherten Daten an neue zuständige Knoten unterstützen. Weiterhin wirken Replikationstechniken auch gegen böswillig, d.h. bewusst nicht protokollkonform handelnde Teilnehmer.

Jeder Knoten kann - wie bereits erwähnt - eine Suche nach Bezeichnern durchführen. Ein Bezeichner b kann einen gesuchten Datensatz spezifizieren (FileID) oder aber einen Knoten (b heißt dann NodeID). Für die Suche wird eine Anfrage nach dem bzw. den zuständigen Knoten für einen solchen Bezeichner über ein rekursives Routing-Verfahren in Richtung $\mathcal{N}_b$ weitergeleitet. In jedem Schritt nähert sich der Lookup dabei den zuständigen Knoten.

Bezieht sich die Suche auf einen Datensatz, fordert der Initiator der Anfrage diesen von einem Knoten aus $\mathcal{N}_b$ an. Stellt sich nach sukzessiver Kontaktierung aller Knoten aus $\mathcal{N}_b$ heraus, dass keiner davon den entsprechenden Datensatz gespeichert hat, so wird die Suche erfolglos beendet. Für den Initiator ist dann sicher, dass das gesuchte Datum nicht im Netz existiert.

Zur Realisierung eines Dienstes zur effizienten und zuverlässigen Datenspeicherung und -bereitstellung kommt daher *nur* ein *strukturiertes* Peer-to-Peer-System in Frage, da nur ein solches die lückenlose Suche in der Datenbasis und damit die entsprechende Sicherheit hinsichtlich der Richtigkeit und Vollständigkeit der Suchergebnisse leisten kann.

Die Begriffe P2P-Protokoll, -Regelwerk, -System und -Netzwerk werden in der Literatur häufig synonym verwendet. Im Folgenden wird aber eine deutliche Abgrenzung zwischen diesen Bezeichnungen vorgenommen.

Definition 2.5 ***Strukturiertes Peer-to-Peer-Protokoll:***
Ein strukturiertes Peer-to-Peer-Protokoll regelt auf Basis einer Distanzmetrik die Zuordnung von Bezeichnern zu NodeIDs und darauf aufbauend die Suche nach Bezeichnern sowie den Join und Leave von Knoten.

Definition 2.6 ***Strukturiertes Peer-to-Peer-Regelwerk:***
Ein strukturiertes Peer-to-Peer-Regelwerk ist die Kombination eines P2P-Protokolls mit zusätzlichen Methoden und Anforderungen. Das Regelwerk umfasst beispielsweise Methoden für die Speicherung und Replikation von Datensätzen, wobei für Suche und Zuordnung das zugrunde liegende Protokoll verwendet wird.

Definition 2.7 ***Strukturiertes Peer-to-Peer-System:***
Ein strukturiertes Peer-to-Peer-System (auch als strukturiertes P2P-Netzwerk bezeichnet) ist ein real existierendes Netzwerk von multiplen Knoten, welches auf einem P2P-Regelwerk basiert. Die teilnehmenden Peers handeln dabei im Idealfall[6] *regelkonform.*

Da von nun an nur noch von *strukturierten* Peer-to-Peer-Protokollen, -Regelwerken usw. die Rede ist, kann der Zusatz „strukturiert“ im Folgenden weggelassen werden.

2.3 Eigenschaften strukturierter Peer-to-Peer-Systeme

Strukturierte Peer-to-Peer-Systeme zeichnen sich durch eine Vielzahl positiver Eigenschaften aus, die sie gegenüber den bislang vorherrschenden Client-Server-Systemen und gegenüber unstrukturierten Systemen abheben. Einige der im Folgenden genannten positiven Eigenschaften gelten jedoch für unstrukturierte und strukturierte Systeme gleichermaßen und sind entsprechend kenntlich gemacht.

- *Stabilität und Ausfallsicherheit:*
  Der durch ein strukturiertes Peer-to-Peer-System angebotene Dienst ist auch unter

[6] Da es sich um ein reales System mit nicht vollständig kontrollierbaren Benutzern handelt, ist absichtliches oder unabsichtliches Fehlverhalten von Teilnehmern bis hin zu böswilligen Angriffen nicht auszuschließen.

negativen Rahmenbedingungen, z.B. bei Angriffen und Ausfällen von Teilnehmern, mit hoher Wahrscheinlichkeit verfügbar. Er wird von jedem einzelnen Knoten angeboten, so dass auch umfangreiche Ausfälle oder böswilliges Verhalten von Knoten nicht den gesamten Dienst deaktivieren können. Zudem existieren in der Regel Stabilisierungs- und Datenerhaltungsmechanismen wie beispielsweise die Replikation von Datensätzen.
Für unstrukturierte P2P-Systeme gilt ähnliches, wobei hier per definitionem weniger solche Regeln und Mechanismen existieren.
Im Gegensatz dazu ist in einem Client-Server-System der Server der kritische Punkt; fällt er aus, ist der Dienst unverfügbar. Abhilfe schafft nur das Anlegen einer redundanten Serverstruktur (z.B. Clustering), was mit finanziellem und organisatorischem Aufwand verbunden ist.

- *Performanz:*
  Nachrichten werden effizient zwischen den Knoten ausgetauscht, wobei die Länge der Routing-Pfade in der Regel logarithmisch skaliert. Damit arbeiten Suche und Speicherung von Datensätzen sehr performant im Vergleich zu unstrukturierten Netzen.
  Client-Server-Systeme hingegen sind durch ihren zentralisierten Ansatz und die Leistungsparameter der Server effizienter in Datenanfrage und -verwaltung.

- *Skalierbarkeit:*
  Ein Peer-to-Peer-System kann sich dynamisch an wechselnde Teilnehmerkreise und variierende Netzgrößen anpassen. Die Verkleinerung oder Vergrößerung läuft selbstorganisiert ohne großen Anpassungsaufwand ab.
  Durch diese gute Skalierbarkeit kann prinzipiell jede interessierte Entität am System partizipieren, da keine Maximalgröße für das Netzwerk existiert. Dies ist ein wesentlicher Vorteil gegenüber Client-Server-Systemen, die auf eine Vergrößerung der Nutzergruppe mit aufwendiger Leistungsaufstockung reagieren müssen und auf Verkleinerung gar nicht reagieren können.

- *Plattformunabhängigkeit:*
  Voraussetzung für den Betrieb eines Knotens in einem strukturierten P2P-System ist neben einer Software, welche das verwendete Peer-to-Peer-Regelwerk implementiert, in vielen Fällen nur ein Computer mit Festplatte und einer Internetverbindung. Die Anforderungen bezüglich Speicherplatz, Rechenkapazität und Bandbreite sind i.A. gering, so dass ein handelsüblicher PC mit Modem- oder ISDN-Internetanbindung bereits als Knoten eingesetzt werden kann. Auch sind bezüglich der sonstigen Softwareumgebung und des Betriebssystems in der Regel keine festen Vorgaben zu erfüllen.Gleiches gilt für Knoten eines unstrukturierten Netzwerks.
  Ein Server muss hingegen wesentlich höhere Anforderungen bezüglich der Ausfallsicherheit der Hardware, der Qualität der Netzwerkanbindung und der vorhandenen Software inkl. Betriebssystem erfüllen.

- *Dezentrale, verteilte Organisation und organisatorische Unabhängigkeit:*
  Für die Dienstbereitstellung ist keine zentrale Instanz notwendig, welche den Betrieb koordiniert, verwaltet und die finanzielle, rechtliche und organisatorische Ver-

antwortung übernimmt. Stattdessen tragen alle Teilnehmer eines P2P-Systems gleichermaßen zur Funktionsfähigkeit des Dienstes bei und teilen sich zugleich die Gesamtverantwortung. Damit sind die Hürden für die Inbetriebnahme gering, während die dezentrale Systemverwaltung der bereits in Abschnitt 1.1 beschriebenen Tendenz zur organisatorischen Dezentralisierung im privaten, geschäftlichen und wissenschaftlichen Umfeld Rechnung trägt.
Ein Peer-to-Peer-Netzwerk kann außerdem - im Gegensatz zu einem zentralen Server - nicht einfach abgeschaltet werden, sondern unabhängig von der Interessenslage externer Instanzen autonom existieren.

- *Benutzerakzeptanz:*
  Peer-to-Peer-Systeme haben in den letzten Jahren eine große Verbreitung erfahren, insbesondere zum Austausch von Musik- und Filmdateien (z.B. Gnutella). Auch ist in diesem Anwendungsfeld ein Trend zur Nutzung strukturierter P2P-Regelwerke festzustellen, beispielsweise in der Kademlia-Erweiterung für eMule [EMKAD] oder der BitTorrent-Kademlia-Erweiterung eXeem [EXEEM].
  Diese Erfahrungen hinsichtlich der Akzeptanz von Peer-to-Peer-Systemen lassen die Unterstützung ergänzender und alternativer Anwendungen auf P2P-Basis erwarten.

- *Geringe Kosten:*
  Pro Teilnehmer des Peer-to-Peer-Netzwerks entstehen nur sehr geringe Aufwendungen, da für den Betrieb des Knotens ungenutzte Ressourcen auf ohnehin betriebenen Rechnersystemen eingesetzt werden und Strom- sowie Verbindungskosten usw. daher vernachlässigbar sind. Dies ist ein wesentlicher Vorteil gegenüber Client-Server-Systemen, deren Wartung und Betrieb allgemein kostspielig ist.

Zudem haben strukturierte Peer-to-Peer-Systeme aber auch negative Eigenschaften, besonders bezüglich der Verfügbarkeit der einzelnen Knoten:

- *Ausfallwahrscheinlichkeit:*
  Das Online-Verhalten der Knoten ist erwartungsgemäß nicht stabil. Da gerade private Computer oft über keine Standleitung verfügen, ist mit ihrer regelmäßigen Trennung vom Netz zu rechnen. Weiterhin ist die Wahrscheinlichkeit für Software- und Hardwarefehler hoch. Auch das Nutzerverhalten hat großen Einfluss darauf, wie lange ein Knoten online bleibt.

- *Fehlverhaltenswahrscheinlichkeit:*
  Mit einer sehr hohen Wahrscheinlichkeit existieren im Netzwerk Teilnehmer, welche nicht regelkonform agieren, z.B. indem sie Nachrichten nicht korrekt oder gar nicht weiterleiten. Ein besonders häufiger Fall des Fehlverhaltens ist das sogenannte Free Riding: ein Knoten nutzt zwar den durch das Peer-to-Peer-Netzwerk angebotenen Dienst, beteiligt sich aber nicht selbst an dessen Bereitstellung, z.B. indem er sich weigert, Datensätze zu speichern.

- *Angriffswahrscheinlichkeit:*
  Es ist weiterhin wahrscheinlich, dass sich Teilnehmer des Netzwerks böswillig verhalten, d.h. den ordnungsgemäßen Betrieb des Dienstes stören oder verhindern wollen, beispielsweise mittels falscher Nachrichten, absichtlichen Löschens oder Unterdrückens von Datensätzen.

- *Geringe Integrität und Verfügbarkeit einzelner Datensätze:*
  Durch die genannten Wahrscheinlichkeiten für Ausfall, Fehlverhalten und Angriffe ist es möglich, dass einzelne Datensätze kompromittiert werden. Sie können dabei sowohl beschädigt als auch unverfügbar werden.
  Bei der Speicherung der Daten auf zentralen Servern hingegen tritt diese Problematik nicht oder nur in abgeschwächter Form auf, da hier der Zugriff auf die Datenbasis kontrolliert und nur durch wenige berechtigte Entitäten erfolgt.

- *Fehlende Eingriffsmöglichkeiten:*
  Die Dienstqualität hängt vom Verhalten der Teilnehmer ab. Zwar ist anzunehmen, dass sich diese zum Großteil regelkonform verhalten; ist das aber nicht der Fall, kann nicht regulierend eingegriffen werden, da die dezentrale Organisation privilegierte Entitäten und damit auch Eingriffe in die Aktivitäten einzelner Knoten generell verbietet. Weiterhin kann der Gesamtzustand des Netzwerks nicht zentral überwacht werden.

Das Peer-to-Peer-System muss also im Rahmen seiner selbstorganisierenden Struktur mit den genannten Problemen umgehen können und entsprechende Sicherheitsmechanismen (z.B. Replikationstechniken) implementieren.

2.4 Begriffsbestimmungen

Die folgenden Definitionen der zentralen Begrifflichkeiten bereiten die Einführung konkreter strukturierter P2P-Protokolle und -Regelwerke in Abschnitt 2.5 vor.

Jeder Knoten in einem strukturierten Peer-to-Peer-Netzwerk verfügt über einen systemweit einzigartigen Bezeichner, die NodeID:

Definition 2.8 ***NodeID:***
Die NodeID n_i eines Knotens i ist sein eindeutiger Bezeichner, ein Bitstring mit fester Länge m_n. Sie wird mittels einer systemweit festgelegten und jedem Teilnehmer des Peer-to-Peer-Netzwerks bekannten Funktion ν über knotenspezifische Daten d_i berechnet:

$$n_i = \nu(d_i),\ \nu : \{0,1\}^* \rightarrow \{0,1\}^{m_n}$$

Bemerkung: Im Folgenden wird die NodeID n_i auch zur Referenzierung des Knotens i verwendet; n_i wird also synonym zu i gebraucht.

Analog zur NodeID ist die FileID[7] als Bezeichner für Datensätze definiert.

Definition 2.9 ***FileID:***
Die FileID f_j eines Datensatzes j ist sein nicht notwendigerweise eindeutiger identifizierender Bezeichner, ein Bitstring mit fester Länge m_f. Sie wird mittels einer systemweit

[7] In der P2P-Literatur wird die FileID eines Datensatzes in der Regel als *Key* bezeichnet. Um Verwechslungen mit Public Keys und Private Keys auf der AAI-Ebene zu vermeiden, wird auf diese gängige Notation in dieser Arbeit absichtlich verzichtet.

festgelegten und jedem Teilnehmer des Peer-to-Peer-Netzwerks bekannten Funktion ϕ über spezifische Metadaten d_j des Datensatzes berechnet:

$$f_j = \phi(d_j),\ \phi : \{0,1\}^* \rightarrow \{0,1\}^{m_f}$$

Die FileID wird in der Regel aus allgemeinen Metadaten über den Datensatz berechnet; damit ergeben sich nicht-eindeutige FileIDs für Datensätze mit ähnlichem Inhalt und dadurch identischen allgemeinen Metadaten. Im weiteren Verlauf dieser Arbeit (Kapitel 5) werden auch für das AAI-Datenverwaltungssystem nicht-eindeutige FileIDs vorkommen.

Definition 2.10 ***Adressraum:***
Der mögliche Wertebereich für NodeIDs und FileIDs wird als Adressraum bezeichnet. Für NodeIDs umfasst der Adressraum alle möglichen Bitstrings der Länge m_n, für FileIDs alle möglichen Bitstrings der Länge m_f.

In den meisten Peer-to-Peer-Regelwerken gilt $m_n = m_f = m$.

Eine grundlegende Maßzahl ist die Distanz zwischen zwei Bezeichnern:

Definition 2.11 ***Distanz:***
Die Distanz zwischen zwei Bitfolgen x und y gibt gemäß einer festgelegten Relation Δ die Entfernung zwischen x und y an. Bei dieser Relation handelt es sich definitionsgemäß um eine Metrik, d.h. sie erfüllt die folgenden Axiome:

1. $\Delta(x,x) = 0$
2. $\Delta(x,y) > 0 \Leftrightarrow x \neq y$
3. $\Delta(x,y) = \Delta(y,x)$ *(Symmetrie)*
4. $\Delta(x,y) \leq \Delta(x,z) + \Delta(z,y)$ *(Dreiecksungleichung)*

In P2P-Umgebungen sind x und y NodeIDs oder FileIDs. Mit Hilfe dieser Distanzdefinition wird dann beispielsweise der FileID-Bereich definiert, für den ein Knoten zuständig ist, oder die Auswahl des nächsten Knotens, an den eine Suchanfrage weitergeleitet wird (siehe Definition 2.14 des Lookup).

Definition 2.12 ***Mapping und Zuständigkeit:***
Ein Bezeichner b (FileID oder NodeID) wird in Abhängigkeit von der Distanz zwischen b und den NodeIDs der Knoten im Netzwerk einer Knotenmenge $\mathcal{N}_b$ zugeordnet (Mapping). Damit sind die Knoten $\in \mathcal{N}_b$ für b zuständig:

$$\mathcal{N}_b = \nu_r(b),\ \nu_r : \{0,1\}^m \rightarrow \mathcal{P}(\{0,1\}^m)$$

Es gilt $|\mathcal{N}_b| \geq 1$. Für einen Knoten $n_{r_i} \in \mathcal{N}_b$ schreibt man auch $n_r\langle b \rangle$.

Als Basis für die Definition der Lookup-Prozedur werden nun die Grundbegriffe von Pfaden und deren Längen vorgestellt.

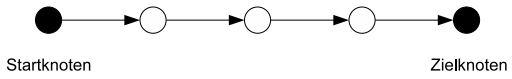


Abbildung 2.3: Pfad der Länge 5

Definition 2.13 ***Pfad und Pfadlänge:***
Ein Pfad ist eine Folge von Knoten. Der Anfangspunkt des Pfades wird durch den Startknoten, das Ende durch den Zielknoten determiniert. Die Knoten sind durch gerichtete Kanten in Richtung des Zielknotens verbunden. Die Länge eines Pfades ist die Anzahl der in ihm enthaltenen Knoten.

Eine Kernmethode von Peer-to-Peer-Regelwerken und Bestandteil jedes P2P-Protokolls ist der Lookup:

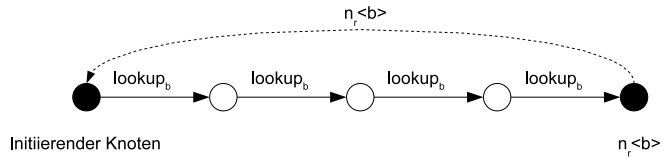
Definition 2.14 ***Lookup und Routing:***
Der Lookup bezeichnet die von einem Peer initiierte Suchanfrage nach zuständigen Knoten für einen Bezeichner, also für die FileID eines Datensatzes oder die NodeID eines anderen Knotens.
Als Lookup-Nachricht $lookup_b$ für Bezeichner b bezeichnet man die Bitfolge, welche die Suchanfrage repräsentiert. Sie wird gemäß systemweit definierter Regeln von Knoten zu Knoten weitergeleitet. Dabei nutzt jeder Peer die in seiner Routing-Tabelle gespeicherten Verweise auf andere Knoten, so dass sich der Lookup mit jedem Schritt den zuständigen Peers annähert, bis schließlich ein Knoten $n_r\langle b\rangle$ gefunden ist und an den Initiator der Anfrage gemeldet wurde. Der konkrete Weiterleitungsprozess wird als Routing bezeichnet.

Gegenüber der Suche mittels Flooding in unstrukturierten P2P-Systemen terminiert ein Lookup in einem strukturierten System immer, da er durch die definierten Regeln stets die zuständigen Knoten erreicht.

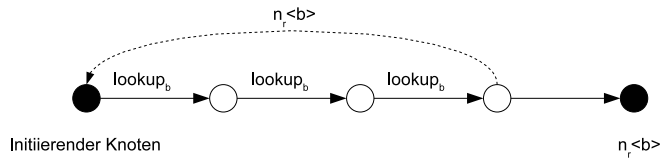
Definition 2.15 ***Routing-Pfad:***
Ein Routing-Pfad ist die Abfolge aller Knoten, welche an einem Lookup beteiligt sind.

Die Lookup-Nachricht für Bezeichner b wird vom initiierenden Knoten an den ersten Knoten des Pfades übergeben und in der Regel von dort aus weitergereicht.
Die Verbindung vom vorletzten Knoten zum letzten repräsentiert je nach konkretem Protokoll entweder den Transfer der Lookup-Nachricht, wobei der zuständige Knoten dann selbst eine Antwort an den initiierenden Peer generiert (siehe Abb. 2.4(a)), oder eine logische Verbindung, d.h. einen Zeiger auf $n_r\langle b\rangle$, der vom *vor*letzten Knoten gemeldet wird (siehe Abb. 2.4(b)). Ist im weiteren von Pfaden die Rede, so sind stets Routing-Pfade gemeint.

In den Beschreibungen aus Abschnitt 2.5 tauchen wiederholt Hashfunktionen auf; sie werden vor allem für die Berechnung von NodeIDs und FileIDs verwendet. Es handelt sich hierbei um konsistente Hashfunktionen [Sch95] wie beispielsweise SHA-1 [SHS01].



(a) Routing-Pfad mit Rückmeldung durch zuständigen Knoten



(b) Routing-Pfad mit Rückmeldung durch vorletzten Knoten

Abbildung 2.4: Routing-Pfad der Länge 5

Definition 2.16 ***Konsistente Hashfunktion:***
Eine Hashfunktion ist eine Funktion

$$H : \Sigma^* \rightarrow \Sigma^m,$$

welche Zeichenfolgen beliebiger Länge aus dem Alphabet Σ auf Zeichenfolgen der festen Länge m abbildet. In der Informationstechnik gilt allgemein $\Sigma = \{0, 1\}$. Der Funktionswert $y = H(x)$ wird auch als Hashwert von x bezeichnet.
$H^i(x)$, $i \in \mathbb{N}_0$ ist die i-fache Anwendung der Hashfunktion H auf Eingabewert x:

$$H^i(x) = \underbrace{H(H(\ldots H}_{i-mal}(x)\ldots))$$

Für den Sonderfall $i = 0$ gilt $H^0(x) = x$.

In strukturierten Peer-to-Peer-Netzwerken kommen ausschließlich konsistente Hashfunktionen [KLL97] zum Einsatz, welche sich insbesondere durch die folgenden Eigenschaften auszeichnen:

1. ***Einwegeigenschaft:*** *Der Hashwert eines Bitstrings x ist effizient berechenbar: $H(x) = y$. Es ist jedoch praktisch unmöglich, einen Eingabewert x' zu einem gegebenen y zu finden, so dass $y = H(x')$.*

2. ***Schwache Kollisionsresistenz (2nd-preimage resistance):*** *Es ist praktisch unmöglich, zu einem gegebenen Wert x ein $x' \neq x$ zu finden, so dass $H(x) = H(x')$.*

3. ***Gleichverteilung:*** *Für einen beliebigen Eingabewert x ist jeder Funktionswert y aus der Menge aller möglichen Hashwerte als Wert für $H(x)$ gleich wahrscheinlich.*

Die Gleichverteilung ist eine entscheidende Eigenschaft für den Einsatz von konsistenten Hashfunktionen zur Generierung von NodeIDs und FileIDs. So wird sichergestellt, dass sich diese Bezeichner gleichmäßig in ihrem Adressraum verteilen. So ist ein Knoten

jeweils für in etwa den gleichen Anteil der insgesamt im Netz vorhandenen Daten verantwortlich. Zudem ist so insbesondere bei Join und Leave von Knoten der Aufwand für die Umverteilung der FileIDs auf die zuständigen Knoten minimal.

2.5 Vorstellung strukturierter Peer-to-Peer-Protokolle und -Regelwerke

Im Folgenden wird eine Auswahl strukturierter P2P-Protokolle und -Regelwerke beschrieben. Chord, Pastry und Viceroy sind reine Protokolle. CAN, Kademlia, P-Grid und Symphony beinhalten zusätzliche Regeln für Datenspeicherung und Replikation und werden daher als (rudimentäre) P2P-Regelwerke eingeordnet.
Auf diese detaillierte Beschreibung wird in Kapitel 4 die Entscheidung für ein konkretes Protokoll bzw. Regelwerk aufgebaut.

2.5.1 CAN

Der Entwurf von CAN (Content-Addressable Network) begründete 2001 die Entwicklung strukturierter Peer-to-Peer-Overlays. Ein CAN [RFH01] ist ein Overlay in Form eines d-dimensionalen Torus. Der für NodeIDs und FileIDs zur Verfügung stehende Adressbereich ist ein virtueller d-dimensionaler kartesischer Koordinatenraum auf diesem Torus. Ein Knoten n_i ist jeweils für einen bestimmten Wertebereich im Umfeld seiner NodeID entlang aller Dimensionen zuständig.

Die Relation Δ zwischen zwei Bezeichnern $x = (x_1, x_2, \ldots, x_d)$ und $y = (y_1, y_2, \ldots, y_d)$, $x_i, y_i \in \mathbb{N}_0$ ist als die kartesische Distanz (Euklidische Distanz) definiert:

$$\Delta(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Lookup. Jeder Knoten speichert in seiner Routing-Tabelle Verweise (3-Tupel aus NodeID, verwaltetem Wertebereich und Netzwerkadresse) auf alle direkten Nachbarn, d.h. alle Knoten, deren Wertebereiche in $d-1$ Dimensionen mit seinem überlappen und in einer Dimension unmittelbar an seinen angrenzen.
Der Knoten, welcher den Lookup nach einem Bezeichner b initiiert, sucht aus seiner Routing-Tabelle den Peer mit derjenigen NodeID, welche die geringste Distanz zu b hat. An diesen übermittelt er eine Lookup-Nachricht. Der kontaktierte Knoten sucht dann wiederum in seiner Routing-Tabelle nach dem Peer mit nächster NodeID zu b, so dass schließlich nach mehreren Schritten der zuständige Knoten die Nachricht erhält. Abbildung 2.5 zeigt den Lookup für den Bezeichner $b = (6, 2)$ in einem Beispiel-CAN mit $d = 2$.

Zum Speichern eines Datensatzes wird zunächst die FileID $f_j = \tilde{\phi}(d_j) = (f_{j_1}, f_{j_2}, \ldots f_{j_d})$ unter Verwendung einer abgewandelten FileID-Berechnungsfunktion $\tilde{\phi} : \{0,1\}^* \rightarrow \mathbb{N}_0^d$ bestimmt. Dann wird ein Lookup nach dem Punkt f_j durchgeführt, welcher die NodeID des zuständigen Knotens $n_r\langle f_j \rangle$ liefert ($|\mathcal{N}_b| = 1$). An diesen übergibt der publizierende Knoten dann den Datensatz.

Die durchschnittliche Pfadlänge für einen Lookup beträgt $\frac{d}{4}N^{\frac{1}{d}}$, wobei N die Anzahl aller im Netzwerk vorhandenen Knoten ist. Durch Inkrementierung von d oder durch die Implementierung multipler Realitäten kann die durchschnittliche Länge der Pfade verkürzt werden.
Bei Einsatz multipler Realitäten werden mehrere unabhängige d-dimensionale Koordinatenräume erzeugt. Nun besitzt jeder Knoten multiple NodeIDs und damit auch multiple Listen von Nachbarn, jeder Datensatz aber nur eine einzige FileID für alle Koordinatenräume (Details hierzu siehe [RFH01]). Dadurch entstehen von jedem Datensatz Replika, da pro Realität ein Knoten für den Datensatz zuständig ist und diesen speichert (nun: $|\mathcal{N}_b| = k$ bei k Realitäten).

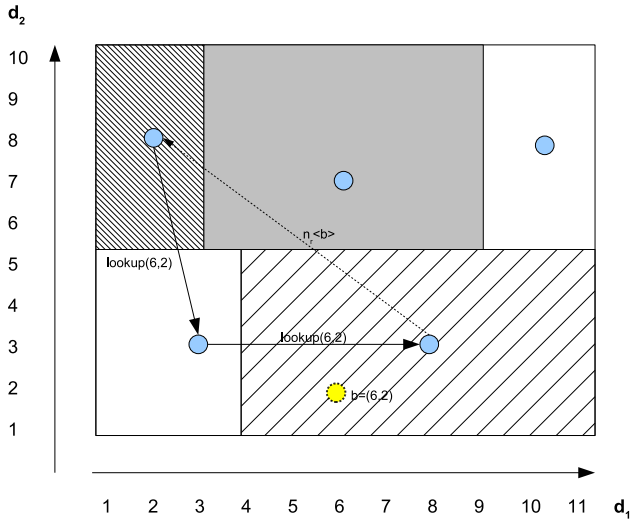


Abbildung 2.5: Beispiel: Lookup nach Bezeichner $b = (6, 2)$ in einem CAN mit $d = 2$

Join. Will ein Knoten i dem Netzwerk beitreten, so berechnet er zunächst aus seinen knotenspezifischen Daten d_i seine NodeID $n_i = \tilde{\nu}(d_i)$ mit einer ebenfalls abgewandelten Funktion $\tilde{\nu} : \{0,1\}^* \to \mathbb{N}_0^d$ und bezieht dann über einen initial bekannten Knoten eine Liste von derzeit aktiven CAN-Peers.
Der neue Knoten muss sich nun im CAN positionieren, d.h. den Wertebereich, für den er zuständig ist, ermitteln. Durch einen Lookup nach seinem Bezeichner n_i erhält er den bislang für den Wertebereich um n_i zuständigen Knoten $n_r\langle n_i\rangle$. Dieser halbiert nun seine Zone gemäß einer festgelegten Rangfolge der Dimensionen und weist dem Knoten n_i eine Hälfte zu. Weiterhin übergibt $n_r\langle n_i\rangle$ den Inhalt seiner Routing-Tabelle an den neuen Knoten als Initialbelegung und aktualisiert seine eigene Tabelle, indem er den neuen Nachbarn aufnimmt. Weiterhin werden alle Nachbarknoten über die Zonenteilung informiert.

Leave. Verlässt ein Knoten das Netzwerk, muss seine Zone von den verbleibenden Knoten übernommen werden. Normalerweise findet eine explizite Übergabe von Werte-

bereich und Datenbestand an einen der Nachbarn statt. Fällt ein Peer aus, gehen die gespeicherten Daten einfach verloren.

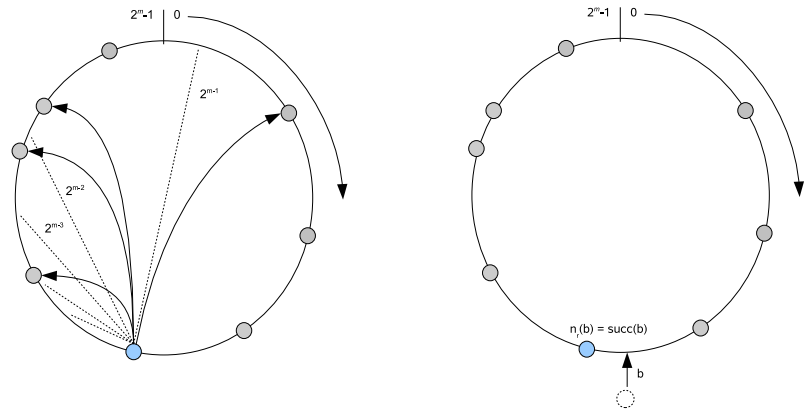
2.5.2 Chord

Alle Bezeichner im Protokoll Chord [SML01] sind Bitstrings der Länge $m = 160$ Bit und auf einem Bezeichner-Ring $modulo\, 2^m$ angeordnet (ringförmiges Overlay).
Zur Bestimmung der Distanz zwischen zwei Bezeichnern $x, y \in \{0,1\}^m$ werden die Bitstrings als Dualzahlen interpretiert. Die Distanz ergibt sich dann als betragsmäßige Differenz zwischen ihnen:

$$\Delta(x, y) = |x - y|.$$

Diese Subtraktion wird durch binäre Addition des Zweier-Komplements realisiert.

Lookup. Jeder Knoten speichert in seiner Routing-Tabelle Informationen (NodeID und Netzwerkadresse) über Peers, die auf dem Chord-Ring im Uhrzeigersinn nach ihm kommen. Durch die ringförmige Topologie haben die einzelnen Kontakte nicht notwendigerweise größere NodeIDs als der aktuelle Knoten, da bei Fortschreiten im Uhrzeigersinn der Übergang von Adresse $2^m - 1$ zu 0 überschritten werden kann (siehe Abb. 2.6(a)).
Lookups werden unidirektional im Uhrzeigersinn durchgeführt. Ein Knoten ist genau dann für einen Bezeichner b zuständig, wenn er der erste im Uhrzeigersinn auf b folgende Peer ist (siehe Abb. 2.6(b)). Der zuständige Knoten wird als direkter Nachfolger von b bezeichnet: $n_r\langle b\rangle = succ(b)$ mit $succ : \{0,1\}^m \to \{0,1\}^m$.



(a) Segmente und eingetragene Knoten in der Routing-Tabelle

(b) Nachfolger von Bezeichner b

Abbildung 2.6: Beispiel: Segmente und Nachfolger in Chord

Eine Routing-Tabelle eines Knotens mit NodeID n_i besitzt maximal m Einträge (Zeilen). Der j-te Eintrag in der Tabelle gibt NodeID n_s und Netzwerkadresse des ersten Knotens an, dessen NodeID zu n_i eine Distanz von mindestens 2^{j-1} hat, d.h.

$$n_s = succ(n_i + 2^{j-1}\ mod\ 2^m)$$

mit $1 \leq j \leq m$. Der Eintrag in der ersten Zeile bezeichnet somit stets den direkten Nachfolger von n_i. Mit diesem Aufbau der Routing-Tabelle wird aus der Sicht von n_i der Chord-Ring in Zuständigkeitssegmente seiner Kontakte aufgeteilt[8]. Die Segmente decken unterschiedlich große Teile des Adressraums ab.
Ein Lookup wird wie folgt ausgeführt: Der initiierende Knoten sucht aus seiner Routing-Tabelle den letzten Knoten mit einer NodeID $n_{next} < b$ und übermittelt an diesen eine Lookup-Nachricht $lookup_b$.
Der Empfänger dieser Nachricht verfährt ebenso und leitet $lookup_b$ weiter. Dieser Schritt wird wiederholt, bis ein Knoten n_p feststellt, dass der Kontakt in der ersten Zeile seiner Routing-Tabelle - sein unmittelbarer Nachfolger - eine NodeID $n_s \geq b$ besitzt. Dann ist $n_s = succ(b)$ der zuständige Knoten für b (auch hier gilt $|\mathcal{N}_b| = 1$). Seine NodeID und Netzwerkadresse werden von n_p an den Initiator des Lookup zurückgegeben.

Da Chord durch die Konstruktion der Routing-Tabelle Sprünge auch an weit entfernte Stellen im Ring erlaubt und sich die Distanz zwischen NodeID und FileID mit jedem Schritt mindestens halbiert, kommt ein Lookup mit einer durchschnittlichen Pfadlänge von $O(log_2 N)$ aus.
Zum Speichern eines Datensatzes j wird ein Lookup auf $f_j = H(d_j)$ ausgeführt und der Datensatz an $succ(f_j)$ übertragen.

Join. Um die Organisation von Join und Leave zu vereinfachen, besitzt jeder Knoten neben seiner Routing-Tabelle auch einen Zeiger auf seinen direkten Vorgänger.
Betritt ein neuer Knoten i das Netz, bestimmt er zunächst seine NodeID n_i. Dann beauftragt er einen beliebigen bestehenden Knoten n_e, einen Lookup nach $succ(n_i)$ auszuführen. Der zuständige Knoten $n_r\langle n_i\rangle$ setzt n_i als direkten Vorgänger (Predecessor). Weiterhin wird der bisherige Predecessor über die Existenz des neuen Knotens informiert, so dass er ihn als seinen neuen direkten Nachfolger registriert. Der neue Peer ist dann erfolgreich eingebunden. Initialwerte für die Belegung seiner Routing-Tabellen erhält er von seinem direkten Vorgänger sowie Nachfolger.
In einem Lookup ermittelt n_i nun noch die Knoten, in deren Routing-Tabellen er möglicherweise erfasst werden muss, und informiert diese über seine Anwesenheit. Der letzte Schritt ist die Übertragung entsprechender gespeicherter Datensätze von $n_r\langle n_i\rangle$ an den neuen Peer. Der Gesamtaufwand für den Join ist $O(log_2 N)$.

Leave. Verlässt ein Knoten das Netzwerk, so müssen alle Routing-Tabellen, in denen ein Verweis auf ihn enthalten ist, diesen Eintrag durch seinen bisherigen Nachfolger ersetzen. Der Aufwand ist wie schon beim Join $O(log_2 N)$, da die Verfahrensweise ähnlich ist. Jeder Knoten verfügt hierfür über eine Nachfolgerliste mit k Einträgen; fällt der erste Knoten aus dieser Liste aus, wird der zweite Eintrag als neuer Nachfolger verwendet.

[8] Ein einziger Knoten n_s kann in der Routing-Tabelle von n_i mehrere Zeilen belegen, wenn aufgrund geringer Netzgröße die Knoten weit auseinanderliegen und n_s damit direkter Nachfolger von 2^{j-1} für mehrere j ist. Abbildung 2.6(a) zeigt einen solchen Fall, in welchem der unmittelbare Nachfolger zwei Zeilen der Routing-Tabelle belegt.

2.5.3 Kademlia

Kademlia [MaM02] ist ein Peer-to-Peer-Regelwerk, welches eine Baumtopologie implementiert. Die teilnehmenden Knoten stellen die Blätter eines Binärbaums dar. Die Position eines Knotens wird durch das kürzeste eindeutige Präfix der NodeID festgelegt, welche wie bei Chord die Länge $m = 160$ Bit besitzt. Es ist zu erwarten, dass der entstehende Binärbaum durch die Gleichverteilungseigenschaft der Hashfunktion zur Bezeichnerberechnung annähernd ausbalanciert ist. Kademlia ist so konstruiert, dass Konfigurationsnachrichten, z.B. zur Aktualisierung von Routing-Tabellen, fast vollständig vermieden werden. Stattdessen werden Informationen über andere Knoten aus empfangenen Lookup-Nachrichten extrahiert.

Als Distanzmetrik wird die XOR-Metrik verwendet: Die Bitstrings x und y werden über das binäre XOR verknüpft (bitweise Addition ohne Übertrag). Damit ist die Distanz zwischen zwei Knoten gering, wenn ihre NodeIDs ein langes gemeinsames Präfix besitzen, d.h. ihr kleinster gemeinsamer Unterbaum durch einen möglichst weit von der Wurzel entfernten inneren Knoten begrenzt wird. Im Gegensatz zu den einfacheren Distanzdefinitionen in den bislang vorgestellten Protokollen ist diese Metrik nicht unmittelbar einsichtig, da die Distanz - als Dualzahl interpretiert - eine abstrakte, nicht intuitive Maßzahl darstellt. Abbildung 2.7 und Tabelle 2.1 verdeutlichen daher die Anwendung der XOR-Metrik zur Bestimmung der Distanz zwischen Knoten im Binärbaum.

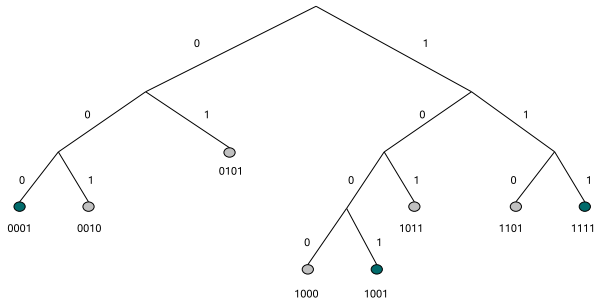


Abbildung 2.7: Beispiel: Binärbaum in Kademlia mit $m = 4$

$x \downarrow \quad y \rightarrow$	0001 (1)	0010 (2)	0101 (5)	1000 (8)	1001 (9)	1011 (11)	1101 (13)	1111 (15)
0001	0000 (0)	0011 (3)	0100 (4)	1001 (9)	1000 (8)	1010 (10)	1100 (12)	1110 (14)
1001 (9)	1000 (8)	1011 (11)	1100 (12)	0001 (1)	0000 (0)	0010 (2)	0100 (4)	0110 (6)
1111 (15)	1110 (14)	1101 (13)	1010 (10)	0111 (7)	0110 (6)	0100 (4)	0010 (2)	0000 (0)

Tabelle 2.1: Beispiel: Distanzen $\Delta(x, y)$ im Kademlia-Binärbaum (dezimale Entsprechung in Klammern)

Es sind stets diejenigen Knoten $n_{r_1}, n_{r_2}, ...n_{r_k} \in \mathcal{N}_b$ für einen Bezeichner b verantwortlich, für welche die jeweilige Distanz $\Delta(n_{r_i}, b)$ (als Dualzahl interpretiert) zu den k geringsten im Netz gehört, wobei k ein systemweit festgelegter Parameter ist (es gilt also: $|\mathcal{N}_b| = k$).

Im Beispiel-Baum wären also bei $k = 3$ für den Bezeichner $b = 1110$ die Knoten mit den IDs 1111,1101 und 1011 zuständig.

Lookup. Die Routing-Tabelle eines Knotens n_i umfasst $m = 160$ Zeilen. Jede dieser Zeilen enthält eine FIFO-Liste mit maximal k Einträgen. Jedes dieser sogenannten k-Buckets repräsentiert dabei einen Unterbaum, in welchem n_i nicht enthalten ist. Die in der j-ten Zeile der Routing-Tabelle erfassten Knoten teilen die ersten $j - 1$ Bits mit n_i, während das j-te Bit abweicht. Die Routing-Tabelle muss, wenn ein Unterbaum mindestens einen Knoten enthält, in der entsprechenden Zeile mindestens einen Kontakt beinhalten.
Empfehlung der Entwickler von Kademlia [MaM02] ist $k = 20$. Abbildung 2.8 zeigt am Beispiel, welche Unterbäume in welcher Zeile der Routing-Tabelle vom Knoten mit ID 1011 abgedeckt werden müssen.

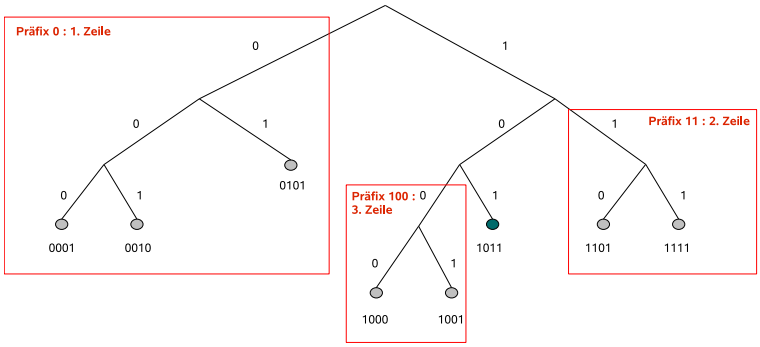
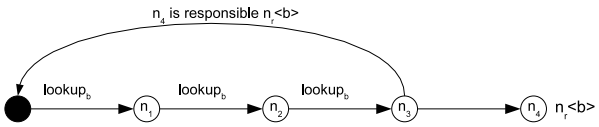


Abbildung 2.8: Beispiel: Abzudeckende Unterbäume der Routingtabelle von 1011

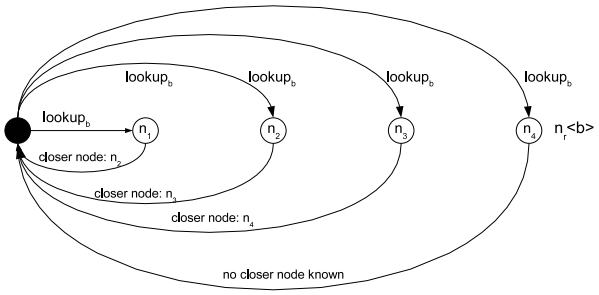
Jede Lookup-Nachricht enthält die NodeID des sendenden Peers. Empfängt ein Knoten eine solche Nachricht, untersucht er die enthaltene NodeID und überprüft den ältesten Eintrag aus dem k-Bucket, dem diese ID zuzuordnen ist. Ist dieser alte Knoten offline, wird er aus der Liste entfernt und die gerade empfangene NodeID an den Anfang der Liste gestellt (Least Recently Seen Policy). Ist er hingegen noch online, so wird er an den Anfang der Liste geschrieben und die NodeID aus dem empfangenen Lookup verworfen. Diese Form des „Lazy Update“ der Routing-Tabellen unterscheidet Kademlia wesentlich von den meisten anderen strukturierten Peer-to-Peer-Protokollen und -Regelwerken.

Die Kontrolle für den Lookup liegt vollständig bei dem Knoten, welcher ihn initiiert: er entscheidet über die jeweils im nächsten Schritt zu kontaktierenden Knoten und übermittelt selbst in jedem Schritt die Lookup-Nachricht an die gewählten Peers - in *allen* anderen P2P-Protokollen wird hingegen die Lookup-Nachricht von Knoten zu Knoten weitergereicht. Einen Vergleich zwischen dem Transfer von Lookup-Nachrichten in Chord (als Repräsentant für die anderen Protokolle) und Kademlia liefern die Abbildungen 2.9(a) und 2.9(b).

In Kademlia existieren zwei verschiedene Lookup-Methoden: `FIND_NODE` und `FIND_VALUE`. In jeder Lookup-Nachricht muss vermerkt sein, welche von beiden verwendet wird.



(a) Chord



(b) Kademlia

Abbildung 2.9: Transfer von Lookup-Nachrichten

Mit der Methode `FIND_NODE` wird nach den k zuständigen Knoten für einen Bezeichner gesucht. Hierbei ermittelt der Initiator zunächst α Knoten aus dem k-Bucket, welches in der Zeile mit dem längsten gemeinsamen Präfix mit b steht. Sucht der Knoten 1011 aus Abb. 2.8 also nach den zuständigen Knoten für den Bezeichner 0111, so verwendet er das k-Bucket für das Präfix 0 (1. Zeile).
Empfängt ein Knoten eine `FIND_NODE`-Nachricht für b, liefert er - wenn möglich - eine Liste von k Knoten zurück, deren NodeIDs eine geringere Distanz zu b aufweisen als seine eigene. Aus diesen maximal $\alpha \cdot k$ pro Schritt gelieferten NodeIDs wählt der Initiator des Lookup wiederum α aus und sendet ihnen $lookup_b$. Dieser Schritt wird wiederholt, bis die k Knoten mit der geringsten Distanz zu b gefunden sind und geantwortet haben.
Die Methode `FIND_VALUE` hingegen sucht konkret nach dem Datensatz mit Bezeichner b. Wenn ein Knoten also eine Lookup-Nachricht mit dieser Methode empfängt, überprüft er zunächst, ob er einen Datensatz mit Bezeichner b gespeichert hat. Wenn ja, liefert er diesen zurück und der Lookup ist beendet. Ansonsten verfährt er wie bei Methode `FIND_NODE` und liefert k Knoten mit näher an b liegenden NodeIDs.
Zum Einfügen eines Datensatzes werden über einen Lookup die k zuständigen Peers ermittelt. Der publizierende Knoten sendet dann jedem von ihnen eine Speicheraufforderung für den Datensatz, so dass k Replika erzeugt werden.
Der Aufwand für den Lookup ist $O(log_2 N)$. Durch Vergrößerung der Routing-Tabelle auf $2^b log_{2^b} N$ Einträge (für erwartete Netzgröße N) kann er auf $O(log_{2^b} N)$ verringert werden.

Join. Wie bereits beschrieben, werden die Routing-Tabellen der Knoten ständig aktuell gehalten, indem die Absender aller eingehenden Lookups untersucht und bei Bedarf gespeichert werden. Ein k-Bucket wird nur dann aktualisiert, wenn ein alter Kontakt das Netz verlassen hat.

Betritt ein Knoten das Netzwerk, nimmt er Kontakt mit einem aktiven Peer n_e auf und fügt diesen als ersten Kontakt in sein entsprechendes k-Bucket ein. Dann initiiert er mittels Weiterleitung einer Lookup-Nachricht an n_e die Suche nach seiner eigenen NodeID n_i. Die im Zuge dieses Lookups kontaktierten Knoten werden dabei bereits von der Anwesenheit des neuen Knotens informiert und erfassen ihn bei Bedarf. Der neue Peer selbst speichert wiederum seine neu gewonnenen Kontakte.
Diese können aber nur die k-Buckets zwischen n_e und seiner eigenen NodeID n_i befüllen. Knoten in anderen, weiter entfernten Unterbäumen wurden beim Lookup nach n_i nicht kontaktiert, da sie ein kürzeres Präfix mit n_i teilen als n_e selbst. Daher führt n_i einen sogenannten Bucket Refresh aus: für jedes noch leere k-Bucket erzeugt er einen Bezeichner mit entsprechendem Präfix und führt einen `FIND_NODE`-Lookup dafür durch, um Kontakte in den noch unbekannten Unterbäumen zu finden.
Sobald die zuständigen Knoten für Bezeichner n_i von dem neuen Peer erfahren, übergeben sie ihm die gespeicherten Datensätze, für die er nun mit zuständig ist.

Leave. Verlässt ein Knoten das Netzwerk, werden keine anderen Peers informiert. Erst im Zuge neuer Lookups wird der Kontakt gemäß der bereits beschriebenen Least Recently Seen Policy nach und nach aus den k-Buckets anderer Knoten entfernt.
Eine Übertragung der gespeicherten Daten kann somit nicht stattfinden - sie gehen verloren. Die verbleibenden zuständigen Knoten halten aber weiterhin Replika der Datensätze und übergeben sie an jeden dem Netz neu beitretenden Peer, dessen NodeID ihn selbst zu einem zuständigen Knoten für diese Datensätze macht.
Zudem wird zur Erhaltung der Replika in regelmäßigen Abständen ein Republishing durchgeführt, d.h. jeder Knoten sendet regelmäßig und selbständig Speicheraufforderungen für seine gespeicherten Datensätze an jeweils alle anderen zuständigen Knoten.

2.5.4 P-Grid

P-Grid [ADH03] [APH02] ist ein Peer-to-Peer-Regelwerk mit Baumtopologie, bei dem jedoch die Knoten selbst keine Blätter des Binärbaums sind. Stattdessen wird der Zuständigkeitsbereich der Peers unabhängig von deren NodeID festgelegt.
Jedem Knoten ist ein Pfad zugewiesen, für den er verantwortlich ist. Er speichert alle Datensätze, deren Bezeichner mit dieser Bitfolge (Präfix) beginnen. Um Fehlertoleranz zu erreichen, können mehrere Knoten für dasselbe Präfix verantwortlich sein; ein Peer kennt stets die anderen Teilnehmer, die für den gleichen Pfad wie er selbst zuständig sind. Es existiert keine a priori feste Bezeichnerlänge für FileIDs, d.h. $f_j \in \{0,1\}^*$.
P-Grid ist nicht nur ein P2P-Regelwerk, sondern definiert zugleich auch eine Public-Key-Infrastruktur nach dem Web-of-Trust-Prinzip, in der die öffentlichen Schlüssel der teilnehmenden Knoten verwaltet werden. Die Peers führen dann bei jeder Interaktion miteinander zuerst ein wechselseitiges Authentifizierungsverfahren durch.

Bemerkung: Um das Verständnis des Lookup zu erleichtern, wird nun zunächst der - sehr komplexe - Join beschrieben.

Join. Ein neuer Knoten bestimmt zunächst seine NodeID n_i aus einem aktuellen Zeitstempel, seiner derzeitigen IP-Adresse und einer Zufallszahl. Der Operator $\dot{+}$ definiert

die Konkatenation von Zeichenketten.

$$n_i = H(timestamp \dot{+} \text{IP-Addr}_i \dot{+} rand).$$

Zudem erstellt er ein Schlüsselpaar $(pubkey_i, privkey_i)$. Mit Hilfe einer digitalen Signaturfunktion $\mathcal{S}$ (siehe Abschnitt 3.4.1) wird unter Verwendung von $privkey_i$ eine Signatur und daraus ein 5-Tupel generiert:

$$\begin{aligned} tsig_{n_i} &= \mathcal{S}((n_i \dot{+} \text{IP-Addr}_i \dot{+} pubkey_i \dot{+} timestamp), privkey_i) \\ t_{n_i} &= (n_i, \text{IP-Addr}_i, pubkey_i, timestamp, tsig_{n_i}) \end{aligned}$$

Nun übermittelt der neue Knoten eine Join-Anfrage mit dem Inhalt t_{n_i} an einen existierenden Peer des P-Grid, der daraufhin einen Lookup nach einem zuständigen Knoten für n_i startet. Dieser Peer $n_r\langle n_i\rangle$ speichert dann das Tupel t_{n_i} als zugehörigen Datensatz zum Bezeichner n_i.
Der Lookup wird k-fach mit verschiedenen Peers als Einstiegsknoten durchgeführt, wobei erwartet wird, dass verschiedene zuständige Knoten für den entsprechenden Pfad gefunden werden, so dass der Datensatz t_{n_i} schließlich auf $k_{rpl} \leq k$ Knoten vorhanden ist.
Betritt ein Knoten das Netz nach einer Offline-Phase, so überprüft er, ob sich seine IP-Adresse seit der letzten Online-Phase geändert hat. Wenn ja, startet er ein Update, d.h. er generiert ein neues Tupel t_{n_i}, welches dann das bisherige aktualisiert. Die NodeID n_i selbst bleibt aber auch bei wiederholtem Eintritt und Verlassen des Netzes erhalten. Der Aufwand für einen Join ebenso wie für einen Wiedereintritt ist $O(log_2 N)$.

Lookup. Der Aufbau der Routing-Tabellen ist im wesentlichen identisch mit dem in Kademlia: für jedes Bit in seinem Pfad speichert der Knoten eine Menge von Kontakten, die für Pfade aus dem komplementären Unterbaum (wie in Kademlia) verantwortlich sind. Jeder Kontakt besteht aus NodeID, IP-Adresse und dem 5-Tupel t_n wie oben beschrieben. Zu jedem komplementären Unterbaum muss mindestens ein Kontakt erfasst sein, eine Obergrenze für die gespeicherten Verweise ist nicht festgelegt.
Empfängt ein Knoten die Anfrage nach einem Bezeichner b, so prüft er zunächst, ob er für das gegebene Präfix selbst zuständig ist. Ist das der Fall, antwortet er direkt. Ansonsten ermittelt er aus seiner Routing-Tabelle das Tupel $t_{n_{next}}$ für einen Knoten mit ID n_{next}, dessen verwalteter Pfad ein längeres gemeinsames Präfix mit b aufweist. Dann wird eine Challenge-Response-Authentifizierung unter Verwendung des im Tupel $t_{n_{next}}$ enthaltenen öffentlichen Schlüssels durchgeführt. Ist sie erfolgreich, gibt der aktuelle Knoten die Suchanfrage $lookup_b$ an den nun authentifizierten Knoten weiter.
Ist die Authentifizierung hingegen nicht erfolgreich, wird angenommen, dass die in der Routing-Tabelle erfasste IP-Adresse von n_{next} nicht mehr aktuell ist. In diesem Fall muss im P-Grid ein neues Tupel des betreffenden Knotens mit dessen neuer IP-Adresse gespeichert sein, welches der aktuelle Peer über einen Lookup anfordert. Dann startet er einen erneuten Versuch der Challenge-Response-Authentifizierung.
Ist n_{next} offline oder aber die Authentifizierung mit dem neuen Tupel nicht erfolgreich, wird die Routing-Tabelle nicht verändert, sondern ein alternativer Knoten mit demselben Pfad-Präfix wie n_{next} aus der Tabelle ausgewählt und die Lookup-Nachricht dorthin weitergeleitet. Die Kosten für einen Lookup sind allgemein $O(log_2 N)$.

Leave. Verlässt ein Knoten das Netz oder fällt aus, sind keine bestimmten Aktionen notwendig. Die Datensätze gehen für die Zeit, in der ein Knoten offline ist, aus dem Netz verloren, was durch die automatische Replikation auf Knoten mit dem gleichen Zuständigkeitspfad aufgefangen werden soll.

2.5.5 Pastry

Pastry [RoD01a] implementiert eine hybride Topologie mit Bezeichnerlänge $m = 128$. Die Knoten sind wie in Chord ringförmig angeordnet, die Routing-Tabellen aber multidimensional aufgebaut, so dass die Pfade im Durchschnitt etwas kürzer als bei Chord sind ($O(log_{2^b} N)$ in Pastry statt $O(log_2 N)$ in Chord).
Bezeichner werden dafür in Bitfolgen der Länge b aufgeteilt, wobei in der Regel $b = 4$, d.h. jede Bitfolge repräsentiert eine Hexadezimalstelle. Die Distanz zwischen zwei Bezeichnern ergibt sich dementsprechend als betragsmäßige Differenz der als Hexadezimalzahlen interpretierten Bitfolgen.

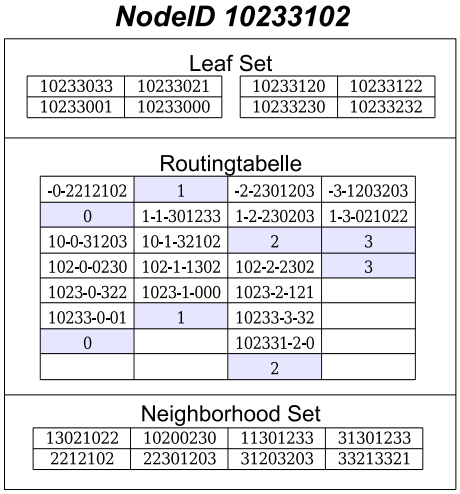


Abbildung 2.10: Beispiel: Pastry-Routing-Tabelle des Knotens 10233102 mit $b = 2$, $m = 16$, $|L| = 8$

Lookup. Es ist stets der Knoten für einen Bezeichner b_l zuständig, dessen NodeID die geringste Distanz zu b_l besitzt, wobei unerheblich ist, ob die NodeID $n_r\langle b\rangle \geq b_l$ oder $< b_l$ ist (auch hier $|\mathcal{N}_{b_l}| = 1$).
Jeder Knoten hält drei Tabellen vor: die Routing-Tabelle, das Neighborhood Set und das Leaf Set. Für den Lookup sind nur die Routing-Tabelle und das Leaf Set von Bedeutung, während das Neighborhood Set Knoten enthält, die auf TCP/IP-Ebene in der Nähe[9] des Tabellenbesitzers liegen.
Die Routing-Tabelle eines Knotens n_i dient der Kontaktaufnahme mit Peers, die nicht zu einer definierten Anzahl (s.u.) seiner nächsten Nachbarn auf Overlay-Ebene gehören.

[9] im Sinne weniger Routingstationen (Hops) oder gleicher Subnetze, in denen sich die Knoten befinden

Sie enthält 2^b Spalten, $\lceil log_{2^b} N \rceil$ Zeilen und muss nicht vollständig belegt sein. In einer Zeile j verweist der Eintrag in der l-ten Spalte ($0 \leq l < 2^b$) auf einen Knoten, der mit n_i die ersten j b-Bit-Ziffern (das Präfix) teilt, aber dessen $j+1$. Ziffer dem Ziffernwert l entspricht. Das Feld in der Spalte, deren Wert l dem Wert der $j+1$. Ziffer von n_i entspricht, bleibt leer (siehe Abb. 2.10).
Das Leaf Set enthält $|L|$ Knoten, welche die gemäß der Distanzmetrik die nächsten zu n_i sind. Dabei haben $\frac{|L|}{2}$ Knoten eine NodeID kleiner der aktuellen, die anderen $\frac{|L|}{2}$ Knoten eine größere. In der Regel gilt $|L| = 2^b = 16$ oder $|L| = 2^{b+1} = 32$. Die Größe des Neighborhood Set ist in der Regel identisch zur Größe des Leaf Set.
Der Initiator eines Lookups nach Bezeichner b_l erstellt $lookup_{b_l}$ und stößt dann das Routing-Verfahren an:

1. Der aktuelle Knoten n_i prüft, ob b_l in den Bereich von NodeIDs fällt, der von seinem Leaf Set abgedeckt wird. In diesem Fall ermittelt er daraus als zuständigen Knoten denjenigen Peer, dessen NodeID die geringste Distanz zu b_l hat, und leitet die Nachricht an ihn weiter. Dieser Knoten ist $n_r\langle b_l \rangle$.

2. Liegt der Bezeichner nicht im Adressbereich des Leaf Set, wird die Routing-Tabelle verwendet:

   - Die Länge l_p des gemeinsamen Präfix von b_l und n_i wird bestimmt. D_{l_p} ist der Wert des Bits von b_l an der $l_p + 1$. Position.
   - Der Knoten ermittelt in der Routing-Tabelle den Eintrag in Zeile l_p und Spalte D_{l_p}. Ist diese Tabellenzelle belegt, leitet er die Lookup-Nachricht an den dort referenzierten Knoten weiter. Mit jedem Schritt wird dabei die Menge der Knoten mit längerem gemeinsamen Präfix mit b_l um den Faktor 2^b verringert.
   - Ist die entsprechende Tabellenzelle nicht belegt, wird die Nachricht an einen Knoten aus dem Leaf Set weitergeleitet, dessen ID-Distanz zu b_l geringer ist als $\Delta(n_i, b_l)$. Dieses Vorgehen gründet sich auf die Vermutung, dass dieser Peer über einen passenden Eintrag in seiner Routing-Tabelle verfügt.

Schritt 2 wird so lange wiederholt, bis bei einem involvierten Knoten die Prüfung in 1. erfolgreich ist. Dann ist der zuständige Knoten gefunden; er sendet eine Rückmeldung an den Initiator der Anfrage. Der Lookup generiert den Aufwand $O(log_{2^b} N)$. Zur Speicherung wird $n_r\langle f_j \rangle$ ermittelt und der Datensatz j dorthin übertragen.

Join. Ein neuer Knoten n_i berechnet seine NodeID und sendet eine Join-Nachricht mit Bezeichner n_i an einen existierenden Knoten n_e. Über das Lookup-Verfahren wird ein Pfad zum zuständigen Knoten mit $n_r\langle n_i \rangle$ ermittelt.
Nun geben alle Knoten innerhalb dieses Pfades ihre drei Tabellen an den neuen Peer weiter. Das Leaf Set wird großteils vom zuständigen Knoten übernommen, das Neighborhood Set vom existierenden Knoten n_e. Die Routing-Tabelle wird wie folgt aufgebaut: Zeile 0 wird im wesentlichen von n_e übernommen, Zeile 1 vom nächsten Knoten im Routing-Pfad usw.
Nach Vervollständigung seiner Tabellen informiert der neue Peer alle ihm nun bekannten Knoten über seine Existenz und seine Tabellenbelegungen, damit diese ihre eigenen Tabellen entsprechend aktualisieren können. Der Aufwand für den Join ist $O(log_{2^b} N)$.

Leave. Es existiert kein Mechanismus zur Abmeldung eines Knotens aus dem Pastry-Netzwerk, die bei ihm gespeicherten Daten gehen verloren. Die Peers, welche den ausgefallenen Teilnehmer in ihren Tabellen haben, müssen diese aktualisieren, sobald sie sein Fehlen bemerken.

2.5.6 Symphony

Symphony [MBR03] ist ein Regelwerk, welches wie Chord eine ringförmige Overlay-Topologie implementiert und die gleiche Distanzmetrik verwendet. Der wesentliche Unterschied liegt in der probabilistischen Auswahl und der variablen Anzahl der Kontakte in der Routing-Tabelle eines Knotens. Das Routing ist bidirektional organisiert. Es existiert zudem keine A-Priori-Festlegung für die Anzahl der Bezeichner-Bits m.

Lookup. Die Routing-Tabelle eines Knotens enthält je einen Zeiger auf direkten Vorgänger und Nachfolger im Ring sowie $q \geq 1$ Verweise auf entferntere Knoten. Ein gängiger Wert ist $q = 4$; q kann aber nach Belieben auch zur Laufzeit an Parameter wie Netzwerkgröße, Netzanbindung eines Knotens oder dessen Leistungsfähigkeit angepasst werden.
Im Gegensatz zu Chord existieren in Symphony keine Vorgaben für die Eigenschaften der Einträge in der Routing-Tabelle (z.B. die Distanz der Kontakte zur eigenen NodeID). Stattdessen werden diese Verbindungen über eine Zufallsverteilungsfunktion ermittelt[10].
Beim Lookup wird immer derjenige Knoten aus der Routing-Tabelle kontaktiert, welcher die Entfernung zum gesuchten Bezeichner b minimiert. Dafür werden nicht nur die q ausgehenden Verbindungen zu entfernten Knoten genutzt (Routing im Uhrzeigersinn) sondern auch die maximal q eingehenden (Inbound) Links, die von anderen Knoten aus auf den aktuellen Knoten zeigen (Routing gegen den Uhrzeigersinn). Der zuständige Knoten mit $n_r\langle b\rangle = succ(b)$ ist der direkte Nachfolger des gesuchten Bezeichners (im Uhrzeigersinn). Der Aufwand für den Lookup ist $O(\frac{1}{q}log_2^2 N)$.
Zum Einfügen eines Datensatzes wird über einen Lookup $succ(f_j)$ ermittelt und der Datensatz an diesen übertragen. Der Knoten $succ(f_j)$ generiert dann selbständig Replika auf seinen $k-1$ Nachfolgern (also ist $|\mathcal{N}_b| = k$).

Join. Beim Eintritt in das Netzwerk berechnet der neue Knoten seine NodeID n_i, ermittelt über einen Lookup seinen direkten Vorgänger und direkten Nachfolger und füllt dann seine Routing-Tabelle mit weiter entfernten Knoten wie folgt:
Er ermittelt einen zufälligen Bitstring $x \in \{0,1\}^m$ mit Hilfe einer Zufallsverteilungsfunktion. Dann sucht er den zuständigen Knoten für x, den er in die Routing-Tabelle aufnimmt. Dieser Schritt wird q-mal durchgeführt. Der Aufwand für den Join ist $O(log_2^2 N)$.

Leave. Die eingehenden Verbindungen werden gelöst, indem der verlassende Peer die betroffenen Knoten per Nachricht über den bevorstehenden Leave informiert. Diese füllen ihre Routing-Tabellen dann mit je einem neuen zufällig gewählten Kontakt auf. Weiterhin müssen Vorgänger und Nachfolger ihre Zeiger aktualisieren, um den Ring wieder zu schließen. Die Übertragung der gespeicherten Datensätze ist kein Bestandteil des beschriebenen Leave-Mechanismus. Der Aufwand ist wie beim Join $O(log_2^2 N)$.

[10] Die durch diese Funktion entstehende Verteilung gehört zur Klasse der harmonischen Verteilungen - daher die Bezeichnung Symphony.

2.5.7 Viceroy

Viceroy [MNR02] implementiert ein für Netze mit hohem Datenaufkommen optimiertes Overlay, in dem die Knoten wie bei Chord ringförmig angeordnet sind. Zusätzlich existieren miteinander vernetzte Ebenen. Die entstehende Topologie wird als Butterfly-Topologie[11] bezeichnet [ACE96].
Für alle Bezeichner gilt $m = 128$. Zusätzlich zu seiner NodeID besitzt jeder Knoten eine Ebene (Level) l, auf der er angesiedelt ist. Für einen Bezeichner ist wie in Chord der direkte Nachfolger $n_r\langle b\rangle = succ(b)$ zuständig; auch die Distanzdefinition ist identisch.

Lookup. Jeder Knoten n_i speichert in seiner Routing-Tabelle maximal sieben Kontakte:

- Direkter Vorgänger und Nachfolger auf dem Ring:
  $succ(n_i)$ und $pred(n_i)$ (Ring Pointer)
- Vorgänger und Nachfolger auf gleicher Ebene $l(n_i)$:
  $nextonlevel(n_i)$ und $prevonlevel(n_i)$ (Level Ring Pointer)
- Zwei Knoten auf der nächsthöheren Ebene $l(n_i) + 1$, sofern $l(n_i) \neq max$:
  $right(n_i)$ und $left(n_i)$.
- Einen Knoten auf der nächstniedrigeren Ebene $l(n_i) - 1$, sofern $l(n_i) > 1$:
  $up(n_i)$.

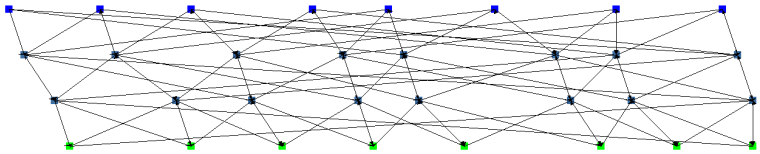
Die Kontakte $right(n_i)$, $left(n_i)$ und $up(n_i)$ werden Butterfly Links genannt. Ein Knoten hält aber nicht nur die beschriebenen Kontakte, sondern verwaltet auch die eingehenden Zeiger anderer Knoten auf ihn selbst (Level Ring Pointer und Butterfly Links). Abbildung 2.11 zeigt die Vernetzung der Ebenen an einem kleinen Beispielnetz in Level- und Ringdarstellung[12].
Ein Lookup nach Bezeichner b wird in drei Phasen ausgeführt:

1. *Proceed to Root:* Mittels der Weiterleitung über *up*-Links wird die Nachricht an einen Knoten auf Level 1 transferiert.
2. *Traverse Tree:* Über abwärtsgerichtete Links ($right$ oder $left$) wird die Lookup-Nachricht in Richtung des Ziels weitergeleitet, wobei $left(n_i)$ verwendet wird, wenn

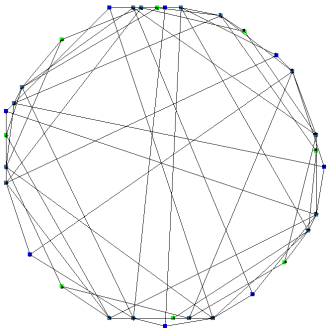
   $$\Delta(n_i, b) < \frac{1}{2^{l(n_i)}}$$

   für den aktuellen Knoten n_i, ansonsten $right(n_i)$. Diese Phase wird beendet, wenn der benötigte abwärtsgerichtete Link nicht existiert oder über das Ziel hinausschießt, d.h. eine NodeID $> b$ besitzt.
3. *Traverse Ring:* Diese Phase wird rekursiv durchlaufen:

[11] In Anlehnung an die Bezeichnung „Butterfly-Topologie“ wurde der Name Viceroy gewählt: Viceroy (Vizekönig) ist eine nordamerikanische Schmetterlingsart (Limenitis archippus).
[12] Die Abbildungen wurden mit dem Viceroy-Applet [VICAP] generiert.



(a) Level-Darstellung



(b) Ring-Darstellung

Abbildung 2.11: Beispiel: Butterfly Links in einem Viceroy-Netzwerk mit 32 Knoten

- Besitzt der aktuelle Knoten die NodeID $n_i = succ(b)$, ist der Lookup erfolgreich beendet und er sendet eine Antwort, dass er für b zuständig ist.
- Ansonsten:
  - Ist $nextonlevel(n_i)$ ein Knoten im Adressraum zwischen n_i und b, wird die Lookup-Nachricht an diesen übertragen.
  - Liegt $prevonlevel(n_i)$ zwischen n_i und b, wird der Lookup an diesen übermittelt.
  - Ist beides nicht der Fall, wird $succ(n_i)$ oder $pred(n_i)$ kontaktiert, je nachdem, welcher der beiden Knoten näher an b liegt.

Es wird also zunächst auf dem gleichen Level vorangeschritten und erst in der Nähe des zuständigen Knotens für b auf die Ring Pointer zurückgegriffen.

Der Aufwand für einen Lookup ergibt sich zu $O(log_2 N)$, da jede Phase selbst den Aufwand $O(log_2 N)$ erzeugt.
Die Speicherung von Datensätzen erfolgt wie in Chord ($|\mathcal{N}_b| = 1$).

Join. Beim Eintritt eines neuen Knotens in ein Viceroy-Netzwerk bestimmt dieser zunächst seine NodeID n_i. Dann startet er einen Lookup nach n_i und erhält $n_r\langle n_i\rangle = succ(n_i)$. Von diesem bezieht er die Daten seines direkten Vorgängers und setzt seine Ring Pointer $succ(n_i)$ und $pred(n_i) = pred(succ(n_i))$. Vorgänger und Nachfolger aktualisieren zugleich ihre eigenen Ring Pointer. Im nächsten Schritt überträgt $n_r\langle n_i\rangle$ alle Daten an n_i, für welche dieser nun verantwortlich ist. Der Level $l(n_i)$ wird zufällig gewählt, wobei die Anzahl der zur Verfügung stehenden Ebenen von der geschätzten Netzgröße N abhängt. Zuletzt ermittelt der neue Knoten noch seine Butterfly Links: $left(n_i)$ ist der nächste

Knoten im Ring, der einen Level $l(n_i) + 1$ hat. $right(n_i)$ ist der Knoten im Ring, der ausgehend vom Knoten

$$succ(n_i + \frac{1}{2^{l(n_i)}})$$

der nächste auf Ebene $l(n_i) + 1$ ist. $up(n_i)$ ist der nächste Peer im Ring, der auf Ebene $l(n_i) - 1$ liegt. Der Aufwand für den Join ist $O(log_2\ N)$.

Leave. Verlässt ein Knoten das Netzwerk, so informiert er alle Knoten, die ihn als Ring Pointer, Level Ring Pointer oder Butterfly Pointer referenzieren; diese aktualisieren daraufhin ihre Verweise. Dann überträgt er alle bei ihm gespeicherten Daten an seinen Nachfolger. Der Aufwand ist $O(log_2\ N)$.

Kapitel 3

Authentifizierungs- und Autorisierungsinfrastrukturen

In diesem Kapitel werden zunächst die konzeptionellen Grundlagen von AAI vorgestellt, gefolgt von einer Kategorisierung und einer kurzen Einführung in einige Beispielsysteme. Abhängig von der Kategorie einer AAI bestehen spezielle Anforderungen an das zugrunde liegende Datenverwaltungssystem, so dass verschiedene AAI-Kategorien verschiedenen Paradigmen für die Datenverwaltung zuzuordnen sind. Auf Basis einer erweiterten Zertifikatnotation wird ein rollenbasiertes AAI-Modell hergeleitet, welches später für die formale Modellierung (Kapitel 6) wieder aufgegriffen wird.
Die Vorstellung verschiedener AAI-Architekturen bildet einen weiteren Abschnitt. Abschließend wird die Status- bzw. Rückrufproblematik in AAI eingeführt und verschiedene Techniken zur Erzeugung von Status- oder Rückrufinformationen diskutiert.

3.1 Grundlagen und Definitionen

Definition 3.1 ***Authentifizierung***
Authentifizierung ist ein Prozess, in welchem geprüft wird, ob eine Entität eine von ihr beanspruchte Identität besitzt. Ihr Ergebnis ist die Sicherheit des Prüfers darüber, ob die angegebene Identität authentisch der Entität zugeordnet werden kann.

Ein Beispiel aus der Realität ist die Authentifizierung einer natürlichen Person mit Hilfe ihres Personalausweises. Die enthaltenen Personendaten und das Lichtbild ermöglichen einem Prüfer den Abgleich zwischen der realen Person und der Identität, die im Personalausweis zertifiziert ist.
Die Entität, deren Identität geprüft wird, muss nicht zwingend eine Person sein. Beispielsweise kann sich auch ein Computersystem gegenüber einem anderen authentifizieren, z.B. im Rahmen von IPSec oder SSL [AlS04]. Weiterhin kann sich die Authentifizierung auf ein Pseudonym beziehen, d.h. es muss nicht zwangsläufig eine reale Identität verwendet werden - auch frei wählbare Benutzernamen oder Rechnernamen sind möglich, solange sie einer Entität zweifelsfrei zuordenbar sind.

Definition 3.2 ***Autorisierung***
Autorisierung ist ein Prozess, in welchem geprüft wird, über welche Berechtigungen eine

Entität im Rahmen einer definierten Umgebung verfügt. Ihr Ergebnis ist die Sicherheit des Prüfers darüber, welche Aktionen eine Entität ausführen darf.

Die Autorisierung dient dann als Entscheidungsgrundlage für den Prüfer, ob die Anfrage der betreffenden Entität nach Ausführung einer Aktion (z.B. eines Online-Einkaufs) oder nach dem Zugriff auf bestimmte Daten (z.B. auf eine elektronische Zeitschriftenbibliothek) positiv beantwortet wird.
Für die sichere Durchführung der Autorisierung ist es unerlässlich, dass die zu autorisierende Entität zuvor authentifiziert wurde. Ansonsten kann kein Abgleich zwischen dem Inhaber der beanspruchten Rechte und der Entität selbst erfolgen.

Definition 3.3 ***Authentifizierungs- und Autorisierungsinfrastruktur (AAI):*** *Eine Authentifizierungs- und Autorisierungsinfrastruktur ist ein System von Protokollen, Regeln, Daten, technischen Gegebenheiten und Methoden, welches die Dienste Authentifizierung und Autorisierung (A&A) von Entitäten gegenüber Prüfern, z.B. Ressourcenanbietern, ermöglicht und die dafür notwendigen Daten speichert, verwaltet und zur Verfügung stellt.*

Authentifizierungs- und Autorisierungsinfrastukturen sind ein junges Forschungsgebiet, das erst seit dem Jahr 2000 intensiv wissenschaftlich bearbeitet und weiterentwickelt wird. Viele in diesem Rahmen entstandene Systeme sind bislang noch im Forschungs- und Entwicklungsstadium und nicht oder nur vereinzelt im praktischen Einsatz. AKENTI [AKENT] und PERMIS [PERMI] realisieren nur den Autorisierungs-Anteil der AAI in Form einer zertifikatbasierten Privilege-Management-Infrastruktur und verlangen die Kombination mit einem Authentifizierungssystem wie z.B. Kerberos [KERBE] oder FPKI [FPKI]. SDSI/SPKI [ACM02] [RiL96] hingegen bietet sowohl Authentifizierung als auch Autorisierung auf Basis von Zertifikaten.
SESAME [SESAM] ist ein europäisches AAI-Forschungsprojekt, welches ein Framework für die Entwicklung konkreter AAI-Produkte bereitstellt, wie beispielsweise von der Firma Comarch angeboten [COMAR]. Microsoft .NET Passport [PASSP] wird als Single-Sign-On-Lösung für Authentifizierung und Autorisierung auf vielen Internetplattformen genutzt. Das schwerpunktmäßig für die Anwendung im Bildungsbereich entwickelte AAI-System Shibboleth [SHIBB] wird z.B. im Rahmen der Initiative *Switch AAI* als gemeinsame AAI für alle Schweizer Universitäten [SWITC] und im Projekt AAR [AAR] der Universitätsbibliotheken Freiburg und Regensburg verwendet. AAR soll ab 2008 zur Kontrolle von Zugriffsrechten auf elektronische Zeitschriften eingesetzt werden.

Aufgrund der allgemein zunehmenden Digitalisierung von Diensten in der Verwaltung wissenschaftlicher und öffentlicher Einrichtungen und auch in der Wirtschaft, z.B. im Bankensektor oder im E-Commerce, und der gestiegenen Sicherheitsanforderungen an solche Dienste ist mit einer zunehmenden Verwendung von AAI zur Realisierung von Zugriffs- und Berechtigungskontrollen zu rechnen.

3.2 Kategorisierung von AAI

Die existierenden AAI-Konzepte lassen sich hinsichtlich der als Grundlage für den Dienst verwendeten Datenstrukturen kategorisieren. Diese Unterscheidung ist insbesondere sinn-

voll im Hinblick auf das Zusammenspiel einer AAI mit einem verteilten P2P-Verzeichnis für die Datenverwaltung. Wie im folgenden Abschnitt 3.3 analysiert, eignen sich unterschiedliche Paradigmen (Client-Server bzw. Peer-to-Peer) als Grundlage für die Datenverwaltung der verschiedenen AAI-Kategorien.

Die Beispielsysteme für die Kategorien werden nur knapp skizziert, da keine Festlegung auf ein bestimmtes System erfolgen soll. Stattdessen ist flexible Einsetzbarkeit des P2P-Verzeichnisses für möglichst viele verschiedene AAI-Systeme angestrebt.

3.2.1 Online-AAI mit nicht zertifikatbasierter Datenhaltung

In einer Online-AAI werden die für die Dienste notwendigen Daten stets auf Anfrage eines Prüfers spontan von einem Sicherheitsserver bereitgestellt. Dieser verfügt dafür über eine Datenbank, in welcher die entsprechenden Informationen - z.B. Passwörter und Zugriffslisten - verwaltet werden. Bei einer Anfrage prüft nun der Server seine Datenbankeinträge und generiert eine Antwort, z.B. in Form einer Authentizitätsbestätigung für die betreffende Entität[1]. Die Authentifizierung und Autorisierung kann also nur dann erfolgreich durchgeführt werden, wenn zum Zeitpunkt des A&A-Verfahrens eine direkte Verbindung zwischen dem Server und dem Prüfer besteht.
Die dem Dienst zugrunde liegenden Daten sind dementsprechend Online-Informationen, d.h. sie müssen bei jedem Authentifizierungs- und Autorisierungsvorgang für dieselbe Entität neu angefragt werden.

Beispiele
Die Microsoft-Technologie **.NET Passport** [PASSP] [LOP03] erleichtert die Authentifizierung von Benutzern in Web-Umgebungen mittels Kombinationen von Benutzername und Passwort (im Folgenden kurz *UnP* : Username + Password). Die UnP-Kombinationen werden zentral auf einem Microsoft-Server gespeichert. Will sich ein Benutzer auf einer Partnerseite von .NET Passport einloggen, wird seine Anfrage an den .NET Passport Server weitergeleitet und übermittelt an diesen seine UnP-Kombination. Als Antwort erhält er Cookies mit Authentizitätsinformationen für die Partnerseite, an welche die aktuelle Sitzung nun wieder umgeleitet wird. Vorteil dieser Lösung ist die Single-Sign-On-Funktionalität (SSO): es ist nicht notwendig, für jede der Partnerseiten eine eigene UnP-Kombination zu besitzen, sondern es genügt eine einzige für alle. Die Autorisierung von Benutzern findet dann wieder dezentral bei den einzelnen Partnerseiten statt, die ihre eigenen Autorisierungsschemata verwalten.
Liberty Alliance [LIBER] [LOP03] ist eine Allianz von über 160 Organisationen, z.B. Verisign, AOL, Nokia, Visa, Sony und Sun Microsystems, mit dem Ziel der Generierung eines Standards für dezentralisierte SSO-Authentifizierung und offene Autorisierung im Web. Das Liberty-Alliance-System verwendet keine globale Kennung für einen Benutzer, sondern fördert die Verbundbildung und den Datenaustausch zwischen den Teilnehmerseiten, bei denen ein Benutzer verschiedene Kennungen besitzen kann.
Die Dienstanbieter interagieren mit sogenannten *Identity Providers*, welche die Authentifizierung der Benutzer durchführen und dann Rückmeldung an den Dienstanbieter geben.

[1] Diese kann auch ein kurzlebiger digital signierter Datensatz (Zertifikat) sein. Die Kategorisierung der AAI ergibt sich aufgrund der *langfristigen* Speicherform und damit der Art des Dienstangebots.

Ebenso liefern *Attribute Providers* Informationen über Eigenschaften von Benutzern, so dass der Dienstanbieter auf dieser Basis eine Entscheidung treffen kann, ob ein Benutzer über die entsprechenden Berechtigungen verfügt.
Entgegen dem Microsoft-Ansatz arbeitet Liberty auf Basis offener Standards wie SOAP [BaN04] und SAML [CKP05] und stellt den Partnern die Umsetzung des Standards in verschiedene Implementierungen frei[2]. Die zu verwendenden Authentifizierungsmethoden sind prinzipiell nicht festgelegt; es herrscht aber auch hier UnP-Authentifizierung auf Basis von Datenbanken vor.
Shibboleth [SHIBB] ist eine im Rahmen des Internet2-Projektes [INTER] entwickelte und inzwischen insbesondere im akademischen Umfeld verbreitete AAI für browserbasierte Dienste. Wie Liberty setzt Shibboleth auf offene Standards. Authentifizierung wird bei einer sogenannten Home Organization auf Basis ihrer Datenbanken durchgeführt. Sie liefert dann an den Ressourcenanbieter die Attribute des authentifizierenden Benutzers, anhand derer ihm der Anbieter Zugriff auf bestimmte Ressourcen erlaubt oder verweigert.

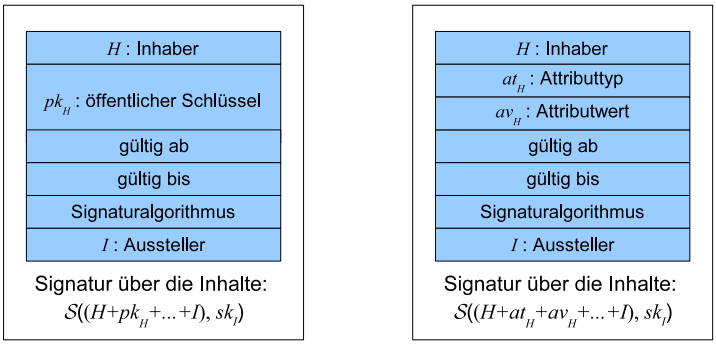
3.2.2 Offline-AAI mit zertifikatbasierter Datenhaltung

In einer zertifikatbasierten AAI bilden digital signierte Datensätze (Zertifikate) die Grundlage für die Realisierung der Authentifizierung und Autorisierung.

Definition 3.4 ***Zertifikat***
Ein Zertifikat ist ein elektronisches Dokument, in dem einer Entität eine bestimmte Eigenschaft, ein öffentlicher Schlüssel, ein Attribut oder eine Berechtigung zugeordnet und beglaubigt wird. Ein Zertifikat ist stets von einer Zertifizierungsstelle (Aussteller) unter Verwendung seines geheimen Signaturschlüssels digital signiert[3]*.*
Die Signatur kann von einem Prüfer mit Hilfe des zum Signaturschlüssel gehörenden öffentlichen Testschlüssels des Ausstellers verifiziert werden.

Auf Basis der in den Zertifikaten enthaltenen Informationen werden Authentifizierung und Autorisierung zwischen einer Entität und einem Anbieter dieser Dienste (Prüfer) durchgeführt.
Ein Aussteller generiert ein Zertifikat und publiziert es dann in einem öffentlichen Verzeichnis, wo es von Prüfern angefordert werden kann. Weiterhin werden Rückrufe oder Gültigkeitsbestätigungen (zusammenfassend als Statusnachrichten bezeichnet) in digital signierter oder aber in gehashter Form publiziert. Sie erlauben es einem Prüfer zu ermitteln, ob ein gegebenes Zertifikat, welches noch nicht sein reguläres Gültigkeitsende erreicht hat, vorzeitig vom Aussteller für ungültig erklärt wurde oder nicht. Es ist also keine direkte synchrone Verbindung zwischen der Zertifizierungsstelle und dem Prüfer zum Zeitpunkt des konkreten Authentifizierungs- und Autorisierungsverfahrens notwendig. Daher wird diese AAI als Offline-AAI bezeichnet.
Die Zertifikate und Statusnachrichten selbst gehören zur Klasse der Offline-Informationen. Sie können nach einmaliger Anfrage aus dem Verzeichnis bei einem Prüfer lokal gespeichert und verwendet werden.

[2] z.B. existiert Liberty-Software in Java und C
[3] siehe Abschnitt 3.4.1



(a) Schlüsselzertifikat (b) Attributzertifikat

Abbildung 3.1: Schematische Darstellung von Zertifikaten

Beispiele
Im ITU-T-Standard **X.509** [HPF02] [Cha02] existieren neben den klassischen Schlüsselzertifikaten für die Authentifizierung von Entitäten Attributzertifikate, welche beglaubigte Informationen über Eigenschaften und Berechtigungen des Inhabers repräsentieren. Diese Zertifikate können, müssen aber nicht, nach Typ getrennt in unterschiedlichen Systemen gespeichert und verwaltet werden [DLM02], z.B. um in Firmen das Outsourcing von Authentifizierungsdiensten zu erlauben, während die möglicherweise vertraulichen Attribute im Rahmen einer Privilege-Management-Infrastruktur (PMI) intern vorgehalten werden. Ein Beispiel für eine X.509-Implementierung ist der Dienst **VOMS** (Virtual Organization Membership Service) [ACC03], welcher für die Zugriffskontrolle auf Ressourcen in Grid-Umgebungen konzipiert ist. Auch **AKENTI** und **PERMIS** als PMI-Lösungen implementieren den X.509-Standard.
In **SDSI/SPKI** [ACM02] werden Zertifikate für die Zuordnung von öffentlichen Schlüsseln (für Authentifizierung) und Berechtigungen (für Autorisierung) zu Entitäten verwendet. Im Gegensatz zu X.509 existieren hier verschiedene Namensräume, so dass die Bezeichnungen der Entitäten nicht global eindeutig sein müssen. Ein weiterer Unterschied zu X.509 ist der Aufbau der Attributzertifikate: anstatt eine Bindung zwischen einem Entitätsbezeichner und einem Attribut oder einer Berechtigung zu zertifizieren, wird in einem SDSI/SPKI-Attributzertifikat eine Zuordnung zwischen einem öffentlichen Schlüssel und einer Berechtigung beglaubigt. Nur in den Schlüsselzertifikaten wird der Bezeichner der Entität mit ihrem öffentlichen Schlüssel verknüpft. Die Entwickler des Systems empfehlen die dezentrale Speicherung der Zertifikate bei ihren Inhabern.

3.3 AAI-Kategorien und geeignete Netzwerkparadigmen für die Datenverwaltung

In den beiden präsentierten AAI-Kategorien werden die Daten in unterschiedlicher Art und Weise vorgehalten, verwaltet und verwendet. Die dezentralisierte, selbstorganisierende Datenspeicherung in einem Peer-to-Peer-Netzwerk ist nicht für beide Kategorien gleichermaßen geeignet.

Online-AAI
In einer Online-AAI wird bei jedem A&A-Vorgang eine Verbindung zwischen dem Prüfer und einem Sicherheitsserver hergestellt. Letzterem müssen die Daten für die Diensterbringung, z.B. UnP-Kombinationen, Access Control Lists oder Attributlisten, immer und unmittelbar zur Verfügung stehen.
Der Prüfer kann den A&A-Prozess daher nicht selbständig durchführen, sondern ist auf eine oder wenige privilegierte Instanzen angewiesen. Die Daten sind in der Regel auf dedizierten Datenbankservern gespeichert, auf welche nur die privilegierten Diensteanbieter Zugriff haben. Ein Diensteanbieter selbst muss ein hochverfügbares Serversystem mit entsprechender Bandbreite betreiben, um eingehende Anfragen verzögerungsfrei beantworten zu können. Damit besteht bereits eine Serverplattform, auf der eine solche Datenbank betrieben werden kann.
Bei Speicherung der Daten in verteilter Form entstünden daher keine wesentlichen Vorteile, dafür aber zusätzlicher Aufwand für Sicherheitsmechanismen wie beispielsweise Replikation der Datensätze. Für eine Online-AAI ist also nur das Client-Server-Paradigma sinnvoll einsetzbar.

Offline-AAI
In einer Offline-AAI können Authentifizierung und Autorisierung dezentral realisiert werden. In diesem Fall ist keine privilegierte Instanz notwendig, die bei jedem A&A-Vorgang kontaktiert werden muss, sondern ein Ressourcenanbieter kann selbständig prüfen, ob eine Entität die notwendigen Rechte besitzt um auf die Ressource zuzugreifen. Er benötigt nur Zugang zur Datenbasis der AAI, welche in Form von Zertifikaten vorliegt.
Die Zertifikate besitzen durch die Signatur per se eine Integritätssicherung. Es kann also leicht geprüft werden, ob es nach der Erstellung manipuliert wurde. Dementsprechend herrscht bereits ein gewisser Sicherheitslevel, welcher die Daten weniger angreifbar macht, auch wenn sie auf verschiedenen Computersystemen vorgehalten werden.
Da Offline-AAI ein dezentralisiertes Dienstmodell ermöglichen, ist auch für die Datenhaltung ein dezentralisierter Ansatz zu präferieren. So kann ohne die Notwendigkeit zentraler Datenbankserver eine AAI mit einem kostengünstigen, gut skalierbaren und organisatorisch unabhängigen verteilten Datenverwaltungssystem realisiert werden. Vor diesem Hintergrund ist ein P2P-Verzeichnis für die Speicherung und Bereitstellung der Zertifikate und Statusnachrichten einer Offline-AAI sehr gut geeignet.

3.4 Aufbau einer zertifikatbasierten AAI

Eine zertifikatbasierte AAI gliedert sich in drei wesentliche Teile [Woe06]:

- *A&A-Verfahren:* Hierbei handelt es sich um die konkreten Methoden und Protokolle zur Durchführung der Authentifizierung und Autorisierung einer Entität bei einem Prüfer.

- *Public-Key-Infrastruktur (PKI):* Eine PKI definiert Regeln, um auf Basis von verwalteten Schlüsselzertifikaten den authentischen öffentlichen Schlüssel einer Entität herzuleiten.

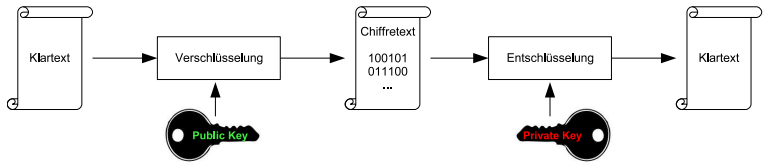
- *Privilege-Management-Infrastruktur (PMI):* Eine PMI definiert Regeln, um auf Basis von verwalteten Attribut- und Privilegzertifikaten authentische Attribute oder Berechtigungen einer Entität herzuleiten.

Die A&A-Verfahren werden jeweils in den Abschnitten 3.4.2 und 3.4.3 in Vorbereitung der Beschreibung von PKI bzw. PMI erläutert.

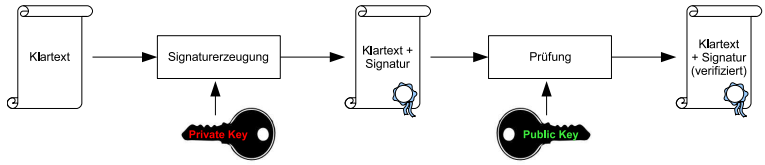
3.4.1 Grundlagen: Public-Key-Kryptographie

AAI basieren sowohl hinsichtlich der konkreten Verfahren als auch in Bezug auf die grundlegenden Datentypen in wesentlichem Maße auf Public-Key-Kryptographie.

In der klassischen symmetrischen Kryptographie wird zur Ver- und Entschlüsselung eines Datensatzes der gleiche geheime Schlüssel verwendet. Anders ist es in der auch als asymmetrisch bezeichneten Public-Key-Kryptographie: Hier kommen *Schlüsselpaare* zum Einsatz, die sich aus einem geheimen (privaten) und einem öffentlichen Schlüssel zusammensetzen.
Der Besitzer eines solchen Schlüsselpaares publiziert seinen öffentlichen Schlüssel. Mit diesem kann jede andere Entität einen Datensatz für den Besitzer chiffrieren oder eine von ihm ausgestellte digitale Signatur prüfen. Der Besitzer kann einen für ihn verschlüsselten Datensatz mit seinem privaten Schlüssel dechiffrieren mit diesem beliebige Datensätze digital unterschreiben.
Abbildung 3.2 zeigt die Verwendung von öffentlichem und privatem Schlüssel zur Verschlüsselung und zur digitalen Signatur.



(a) Ver- und Entschlüsselung



(b) Signaturerzeugung und -prüfung

Abbildung 3.2: Public-Key-Kryptographie: Schematische Darstellung

Ein Schlüsselpaar wird durch Anwendung eines definierten Protokolls erzeugt. Es ist praktisch unmöglich, zu einem bekannten öffentlichen Schlüssel den passenden privaten Schlüssel zu erzeugen. Weiterhin ist es nicht möglich, aus einem Chiffretext (Chiffrat) den Klartext zu ermitteln, ohne den privaten Schlüssel zu kennen. Schlüsselgenerierung und Verschlüsselung basieren dabei auf der Klasse der Einwegfunktionen (siehe auch Def.

2.16 und [MOV96]): ihren Funktionswert $f(x) = y$ zu berechnen ist einfach, aus einem Funktionswert y aber $f^{-1}(y) = x$ zu ermitteln ist praktisch nicht durchführbar.
Bekannte Beispiele für Public-Key-Verfahren sind die ElGamal-Verschlüsselung und die RSA-Verschlüsselung bzw. -Signatur. Die Sicherheit des ersteren basiert auf der Annahme, dass die Berechnung des diskreten Logarithmus nicht mit vertretbarem Aufwand möglich ist, die der letzteren auf der Vermutung, dass die Faktorisierung des Produkts zweier großer Primzahlen schwierig ist. Für weitere Details zu Public-Key-Verfahren und deren mathematischen Grundlagen sei auf Basisliteratur wie [Sti02], [MOV96] und [Sch95] verwiesen.

Definition 3.5 ***Ver- und Entschlüsselungsfunktionen $\mathcal{E}$, $\mathcal{D}$***
Eine Funktion, welche der asymmetrischen Verschlüsselung eines Datensatzes (in Form eines Bitstrings beliebiger Länge) mit Hilfe eines öffentlichen Schlüssels (in Form eines Bitstrings fester Länge l) dient, wird mit $\mathcal{E}$ bezeichnet,

$$\mathcal{E} : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^*.$$

Eine Funktion, welche die Dechiffrierung eines asymmetrisch verschlüsselten Datensatzes mit Hilfe eines privaten Schlüssels der Länge l realisiert, wird mit $\mathcal{D}$ bezeichnet,

$$\mathcal{D} : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^*.$$

Dabei gilt mit korrespondierendem Schlüsselpaar (sk_i, pk_i):

$$\mathcal{D}(\mathcal{E}(x, pk_i), sk_i) = x$$

Definition 3.6 ***Signatur- und Testfunktionen $\mathcal{S}$, $\mathcal{T}$***
Eine Funktion, welche der digitalen Signatur eines Datensatzes (in Form eines Bitstrings beliebiger Länge) mit Hilfe eines privaten Signaturschlüssels der Länge l dient, wird mit $\mathcal{S}$ bezeichnet,

$$\mathcal{S} : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^*.$$

Eine Funktion, welche die Überprüfung eines digital signierten Datensatzes mit Hilfe eines öffentlichen Testschlüssels der Länge l realisiert, wird mit $\mathcal{T}$ bezeichnet,

$$\mathcal{T} : \{0,1\}^* \times \{0,1\}^l \rightarrow \{true, false\}.$$

Dabei gilt mit korrespondierendem Schlüsselpaar (sk_i, pk_i):

$$\begin{aligned} \mathcal{T}(\mathcal{S}(x, sk_i), pk_i) = true \rightarrow &\quad \textit{Verifikation erfolgreich (Signatur ist gültig)} \\ \mathcal{T}(\mathcal{S}(x, sk_i), pk_i) = false \rightarrow &\quad \textit{Verifikation fehlgeschlagen (Signatur ist ungültig)} \end{aligned}$$

3.4.2 Public-Key-Infrastrukturen

Bevor eine Entität autorisiert werden kann, muss zunächst ihre Authentizität zweifelsfrei feststehen. Dafür wird ein Authentifizierungsverfahren auf Basis von Public-Key-Kryptographie verwendet.

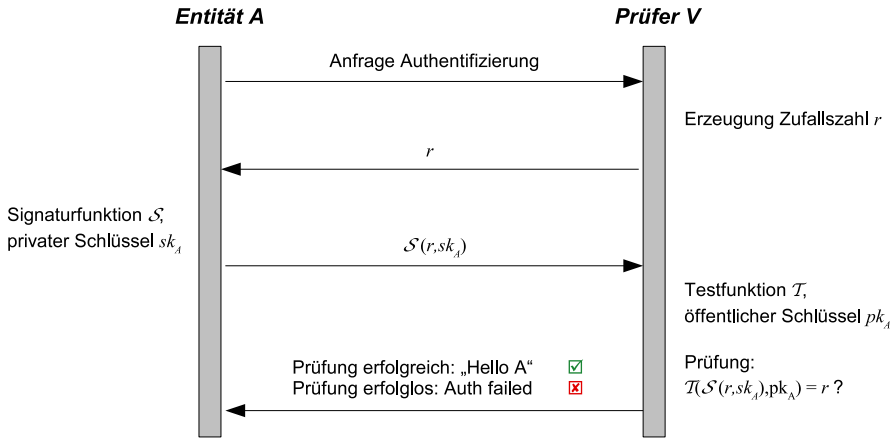


Abbildung 3.3: Beispiel: Simples Verfahren zur einseitigen Authentifizierung

In Abbildung 3.3 wird ein simples Verfahren skizziert, mit dem sich ein Prüfer der Authentizität der mit ihm interagierenden Entität versichern kann.
Für die Prüfung der Antwort von Entität A benötigt der Prüfer V den öffentlichen Schlüssel von A, pk_A. Er kann sich nur dann sicher sein, dass die Antwort $\mathcal{S}_{sk_A}$ tatsächlich von A stammt, wenn der ihm bekannte öffentliche Schlüssel von A echt ist. Es ist einsichtig, dass die elektronische Übermittlung von pk_A von A an V nicht ausreicht: eine böswillige Entität M könnte V einen Schlüssel pk_M übermitteln und diesen als öffentlichen Schlüssel von A ausgeben. V würde dann nach erfolgreichem Durchlauf des Authentifizierungsverfahrens glauben, A habe sich bei ihm authentifiziert, obwohl A am Verfahren nicht beteiligt war (siehe Abb. 3.4).

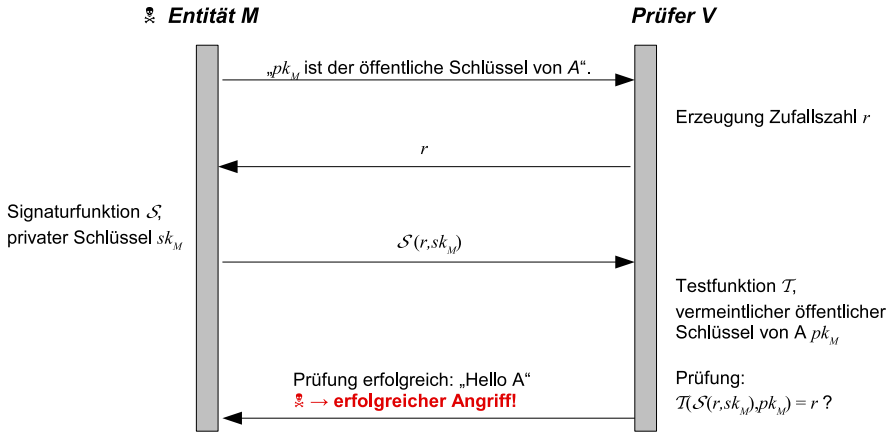


Abbildung 3.4: Beispiel: Angriff auf das Authentifizierungsverfahren

Es muss also sichergestellt werden, dass dem Prüfer dieser authentische öffentliche Schlüssel der mit ihm kommunizierenden Entität zur Verfügung steht und er sich über dessen Authentizität sicher sein kann.

Schlüsselzertifikate bilden daher die fundamentale Datenstruktur in einer PKI:

Definition 3.7 ***Schlüsselzertifikat***
Ein Schlüsselzertifikat ist ein Zertifikat, in welchem einer Entität (Inhaber) ein öffentlicher Schlüssel zugeordnet wird. Die Zertifizierungsstelle beglaubigt mit ihrer digitalen Signatur, dass sie von der Authentizität dieser Zuordnung überzeugt ist.

Ein Prüfer V wird nun genau dann pk_A als authentischen öffentlichen Schlüssel von A akzeptieren, wenn ihm ein Schlüsselzertifikat vorliegt, in dem die Zuordnung $A \rightarrow pk_A$ beglaubigt wird und das von einer Zertifizierungsstelle Z ausgestellt ist, welcher V vertraut und deren authentischen öffentlichen Schlüssel pk_Z zum Test der Signatur auf dem Zertifikat er besitzt.
Verfügt er nicht über pk_Z, wird er ein weiteres Schlüsselzertifikat heranziehen, in welchem die Zuordnung $Z \rightarrow pk_Z$ von einer Zertifizierungsstelle Z_2 attestiert wird. Der Prüfer wird nun so lange Zertifikate einholen, bis er zu einem Zertifikat eines Ausstellers gelangt, dem er vertraut und dessen authentischen öffentlichen Schlüssel er bereits kennt.
Durch dieses Fortschreiten anhand der Zertifikataussteller ergibt sich eine Zertifizierungskette, an deren Anfang ein Zertifikat eines vertrauenswürdigen Ausstellers und an deren Ende das Zertifikat mit der Zuordnung $A \rightarrow pk_A$ steht.

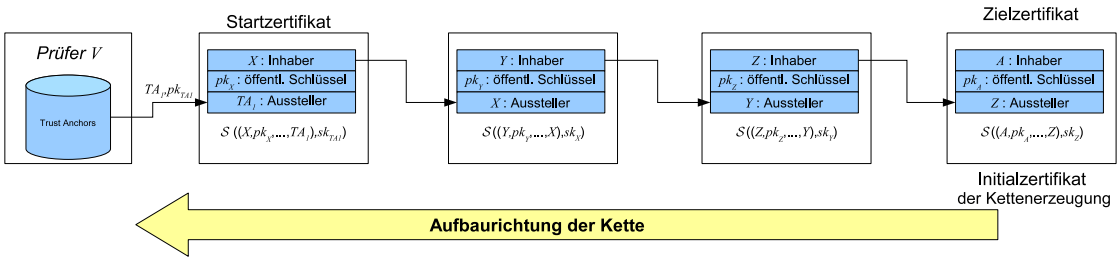


Abbildung 3.5: Beispiel: Zertifizierungskette von Schlüsselzertifikaten

Definition 3.8 ***Zertifizierungskette (Schlüsselzertifikate):***
Eine Zertifizierungskette von Schlüsselzertifikaten ist eine Menge von Zertifikaten, in welchen öffentliche Schlüssel beglaubigt werden. Die Zertifikate sind so verkettet, dass der Aussteller des i-ten Zertifikats der Inhaber des $(i-1)$-ten Zertifikats in der Kette ist.
Das letzte Zertifikat der Kette wird als Zielzertifikat bezeichner. Das erste ist von einem Aussteller signiert, dessen authentischen öffentlichen Schlüssel der Prüfer bereits vor Erzeugung der Kette kennt. Ein solcher Aussteller wird Trust Anchor (TA, Vertrauensanker) genannt.

Ein Prüfer kann die Trust Anchors und ihre authentischen öffentlichen Schlüssel in Form von selbst signierten Zertifikaten vorhalten. In diesem Fall wird bei jeder Kettenerstellung das entsprechende Trust-Anchor-Zertifikat selbst der Kette hinzugefügt und stellt dann das Startzertifikat dar (siehe Abb. 3.6).

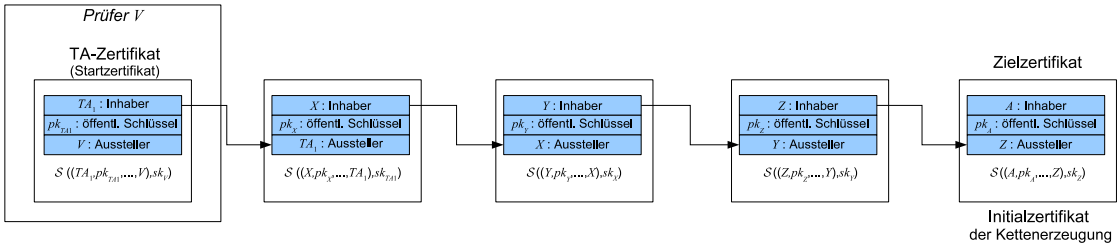


Abbildung 3.6: Beispiel: Zertifizierungskette von Schlüsselzertifikaten mit TA-Zertifikat als Startzertifikat

Die Erzeugung einer Zertifizierungskette beginnt immer mit dem Zielzertifikat: dieses liegt dem Prüfer zuerst vor. Er baut die Kette dann rückwärts auf, bis er zum Startzertifikat gelangt. Aufgrund dieser Sichtweise wird das Zielzertifikat auch als Initialzertifikat bezeichnet, da es den Initialpunkt der Kettenerstellung repräsentiert.

Für die Authentizität eines öffentlichen Schlüssels müssen drei Bedingungen erfüllt sein:

1. Es gibt eine lückenlose Zertifizierungskette vom Prüfer bis zur geprüften Entität.
2. Der Prüfer vertraut jedem Zertifikataussteller innerhalb der Zertifizierungskette. Dieses Vertrauen kann implizit oder explizit modelliert werden (s.u.).
3. Alle Zertifikate der Kette sind zum aktuellen Zeitpunkt noch gültig (siehe 3.9.2)

Definition 3.9 ***Public-Key-Infrastruktur (PKI)***
Eine Public-Key-Infrastruktur ist ein System aus Software, Entitäten, Prozessen, Protokollen, Methoden und Regeln zur Verwaltung von digitalen Schlüsselzertifikaten. Bestandteil der PKI ist auch das Regelwerk für die Prüfung der Zertifikate und die dafür notwendige Herleitung von Zertifizierungsketten mit dem Ziel der Ermittlung authentischer öffentlicher Schlüssel. Die PKI liefert zudem Methoden zur Erzeugung von Statusnachrichten, um die aktuelle Gültigkeit von Zertifikaten zu determinieren.

Die Hauptentitäten der PKI sind neben den Prüfern und den zertifizierten Entitäten (als End-Entitäten bezeichnet, siehe Abschnitt 3.5) die Certificate Authorities (Aussteller) und die Certificate Revocation bzw. Certificate Status Authorities, welche bei Bedarf den Rückruf von Zertifikaten realisieren oder deren Gültigkeit bestätigen (siehe Abschnitte 3.5 und 3.9).

Maurer führt in [Mau96] eine abstrakte Notation für die Daten einer PKI ein. Er unterscheidet:

- *Schlüsselzertifikate (Public Key Certificates):* Zertifikate, in denen ein Aussteller X einem Inhaber Y den öffentlichen Schlüssel pk_Y mittels seiner digitalen Unterschrift zertifiziert. Das Zertifikat ist mit sk_X signiert und daher mit dem öffentlichen Schlüssel pk_X prüfbar. Schlüsselzertifikate werden wie folgt geschrieben:

$$Cert(X, pk_X, Y, pk_Y)$$

- *Empfehlungen (Recommendations):* Zertifikate, in denen ein Aussteller X einem Inhaber Y Vertrauen einer Stufe i zertifiziert. Stufe 1 bedeutet: X bescheinigt sein Vertrauen, dass Y korrekte Schlüsselzertifikate ausstellt. Höhere Vertrauensstufen bedeuten, dass Y dieses Vertrauen an andere Entitäten weitergeben darf (Delegation). Empfehlungen sind ebenfalls digital von X signiert und können mit pk_X geprüft werden. Man schreibt:

$$Rec(X, pk_X, Y, i)$$

Maurer propagiert damit also die explizite Modellierung von Vertrauen. Eine Empfehlung ist dabei ein spezielles Privilegzertifikat wie im Folgenden Abschnitt 3.4.3 in Definition 3.11 beschrieben.

3.4.3 Privilege-Management-Infrastrukturen

Steht die Authentizität des Kommunikationspartners fest, so kann ein Autorisierungsverfahren durchgeführt werden. In diesem wird bestimmt, ob eine Entität über die Berechtigung zur Durchführung bestimmter Aktionen verfügt.
Einem solchen Verfahren liegt wie schon bei der Authentifizierung der Einsatz von Zertifikaten zugrunde, welche unter dem Oberbegriff *Attributzertifikate* zusammengefasst werden. Darunter fallen Deskriptive Attributzertifikate sowie Privilegzertifikate:

Definition 3.10 ***Deskriptives Attributzertifikat***
In einem Deskriptiven Attributzertifikat wird einer Entität ein Wert für ein bestimmtes beschreibendes Attribut, d.h. eine reale oder digitale Eigenschaft, ein Merkmal, Wissensmerkmal oder digitales Besitztum zugeordnet. Die Zertifizierungsstelle beglaubigt mit ihrer digitalen Signatur, dass sie von der Authentizität dieser Zuordnung überzeugt ist.

Ein Deskriptives Attributzertifikat kann dem Inhaber beispielsweise eine bestimmte Augenfarbe, ein biometrisches Referenzmuster oder eine Abteilungs- oder Rollenzugehörigkeit innerhalb eines Unternehmens zuordnen.

Definition 3.11 ***Privilegzertifikat***
In einem Privilegzertifikat wird einer Entität ein Privileg, d.h. eine Berechtigung zur Durchführung einer spezifizierten Aktion, mit einer Delegationsstufe zugeordnet. Diese Delegationsstufe $i \in \mathbb{N}$ gibt an, über wie viele Stufen der Inhaber dieses Privileg delegieren darf. Die Zertifizierungsstelle beglaubigt mit ihrer digitalen Signatur, dass sie von der Authentizität dieser Zuordnung überzeugt ist.

Ein Prüfer V wird nun genau dann eine Entität A zur Durchführung einer bestimmten Aktion autorisieren, wenn ihm ein Privilegzertifikat vorliegt, in welchem A das notwendige Privileg bescheinigt ist. Deskriptive Attributzertifikate können dabei unterstützend herangezogen werden, wie das folgende Beispiel illustriert:
Die Mitarbeiter der Abteilung Y einer Firma haben Zugriff auf ein spezielles Verzeichnis eines Firmenservers. Es gibt mehrere Möglichkeiten, diesen Sachverhalt in Form von Zertifikaten zu modellieren:

1. *Ausschließliche Verwendung von Privilegzertifikaten:* Für jeden Mitarbeiter der Abteilung Y wird ein Privilegzertifikat für den Privilegtyp „Zugriff auf Verzeichnis z“ erzeugt.

2. *Kombination von Privileg- und Deskriptiven Attributzertifikaten:* Für jeden Mitarbeiter von Abteilung Y wird ein Deskriptives Attributzertifikat erzeugt, in welchem die Abteilungszugehörigkeit zertifiziert wird. Weiterhin wird ein einziges Privilegzertifikat generiert, in welchem für den Inhaber „Mitarbeiter von Abteilung Y“ das Privileg „Zugriff auf Verzeichnis z“ bestätigt wird.

Die Kombination von Privileg- und Deskriptiven Attributzertifikaten erlaubt also z.B. die Implementierung von rollenbasierten Zugriffskontrollsystemen (RBAC) [ZhM04] [Woe06].

Angelehnt an die PKI-Notation von Maurer werden die beschriebenen Zertifikate wie folgt notiert:

- *Deskriptive Attributzertifikate:* Zertifikate, in denen ein Aussteller X einem Inhaber Y das Attribut vom Typ at_Y mit Attributwert av_Y mittels seiner digitalen Unterschrift zertifiziert. Das Zertifikat ist mit sk_X signiert und daher mit dem öffentlichen Schlüssel pk_X überprüfbar. Man schreibt:

$$Cert(X, pk_X, Y, at_Y, av_Y)$$

- *Privilegzertifikate:* Zertifikate, in denen ein Aussteller X einem Inhaber Y das Privileg vom Typ pt_Y der Delegationsstufe pd_Y zertifiziert:

$$Cert(X, pk_X, Y, pt_Y, pd_Y).$$

Um ein Privilegzertifikat zu verifizieren, sind durch V folgende Punkte zu prüfen:

1. *Ist das Zertifikat von einem vertrauenswürdigen, authentischen Aussteller signiert?* Hierfür muss eine Kette von Schlüsselzertifikaten erzeugt werden. Ihr Zielzertifikat ist für den Erzeuger des vorliegenden Privilegzertifikats ausgestellt.

2. *Hat der Aussteller das Recht zur Ausstellung dieses Privilegzertifikats?* Zur Prüfung dieses Punktes muss eine Kette von Privilegzertifikaten aufgestellt werden. Definition 3.12 beschreibt die Zusammensetzung einer solchen Kette.

3. *Sind alle verwendeten Zertifikate zum aktuellen Zeitpunkt gültig?*

Definition 3.12 ***Zertifizierungskette (Privilegzertifikate):***
Eine Zertifizierungskette von Privilegzertifikaten ist eine Menge von Zertifikaten, in welchen ein Privileg des Typs pt_Y beglaubigt wird. Die Zertifikate sind so verkettet, dass der Aussteller des i-ten Zertifikats der Inhaber des $(i-1)$-ten Zertifikats in der Kette darstellt. Die Zertifikate variieren nur in ihrer Delegationsstufe, nicht aber in ihrem Typ. Dabei gilt für jede von einem Aussteller X zertifizierte Entität Y:

$$pd_X > pd_Y$$

Aussteller des ersten Zertifikats in der Kette ist eine Entität, die vom Prüfer als Source of Authority (SoA) anerkannt wird. Die SoA verfügt also seiner Überzeugung nach über die unbeschränkte Berechtigung zur Zertifizierung und Delegation bestimmter Privilegien (und damit über eine „unendliche“ Delegationsstufe der entsprechenden Privilegtypen).

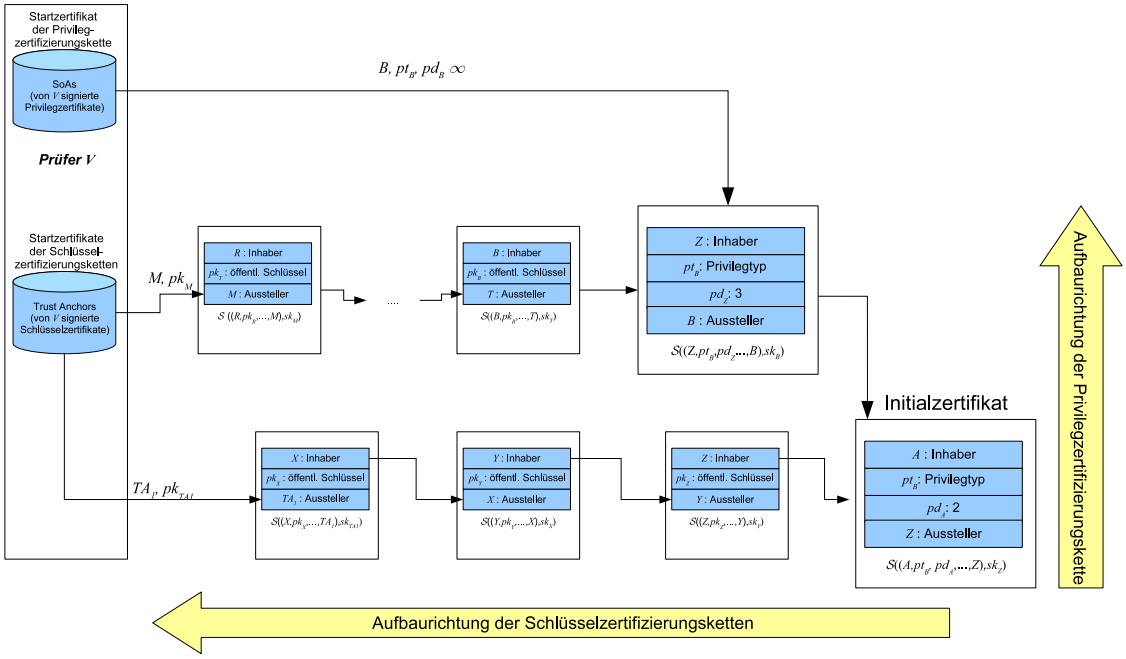


Abbildung 3.7: Beispiel: Zertifizierungsketten für die Validierung von Privilegzertifikaten

Die Source of Authority kann beispielsweise der Inhaber eines Bankkontos sein, der dann Privilegien, welche dieses Konto betreffen, frei delegieren kann, z.B. Privilegien zur Überweisung, Kartenzahlung oder Erstellung von Daueraufträgen. Ein Prüfer V kann die SoAs in Form von selbst signierten Zertifikaten vorhalten (hier für SoA I): $Cert(V, pk_V, I, pt_I, pd_I)$. Ist dies der Fall, beginnen alle gültigen Ketten mit einem von V ausgestellten Zertifikat für diejenige SoA, welche Aussteller für das folgende Privilegzertifikat ist.

Abbildung 3.7 zeigt exemplarische Zertifizierungsketten für die Validierung eines Privilegzertifikats. Die Berechtigung von Z zur Weitergabe des Privilegs pt_B wird von B zertifiziert, B selbst ist V als SoA für pt_B bekannt. Für beide Privilegzertifikate existiert jeweils eine Zertifizierungskette von Schlüsselzertifikaten, um die Signaturprüfung zu ermöglichen.

Um ein Deskriptives Attributzertifikat zu verifizieren, muss der Prüfer V die folgenden Fragen beantworten:

1. *Ist das Zertifikat von einer vertrauenswürdigen Entität ausgestellt und signiert?* Hierfür muss eine Zertifizierungskette von Schlüsselzertifikaten erzeugt werden.

2. *Verfügt der Aussteller des Zertifikats über die Berechtigung, das betreffende Deskriptive Attribut zu zertifizieren?* Diese Berechtigung liegt dann vor, wenn der Aussteller ein Privilegzertifikat für die Ausstellung Deskriptiver Attributzertifikate des entsprechenden Attributtyps vorweisen kann und für dieses eine Zertifizierungskette gemäß Definition 3.12 herleitbar ist.

3. *Sind alle verwendeten Zertifikate zum aktuellen Zeitpunkt gültig?*

Abbildung 3.8 zeigt eine exemplarische Zertifizierungskette für die Validierung eines Deskriptiven Attributzertifikats. Der Aussteller Z ist durch ein Privilegzertifikat mit dem Privileg „Ausstellung von Zertifikaten für das Attribut at_A“ (hier pt(Zertifiz. at_A)) zur Ausstellung berechtigt, wobei dieses Privileg unmittelbar von der SoA B kommt, welche dem Prüfer bekannt ist.

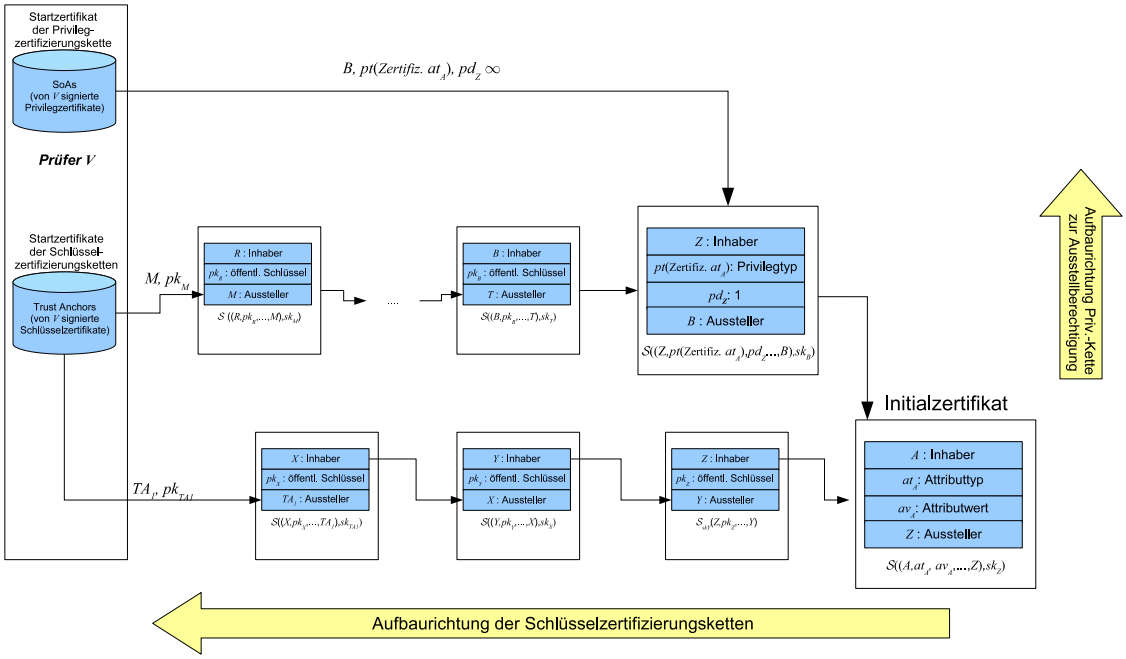


Abbildung 3.8: Beispiel: Zertifizierungsketten für die Validierung von Deskriptiven Attributzertifikaten

Definition 3.13 ***Privilege-Management-Infrastruktur (PMI)***
Eine Privilege-Management-Infrastruktur ist ein System aus Software, Entitäten, Prozessen, Protokollen, Methoden und Regeln zur Verwaltung von digitalen Deskriptiven Attribut- und Privilegzertifikaten. Eine PMI enthält auch das Regelwerk für die Prüfung von Zertifikaten und die Erstellung von Zertifizierungsketten mit dem Ziel, authentische Deskriptive Attribute und Privilegien herzuleiten. Weiterhin werden Methoden für die Erstellung von Statusnachrichten definiert.

Die Akteure der PMI sind neben den zertifizierten Entitäten und den Prüfern die Attribute Authorities, welche Privileg- und Deskriptive Attributzertifikate ausgeben, sowie Attribute Revocation bzw. Attribute Status Authorities.

3.5 Handelnde Entitäten einer AAI

Im Folgenden werden nun die Entitäten und Aktivitäten von PKI und PMI kombiniert, um eine AAI im Ganzen zu beschreiben.

Es existieren die folgenden Klassen von Entitäten:

- *End-Entität (EE):* Entität, welche Authentifizierungs- und Autorisierungsverfahren mit Prüfern durchführt und hierfür Zertifikate für bestimmte Privilegien, Attribute und digitale Objekte, insbesondere seinen öffentlichen Schlüssel, benötigt. Für eine EE werden Schlüssel-, Privileg- und Deskriptive Attributzertifikate ausgestellt. Sie heißt dann *Inhaber* eines Zertifikats.

- *Issuing Authority (IA):* Stellt digital signierte Schlüssel-, Privileg- und Deskriptive Attributzertifikate für End-Entitäten aus. Diese werden daraufhin in das Datenverzeichnis der AAI aufgenommen.
  Die Issuing Authority ist die Generalisierung der Certificate Authority der PKI und der Attribute Authority der PMI.

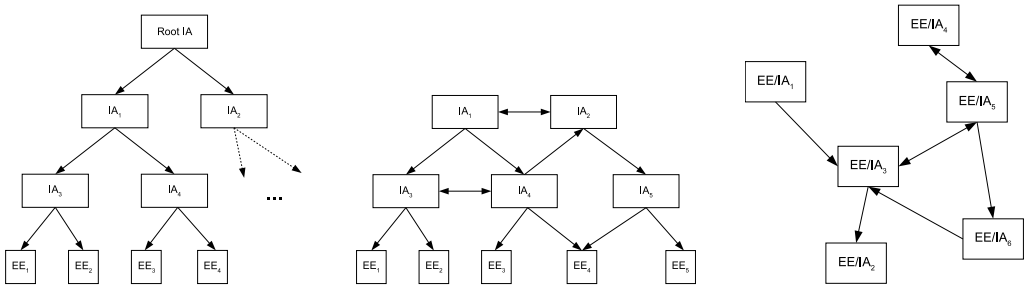
- *Status Authority (SA):* Gibt Auskunft über Gültigkeit oder Ungültigkeit von Zertifikaten zu einem bestimmten Zeitpunkt. Dafür stellt sie Statusnachrichten (Rückrufe oder Gültigkeitsbestätigungen, siehe 3.9) aus, welche als Teil der Datenbasis verfügbar gemacht werden. Sie deckt damit die Entitäten Certificate Revocation / Status Authority einer PKI und Attribute Revocation/Status Authority einer PMI ab.
  In Abhängigkeit vom Rückrufmodell kann jede SA eine separate Entität oder aber in eine IA oder eine EE integriert sein.

- *Prüfer (V):* Führt Authentifizierungs- und Autorisierungsverfahren mit End-Entitäten durch. Der Prüfer bildet aus den im Verzeichnis der AAI vorhandenen Zertifikaten und Statusnachrichten nach einer Herleitungslogik Zertifizierungsketten, um auf dieser Basis eine eindeutige Aussage über Authentizität und Befugnisse einer End-Entität zu treffen.

3.6 AAI-Architekturen

Es existieren verschiedene Möglichkeiten, die Beziehungen zwischen End-Entitäten und Issuing Authorities zu modellieren. Die drei wesentlichen Architekturen werden hier vorgestellt und abgegrenzt.

Hierarchische Architektur. In einer AAI mit hierarchischer Architektur sind die Issuing Authorities in mehreren Schichten vernetzt. Alle Berechtigungen gehen ursprünglich von einer einzigen IA, der *Root IA*, aus und werden auf IAs der nächsten Stufe delegiert. Somit entsteht eine Baumstruktur, deren Blätter die zertifizierten End-Entitäten darstellen (siehe Abb. 3.9(a)). Die Root IA ist dann sowohl Trust Anchor für alle Prüfer als auch Source of Authority für die vorhandenen Zertifikate.



(a) Hierarchische Architektur (b) Hybride Architektur (c) Dezentralisierte Architektur

Abbildung 3.9: AAI-Architekturen

Hybride Architektur. Die hybride Architektur ist eng mit der hierarchischen Architektur verwandt. Allerdings ist die Existenz einer Root IA nicht notwendig: durch Cross-Zertifizierung, d.h. Zertifizierung zwischen Issuing Authorities, werden über eine reine Baumstruktur hinausgehende, auch ebenenübergreifende Verbindungen aufgebaut (siehe 3.9(b)). Es existieren dann verschiedene Trust Anchors und Sources of Authority für die unterschiedlichen Privilegtypen.

Dezentralisierte Architektur. In einer dezentralisierten AAI-Architektur zertifizieren sich die End-Entitäten gegenseitig, d.h. die starre Trennung zwischen IA und EE ist aufgehoben. Das Modell hat keine privilegierten Entitäten (siehe Abb. 3.9(c)), jede Entität kann aus Sicht eines Prüfers prinzipiell Trust Anchor oder Source of Authority sein. Es ist einsichtig, dass in dieser AAI-Architektur die Herleitung von Zertifizierungsketten am aufwendigsten ist, da keine global gültigen TAs oder allgemein bekannten SoAs existieren.

3.7 Erweiterte Zertifikatnotation für AAI

Wölfl [Woe06] erweitert die PKI-Notation von Maurer um Attributzertifikate. Weiterhin hebt er die bislang herrschende Unterscheidung zwischen Deskriptiven Attribut- und Schlüsselzertifikaten auf.

Repräsentation von Attributen:
Deskriptive Attribute werden mit *prop* (engl. property: Eigenschaft), Privilegien mit *priv* bezeichnet.

1. Deskriptive Attribute $prop(T, V)$ besitzen einen Attributtyp T und einen Attributwert V.

2. Privilegien $priv(T, i)$ haben den Privilegtyp T und die Delegationsstufe i.

Weiterhin werden die folgenden speziellen **Attributtypen** eingeführt:

1. *Öffentlicher Schlüssel*: Deskriptiver Attributtyp *pk*

2. *Recht zur Ausstellung von Zertifikaten* (Certificate Issuing) vom Deskriptiven Attributtyp T: Privilegtyp $ci(T)$

3. *Recht zum Rückruf von Zertifikaten* (Certificate Revocation) für Zertifikate, die von Entität X ausgestellt wurden: Privilegtyp $cr(X)$

Zertifikate werden demnach wie folgt notiert:

1. *Schlüsselzertifikate*: $Cert(X, pk_X, Y, prop(pk, pk_Y))$ mit Aussteller X mit dem öffentlichen Schlüssel pk_X, der End-Entität Y und deren öffentlichem Schlüssel pk_Y. Das Zertifikat ist nun also ein Attributzertifikat mit Deskriptivem Attributtyp pk.

2. *Deskriptive Attributzertifikate*: $Cert(X, pk_X, Y, prop(T, av_Y))$. In einem Deskriptiven Attributzertifikat bescheinigt der Aussteller X der End-Entität Y den Wert av_Y für das Attribut vom Typ T.

3. *Privilegzertifikate*: $Cert(X, pk_X, Y, priv(T, pd_Y))$ mit Privilegtyp T und Delegationsstufe $pd_Y \geq 1$.

4. *Empfehlungen*: Eine Empfehlung einer Entität A für eine Entität B ist ein Zertifikat $Cert(A, pk_A, B, priv(ci(pk), pd_B))$.

3.8 Rollenbasierte Sicht auf die AAI und Abstraktion

Da sich die Dissertation auf Fragen der Datenverwaltung innerhalb einer AAI konzentriert, ist eine hierfür geeignete Sicht auf eine zertifikatbasierte AAI zu wählen. In diesem Abschnitt wird daher der rollenbasierte Fokus der weiteren Betrachtung und notwendige Abstraktionen spezifischer Teile einer AAI präsentiert.

Rollenbasierte Entitätsdarstellung. Jede der in Abschnitt 3.5 präsentierten Entitäten wird nur einfach betrachtet, nämlich als abstrakte Rolle, welche von verschiedenen konkreten Entitäten eingenommen werden kann. Die Rolle *Issuing Authority* repräsentiert beispielsweise alle in der AAI existierenden und aktiven IAs. Der logische Aufbau der AAI, d.h. Hierarchien und Beziehungen zwischen verschiedenen Authorities, wird somit abstrahiert, um die Wahl der AAI-Architektur nicht einzuschränken.

Zertifizierungspolitik. Eine Politik legt fest, nach welchen Kriterien Zertifikate ausgegeben oder zurückgenommen werden. Ein Beispiel für eine konkrete Politik: Schlüsselzertifikate werden nur dann ausgegeben, wenn die Issuing Authority den Personalausweis eines Benutzers gesehen hat, der dessen Identität beweist, und er seinen öffentlichen Schlüssel handschriftlich signiert bei der IA bestätigt hat (letzteres ist eine gängige Policy im Rahmen von HBCI (Home Banking)).
Im Modell ist die Sicht auf die Zertifikate aber abstrakt, d.h. weder die konkreten Inhalte noch die Häufigkeit der Zertifikaterstellung oder die Wahrscheinlichkeit, dass für eine End-Entität ein Zertifikat ausgegeben wird, sind relevant, und somit ist die Politik nicht zu berücksichtigen.
Analog werden Status-Policies ausgeblendet: nicht die Kriterien oder Wahrscheinlichkeiten für die Ausstellung von Rückrufen und Gültigkeitsbestätigungen werden betrachtet,

sondern nur, *dass* jedes existierende Zertifikat vor Ende seiner Gültigkeitsdauer ungültig werden und zurückgerufen werden kann.

Herleitungslogik. Die Erzeugung von Zertifizierungsketten findet ausschließlich beim Prüfer statt. Die Regeln operieren auf einer Teilmenge aller in der AAI vorhandenen Zertifikate und stellen damit ihr inhaltliches Herzstück dar. Sie unterscheiden sich abhängig von der Architektur der AAI und den konkreten Beziehungen zwischen beteiligten Entitäten. Daher werden die Herleitungsregeln abstrahiert und vereinfacht betrachtet, weil keine Festlegung auf eine spezielle Architektur erfolgen soll. Sind die Regeln außerdem konsistent und stimmig - wovon ausgegangen wird - so liefern sie korrekte Ergebnisse für die vorhandenen Eingabedaten.
Daher muss sichergestellt werden, dass den Prüfern alle notwendigen Daten zur Verfügung stehen. Im Folgenden wird davon ausgegangen, dass hierfür ein öffentliches, verteiltes P2P-Verzeichnis existiert, wobei die Konkretisierung seiner Eigenschaften später erfolgt (siehe Kapitel 5).

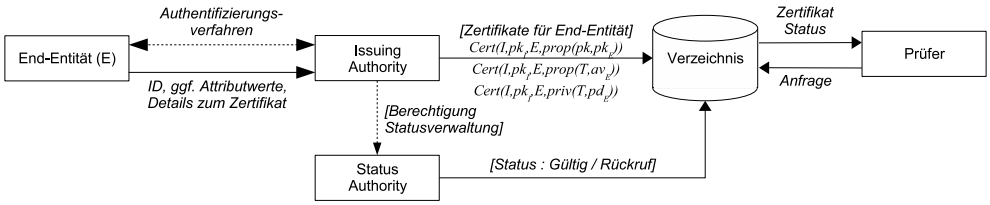


Abbildung 3.10: Rollenbasierte AAI

Abbildung 3.10 zeigt eine AAI gemäß der zuvor definierten Eigenschaften und Abstraktionen. Auf die bisher nur kurz erwähnte Behandlung des Zertifikatstatus wird direkt im Anschluss (Abschnitt 3.9) im Detail eingegangen.

3.9 Rückruftechniken und Statusnachrichten

3.9.1 Grundlegende Problemstellung

In einer AAI ist es notwendig, eine Möglichkeit zum Rückruf von Zertifikaten vor Ende ihrer ursprünglichen Gültigkeitsdauer zu schaffen. Es sind vielfältige Gründe für einen Rückruf denkbar. Eine Auswahl ist in Tabelle 3.1 gegenübergestellt.
Unterschieden werden dabei EE-zentrierte Gründe, d.h. solche, bei denen die End-Entität den Rückruf initiiert, und IA-zentrierte Gründe, bei denen der Zertifikataussteller selbst das Zertifikat vorzeitig als ungültig deklariert.

3.9.2 Rückruftechniken für AAI

Für die Rückrufproblematik in Public-Key-Infrastrukturen existiert eine Vielzahl von Lösungsansätzen, die im Folgenden präsentiert werden. Da die Konzepte unabhängig vom Zertifikattyp sind, können sie auch für andere Deskriptive Attribut- und Privileg-

Zertifikattyp	**EE-zentriert**	**IA-zentriert**
Schlüsselzertifikat	Verlust, Vergessen oder Kompromittierung des privaten Schlüssels	
Sonstiges Deskriptives Attributzertifikat	Verlust des zertifizierten Merkmals *Beispiel:* freiwillige Änderung eines Benutzernamens	Entzug eines von der IA zugewiesenen Merkmals *Beispiel:* Abteilungszugehörigkeit in einer Firma
Privilegzertifikat		Entzug eines Privilegs o. der Delegationsberechtigung *Beispiel:* Unterschriftsberechtigung für Verträge bei Vertrauensmissbrauch

Tabelle 3.1: Exemplarische Rückrufgründe für Zertifikate

zertifikate und somit für AAI allgemein ohne weiteres eingesetzt werden. Die vorgestellten Ansätze werden in drei große Gruppen unterschieden:

1. *Online-Systeme*
   In einem Online-System wird die Prüfung, ob ein bestimmtes Zertifikat zurückgerufen wurde oder noch gültig ist, im direkten Dialog zwischen dem Prüfer und dem Zertifikataussteller (IA) geprüft. Dafür wird ein definiertes Kommunikationsprotokoll verwendet. Alternativ kann der Dialog auch mit einem beauftragten Vertreter (Responder) der IA durchgeführt werden.

2. *Offline-Systeme*
   In einem Offline-Rückrufsystem müssen die IAs nicht ständig kontaktierbar sein. Stattdessen veröffentlicht jede Issuing Authority in regelmäßigen oder unregelmäßigen Abständen Rückrufinformationen zu von ihr ausgestellten Zertifikaten. Diese werden digital signiert veröffentlicht. Will ein Prüfer wissen, ob ein bestimmtes Zertifikat zurückgerufen wurde, lädt er den aktuellen Datensatz aus dem Verzeichnis der AAI und sucht dort nach Informationen über das Zertifikat. Ist kein solcher Eintrag vorhanden, sieht er es als gültig an. Die Integrität jeder Rückrufinformation ist durch eine IA-Signatur gesichert.

3. *Rückruffreie Systeme*
   Hier werden die Zertifikate der AAI so organisiert und gespeichert, dass die Erstellung von Rückrufen nicht notwendig ist. So werden beispielsweise nur die noch gültigen Zertifikate in einem vertrauenswürdigen Verzeichnis (Trusted Directory) gespeichert oder in kurzen Abständen Gültigkeitsbestätigungen publiziert. Rückruffreie Systeme basieren also im Gegensatz zu Offline-Systemen auf *positiven* Aussagen über die Gültigkeit eines Zertifikats.

In den folgenden Abschnitten werden verschiedene konkrete Techniken aus den drei Kategorien vorgestellt. Am Ende jedes Abschnitts folgt eine allgemeine, nicht auf eine spezielle Anwendung bezogene Bewertung der Kategorie und ggf. einzelner Techniken.

3.9.2.1 Online-Techniken

- *OCSP* (Online Certificate Status Protocol)
  OCSP [MAM99] regelt die Interaktion zwischen einem Prüfer und einem Responder, welcher möglichst aktuelle[4] Informationen über den Status der Zertifikate einer oder mehrerer IAs verwaltet. Der Responder übermittelt also eine digital signierte Nachricht, die den Status eines Zertifikats (*expired*, *current* (gegenwärtig gültig) oder *unknown*) zum Zeitpunkt der Kommunikation mit dem Prüfer beinhaltet. Ist der Status *unknown*, wird dem Prüfer empfohlen, die IA direkt zu kontaktieren oder das Zertifikat sicherheitshalber als ungültig anzusehen.

- *Statusnachrichten in SDSI/SPKI*
  Die Infrastruktur kann um die Entität *Suicide Bureau* (SB) erweitert werden [Riv98]. Die öffentlichen Schlüssel der End-Entitäten sind dann bei verschiedenen SBs registriert. Zum Ausstellungszeitpunkt eines Zertifikats signiert der Inhaber eine *Suicide Note* mit seinem privaten Schlüssel. Will er das Zertifikat zurückrufen, übermittelt er diese an ein Suicide Bureau. Das SB gibt sie nach erfolgreicher Signaturprüfung an weitere SBs weiter, so dass der geänderte Zertifikatstatus nach und nach bekannt gemacht wird. SBs können außerdem Positivnachrichten, *Certificates of Health*, ausstellen. Mit einem solchen Zertifikat bestätigt ein SB, dass *nach seinem Wissen* das betreffende Zertifikat zum aktuellen Zeitpunkt noch gültig ist. Da ein SB aber mit einer gewissen Wahrscheinlichkeit nicht über alle Suicide Notes informiert wird, muss diese Angabe nicht notwendigerweise richtig sein.

Bewertung
Vorteile von Online-Systemen:

- *Aktualität:* Die Antwort gibt im Idealfall den Zertifikatstatus exakt zum Anfragezeitpunkt wieder, ist also verzögerungsfrei.

- *Dialog:* Die Information über den Status wird unmittelbar zwischen zwei Entitäten, ggf. unter Einsatz von Mechanismen zur Verbindungsverschlüsselung, ausgetauscht. Dadurch ist das Risiko gering, dass eine andere Entität diese Nachricht manipuliert.

Nachteile:

- *Kosten für die IA:* Online-Techniken verlangen eine ständige Verfügbarkeit jeder IA bzw. des beauftragten Responders. Dadurch entstehen hohe Kosten für den Betrieb von Servern, welche diesen Verfügbarkeitsanforderungen entsprechen.

- *Einschränkung der Software-Architektur:* Bedingt durch den ersten Punkt kann nur eine Client-Server-Architektur zum Einsatz kommen.

- *Einschränkung der AAI-Architektur:* In einer dezentralisierten AAI zertifizieren sich teilnehmende Entitäten, häufig Privatpersonen mit PC-Systemen, wechselsei-

[4] Im Idealfall stehen dem Responder Echtzeitinformationen zur Verfügung, beispielsweise durch Zugriff auf die internen Daten der verwalteten IA(s). Kann dies nicht erreicht werden, muss er auf weniger aktuelle Informationen, beispielsweise aus Certificate Revocation Lists wie in Abschnitt 3.9.2.2 vorgestellt, zurückgreifen.

tig. Hier ist ein Online-System nicht einsetzbar, da die Verfügbarkeitsanforderungen auf beiden Seiten nicht einhaltbar sind.

- *Paradigmenbruch:* Die Realisierung eines Statussystems auf Basis von Online-Informationen steht im Widerspruch zu den sonst in einer zertifikatbasierten AAI bewusst verwendeten Offline-Informationen.

3.9.2.2 Offline-Techniken

- *CRL (Certificate Revocation List)*
  Eine CRL oder Rückrufliste [HPF02] [HoP01] ist ein Dokument, in welchem eine IA einen eindeutigen Bezeichner, z.B. eine Seriennummer, von ungültig gewordenen Zertifikaten erfasst und digital signiert. CRLs werden in regelmäßigen Abständen von einer IA herausgegeben. Varianten sind die Delta CRL, welche nur die seit dem letzten Veröffentlichungszeitpunkt zurückgerufenen Zertifikate enthält, und die verteilte CRL, bei der die gesamte Liste in mehrere separat signierte Teile aufgespalten wird. Es ist dann beispielsweise möglich, den Segmenten der CRL verschiedene Aktualisierungszeiträume zuzuweisen.

- *CRT (Certificate Revocation Tree)*
  CRT ist ein Rückrufsystem [Koc98] auf Basis binärer Hashbäume [Mer79]. Jedes Blatt repräsentiert ein zurückgerufenes Zertifikat in Form von dessen Seriennummer. Jeder Elternknoten speichert den Hashwert über seine Kindknoten (ähnlich wie in CVT, siehe 3.9.2.3), die Wurzel des Baumes wird von der IA signiert.
  Wurde ein Zertifikat zurückgerufen, so existiert einen Pfad von dem entsprechenden Blatt zur Wurzel des Baums. Dieser Pfad muss einem Prüfer einschließlich der Hashwerte benachbarter Teilbäume, welche an den Pfadelementen mit aufgehangen sind, bekannt und zudem erfolgreich validiert sein, damit der Rückruf anerkannt wird. Außerdem verifiziert der Prüfer die Signatur der Wurzel.
  Wesentlicher Vorteil gegenüber CRLs ist, dass dem Prüfer nicht der gesamte Baum bekannt sein muss, um einen Rückruf zu prüfen. Außerdem kann eine End-Entität mit Hilfe eines Teils des Baumes belegen, dass ein spezifisches Zertifikat noch nicht zurückgerufen wurde: nämlich genau dann, wenn im CRT zwei zurückgerufene Zertifikate als Blätter nebeneinander liegen, wobei die Seriennummer des ersten kleiner, die des zweiten größer als die des betreffenden Zertifikats ist.
  Wird ein neuer Rückruf hinzugefügt, ändert sich unter Umständen die ganze Baumstruktur, so dass der CRT neu aufgebaut und signiert werden muss. Um diese vollständige Neuberechnung zu verhindern, wurde 23CRT [NaN98] entwickelt. Es entspricht dem Certificate Revocation Tree mit dem Unterschied, dass statt binären Hashbäumen 2-3-Bäume eingesetzt werden. Dadurch ist bei Hinzufügung eines Rückrufes immer nur ein Teil des Baumes umzustrukturieren.

Bewertung
Vorteile von Offline-Systemen:

- *Geringer Kommunikationsaufwand für die IA:* Die IA hat geringere Kommunikationskosten und niedrigeren Aufwand als in einem Online-System, da Aktualisie-

rungen der Rückrufinformationen nur periodisch durchgeführt werden. Die IA stellt nur dann Nachrichten aus, wenn sich der Status von Zertifikaten ändert.

- *Geringe Verfügbarkeitsanforderungen für die IA:* Wegen der periodischen Ausstellung von Nachrichten sind die Anforderungen an die Verfügbarkeit der IAs gering. Ein permanent verfügbarer Datenspeicher ist zwar notwendig, kann aber IA-übergreifend und -unabhängig realisiert werden, wodurch die Gesamtkosten sinken.

- *Unabhängigkeit der Software-Architektur:* Da Offline-Techniken nachrichtenorientiert und nicht an ein spezielles Dialogprotokoll gebunden sind, sind verschiedene Software-Architekturen zur Bereitstellung der Rückrufinformationen möglich, beispielsweise eine Client-Server-Architektur mit einem zentralen Datenbankserver oder eine vollständig dezentrale Architektur mit verteilter Datenverwaltung.

- *Paradigma-Konsistenz:* Die Verwendung von Offline-Informationen als Statusnachrichten ist konsistent mit der Verwendung von Zertifikaten, die ebenfalls Offline-Informationen darstellen.

Nachteile:

- *Verzögerung:* Da die Rückrufinformationen nicht ständig aktualisiert, sondern nur periodisch publiziert werden, wird jede Statusänderung mit einer gewissen Verzögerung bekannt. Zwischen dem Zeitpunkt des tatsächlichen Ungültigwerdens eines Zertifikats und der Rückrufpublikation wird das Zertifikat von jedem Prüfer als gültig angesehen.

- *Fehlen von Positivnachrichten:* Insbesondere bei CRLs hat ein Prüfer keine Möglichkeit, eine positive Bestätigung einzuholen, die belegt, dass ein Zertifikat tatsächlich zum aktuellen Zeitpunkt noch gültig ist, sondern kann diesen Umstand nur aus dem Fehlen eines Rückrufs folgern. Durch unvollständige Information über die publizierten Rückrufnachrichten kann der Fall eintreten, dass ein Prüfer falsche Annahmen über den Status bestimmter Zertifikate trifft.

- *Overhead für Prüfer:* Will ein Prüfer die Gültigkeit eines einzelnen Zertifikats prüfen, erhält er stets zusätzliche, für ihn uninteressante Daten. Dadurch entsteht ein Overhead hinsichtlich des Kommunikations- und Prüfungsaufwandes.

3.9.2.3 Rückruffreie Systeme

- *Trusted Directory*
  Das Trusted Directory [Gut03] ist das einfachste denkbare rückruffreie System. Hierbei existiert ein vollständig vertrauenswürdiges, hochverfügbares Verzeichnis, in dem alle aktuell gültigen Zertifikate aller Aussteller der AAI vorgehalten werden. Wird ein Zertifikat vorzeitig ungültig, erfolgt seine Löschung aus dem Verzeichnis.

- *CVT/EFECT* (Certificate Verification Tree, Easy Fast Efficient Certification Technique)
  Die Grundidee von CVT/EFECT [GGM00] ist die generelle Abschaffung einzelner

unabhängiger und signierter Zertifikate. Stattdessen werden die zu signierenden Datenstrukturen (Inhaber-Attribut-Zuordnungen) als Blätter in einem binären Hashbaum erfasst. Erfolgt ein Rückruf, wird das betreffende Blatt entfernt. Wie auch bei CRTs enthalten die inneren Knoten des Baums die Hashwerte ihrer jeweiligen Kindknoten wie in Abb. 3.11 dargestellt. Die IA signiert wiederum nur die Wurzel. Die Verifikation eines Blattes wird mittels der Hashwerte auf dem Pfad zwischen Blatt und Wurzel durchgeführt. Dafür ist auch die Kenntnis der Werte notwendig, welche die Verifikation der inneren Knoten erlauben, d.h. die Hashwerte der benachbarten Teilbäume. Der CVT muss in kurzen Abständen (z.B. täglich) aktualisiert, neu signiert und veröffentlicht werden.

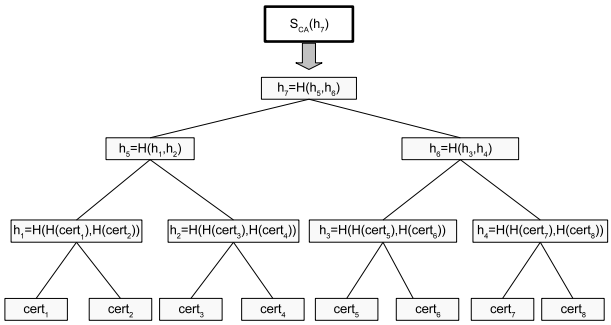


Abbildung 3.11: Beispiel: Certificate Verification Tree

- *CRS (Certificate Revocation Status Directory)/Novomodo*
  CRS [Mic96] / Novomodo [Mic02] ist ein System, welches auf der regelmäßigen Publikation positiver und negativer Statusnachrichten für alle existierenden Zertifikate basiert.
  Es wird angenommen, dass ein Zertifikat d Tage lang gültig ist und jede IA bzw. autorisierte SA die positiven Statusnachrichten, im Folgenden auch Statusinformationen genannt, täglich erzeugt und publiziert.
  Bei der Ausstellung eines Zertifikats wählt eine IA zwei zufällige Bitstrings Y_0 und N_0 der Länge 160 Bit. Mit Hilfe einer konsistenten Hashfunktion H werden die Werte $N_1 = H(N_0)$ und $Y_d = H^d(Y_0)$ berechnet. Y_d wird auch als Validierungsziel bezeichnet.
  Die Werte N_1 und Y_d werden dem Zertifikatinhalt hinzugefügt und im Zuge der Zertifizierung mit signiert. N_0, Y_0 und alle anderen Werte $H^j(Y_0), 1 \leq j \leq d$ bleiben geheim bzw. werden an eine autorisierte SA übergeben.

  Gültigkeitsbestätigung: An jedem Tag $0 < i < d$ berechnet die Status Authority

  $$Y_{d-i} = H^{d-i}(Y_0)$$

  und publiziert diese Statusinformation (SI) ihn in einem öffentlichen Verzeichnis. Ein Prüfer fordert diesen Datensatz dort an und prüft

  $$H^i(Y_{d-i}) \stackrel{?}{=} Y_d.$$

Gelingt die Validierung, sieht der Prüfer das entsprechende Zertifikat als aktuell gültig an. Da das Verzeichnis, in dem die Statusinformationen veröffentlicht werden, nicht vertrauenswürdig sein muss, geht ein Prüfer, der *keinen* aktuellen Wert beziehen kann, sicherheitshalber davon aus, dass das Zertifikat gegenwärtig ungültig ist. Zum nächsten Aktualisierungszeitpunkt kann er erneut versuchen, eine valide SI zu erhalten.

Rückruf: Soll das Zertifikat zurückgerufen werden, publiziert die Status Authority den Wert N_0. Stellt ein Prüfer fest, dass keine aktuelle Positivnachricht verfügbar ist, sucht er im Verzeichnis nach einer solchen Negativnachricht. Er kann dann durch einfache Anwendung der Hashfunktion ermitteln, ob

$$H(N_0) \stackrel{?}{=} N_1.$$

Ist dies der Fall, ist V sicher, dass das Zertifikat ungültig geworden ist.

- *HCRS (Hierarchical Certificate Revocation Scheme)*
  HCRS [ALO98] ist eng mit CRS verwandt. Allerdings werden die positiven Statusnachrichten nicht für jedes Zertifikat separat erstellt und publiziert, sondern für alle Zertifikate einer IA gemeinsam verwaltet.
  Hierfür wird ein Binärbaum aufgebaut, bei dem jedem Blatt der eindeutige Bezeichner eines bestimmten Zertifikats (Bitstring der Länge l) zugeordnet wird. Jeder Elternknoten ist ebenfalls durch einen Bitstring repräsentiert, wobei dieser das den beiden Kindknoten gemeinsame Präfix der Länge $l-1$ darstellt. Die Wurzel des so aufgebauten Baums wird mit $\emptyset$ bezeichnet.
  Für jeden Baumknoten n berechnet die IA dann aus einer Zufallszahl r_n die Werte $H(r_n), H^2(r_n), ..., H^d(r_n)$; dies ist das Äquivalent zur Berechnung der positiven Statusmeldung in CRS. Alle Validierungsziele $H^d(r_n)$ auf dem Pfad vom Blatt zur Wurzel des Baums werden in das Zertifikat aufgenommen.
  Rückrufe werden wie folgt gehandhabt: die IA wählt am Tag i eine Menge $\mathcal{D}_i$ von Knoten, die *nicht* in einem Pfad von einem zurückgerufenen Knoten zur Wurzel vorkommen, und veröffentlicht deren aktuelle Werte $H^{d-i}(r_{n_j})$ mit $n_j \in \mathcal{D}_i$.
  Es sind alle Zertifikate gültig, für die mindestens ein Knoten aus $\mathcal{D}_i$ im Pfad[5] zur Wurzel enthalten ist, und alle Zertifikate ungültig, für die im gesamten Pfad kein Knoten aus dieser Menge vorkommt.
  Fragt ein Prüfer also eine Statusnachricht für ein Zertifikat an, wird ein beliebiger Knoten aus $\mathcal{D}_i$ zurückgeliefert, der im Pfad von dessen entsprechendem Blatt zur Wurzel liegt. Existiert kein solcher Knoten, wird das Zertifikat als ungültig angesehen.
  Abbildung 3.12 zeigt ein Beispiel. Die Zertifikate mit den Seriennummern 010 und 101 werden zurückgerufen (violett). Damit können deren Vorgängerknoten (ebenfalls violett unterlegt) keine Elemente der Menge $\mathcal{D}_i$ sein. Stattdessen wird die Menge der breit umrandeten Knoten als $\mathcal{D}_i$ gewählt.

[5] Der Blattknoten wird selbst zum Pfad gerechnet, d.h. auch ein Zertifikatknoten selbst kann in $\mathcal{D}_i$ enthalten sein, siehe auch Abb. 3.12

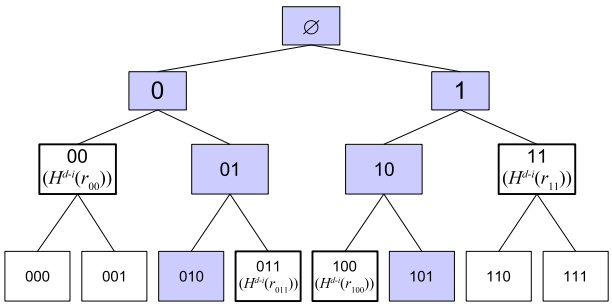


Abbildung 3.12: Beispiel: Hierarchichal Certificate Revocation Scheme

- *NewPKI*
  Das von Zhou [Zho03] vorgestellte rückruffreie NewPKI ähnelt stark CRS/Novomodo. Der grundlegende Unterschied liegt darin, dass eine End-Entität selbst die Kontrolle über die Statusinformationen übernimmt, sowohl über deren initiale Erzeugung als auch über deren regelmäßige Veröffentlichung.
  Will eine End-Entität ein Zertifikat erhalten, wählt sie zunächst selbst eine Zufallszahl r und berechnet $H^d(r)$, wobei d die Gültigkeitsdauer des Zertifikats in Perioden ist. Dann wird $H^d(r)$ zusammen mit weiteren zertifikatbezogenen Daten an eine IA übermittelt. Diese erzeugt daraus ein Zertifikat und sendet es an die End-Entität zurück.
  Nun kann die End-Entität den Status in jeder Periode i neu bestimmen:

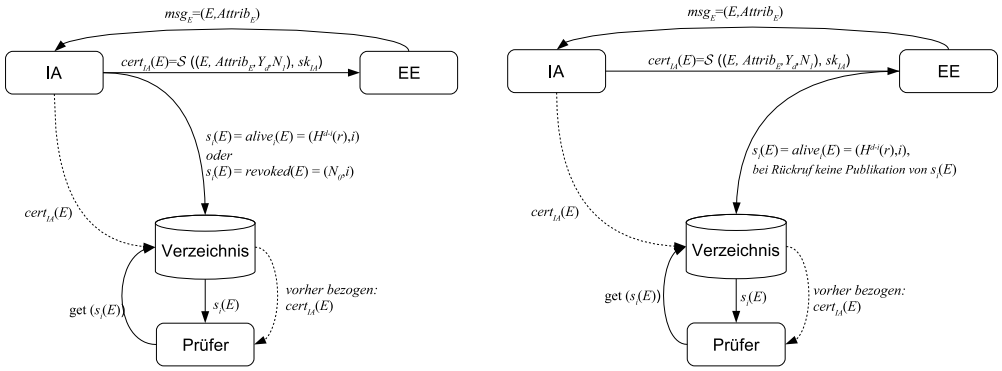
  – Will sie das Zertifikat in der aktuellen Periode nicht verwenden, gibt sie keine aktuelle Statusnachricht heraus. Aus Sicht eines Prüfers gilt das Zertifikat dann als ungültig. Ein expliziter Rückruf wie in CRS/Novomodo ist nicht vorgesehen, d.h. der Prüfer kann die Ungültigkeit nur aus der Abwesenheit der Positivnachricht folgern.

  – Verwendet die End-Entität das Zertifikat in i, veröffentlicht sie $H^{d-i}(r)$ bzw. übermittelt den Wert an den Prüfer. Dieser kann dann die Zertifikatgültigkeit wie bei CRS durch i-faches Hashing und Vergleich mit $H^d(r)$ prüfen.

  Daher ist es möglich, ein Zertifikat nur temporär für ungültig zu erklären, beispielsweise wenn eine End-Entität weiß, dass sie in den kommenden Wochen das Zertifikat nicht verwenden wird. Sie kann dann später jederzeit die Ausstellung von Statusnachrichten wieder aufnehmen.

Bewertung
Für die Bewertung wird im Folgenden unterschieden in:

- Trusted Directory
- CVT als zertifikatloses System
- CRS, HCRS, NewPKI als Systeme auf Basis regelmäßiger Positivnachrichten



(a) CRS/Novomodo (b) NewPKI

Abbildung 3.13: Gegenüberstellung von CRS/Novomodo und NewPKI

Das **Trusted Directory** hat folgende Vorteile:

- *Eindeutigkeit:* Aufgrund der kontrollierten Umgebung kann ein Prüfer sicher sein, aktuelle und korrekte Informationen über Existenz oder Nichtexistenz von gültigen Zertifikaten zu erhalten.
- *Zentralität:* Idealerweise sind die Zertifikate aller IAs in einem einzigen Trusted Directory enthalten.
- *Geringe Datenmengen:* Da nur die gültigen Zertifikate, nicht aber Rückruf- oder andere Statusnachrichten gespeichert werden, sind die insgesamt zu verwaltenden Datenmengen geringer als in jedem anderen der vorgestellten Systeme.

Es treten aber gravierende Nachteile auf:

- *Abhängigkeit von Serversystem:* Der Trusted-Directory-Ansatz stellt sehr hohe Anforderungen an die Verfügbarkeit des Verzeichnisses. Prüfer müssen jederzeit Anfragen an das Verzeichnis stellen können, die autorisierten IAs jederzeit die gespeicherten Daten bearbeiten, löschen oder aktualisieren und neue Zertifikate publizieren. Daher kann dieses System nicht ohne einen hochverfügbaren Datenbankserver auskommen.
- *Vorgegebene AAI-Architektur:* Damit das Verzeichnis vertrauenswürdig ist, dürfen nur privilegierte IAs Schreibzugriff auf die Daten haben. Die Öffnung des Verzeichnisses für beliebige Benutzer, auch private Entitäten, würde hohen Aufwand bei der Benutzerverwaltung erzeugen und die Gewährleistung der Richtigkeit der gespeicherten Daten nahezu unmöglich machen. Daher kann ein Trusted Directory nur für hierarchische oder hybride AAI mit möglichst wenigen IAs, nicht aber für eine dezentralisierte AAI eingesetzt werden.

Ein Trusted Directory eignet sich somit schwerpunktmäßig für den Einsatz innerhalb geschlossener, mit der Außenwelt nicht interagierender Gruppen (z.B. innerhalb eines

Unternehmens) mit hierarchischer AAI-Architektur, umfangreichen finanziellen Ressourcen und einheitlicher, zentralisierter IT-Infrastruktur.

CVT bietet die folgenden Vorteile:

- *Geringer Signaturaufwand für die IA:* Die IA muss immer nur eine einzige Signatur ausstellen, nämlich für die Wurzel des CVT, anstatt einzelne Zertifikate zu generieren und zu signieren.
- *Geringer Kommunikationsaufwand für die IA:* wie für Offline-Systeme, da der Baum nur periodisch aktualisiert wird
- *Unabhängigkeit der Software-Architektur:* wie für Offline-Systeme
- *Positive Statusnachricht:* Es werden ausschließlich Positivnachrichten bereitgestellt. Der Fokus liegt also auf der Verhinderung von falschen Positivannahmen über die Gültigkeit von Inhaber-Attribut-Zuordnungen, weil diese innerhalb einer AAI schwerwiegendere Folgen haben als falsche Negativannahmen.

Nachteilig sind aber die folgenden Punkte:

- *Große zusammenhängende Datenmengen:* Ähnlich CRTs kann ein vollständiger CVT mit vielen Zertifikaten eine sehr große Datenstruktur darstellen, welche aufwendig zu transferieren ist.
- *Verzögerung:* wie für Offline-Systeme
- *Overhead für Prüfer:* wie für Offline-Systeme

Für **Systeme auf Basis von periodisch veröffentlichten Positivnachrichten** (CRS, HCRS, NewPKI) bestehen wiederum andere Vorteile:

- *Aktualität:* Die Statusnachrichten können aufgrund ihrer einfachen und schnellen Berechnung und ihrer geringen Größe in kurzen Zeitabständen bereitgestellt werden. Damit sind die Informationen hinreichend aktuell.
- *Positive Statusnachricht:* Es werden primär Positivnachrichten verwendet, Rückrufe sind allenfalls unterstützend realisiert. So wird verhindert, dass ein Prüfer irrtümlich die Gültigkeit eines Zertifikats annimmt.
- *Geringe Datenmengen:* Die Positiv- und ggf. Negativnachrichten stellen im Gegensatz zu Zertifikaten kleine Datenmengen dar. Beispielsweise beträgt die Größe einer Statusnachricht aus Novomodo nur 1 bis 2% der durchschnittlichen Größe eines X.509-Zertifikats (1-2 Kilobyte [NyB99]), nämlich 20 Byte[6]. Es ist offensichtlich, dass die kleinen, separat verteil- und speicherbaren Statusnachrichten geringe Kommunikations- und Speicherkosten verursachen.

[6]Ein Eintrag in einer CRL mit X.509-Zertifikaten hingegen enthält pro zurückgerufenem Zertifikat mindestens 25 Byte. Außerdem muss noch die Größe der IA-Signatur der Gesamtliste berücksichtigt werden.

- *Unabhängigkeit der Statusnachrichten:* Will ein Prüfer die aktuelle Statusnachricht für ein bestimmtes Zertifikat abrufen, entsteht kein Kommunikationsoverhead, da er nur genau diesen Datensatz erhält und keine Informationen über andere Zertifikate.
- *Paradigma-Konsistenz:* wie für Offline-Systeme

Nachteile:

- *Anfälligkeit für falsche Rückrufannahmen:* Durch den verschobenen Fokus wächst in einem System, das ganz ohne Negativnachrichten arbeitet, die Wahrscheinlichkeit, dass ein Prüfer irrtümlich annimmt, ein Zertifikat wäre zurückgerufen, obwohl er nur die aktuelle Positivnachricht nicht erhalten hat.

Kapitel 4

Sicherheit auf Peer-to-Peer-Ebene

Um die Konstruktion eines zuverlässigen verteilten Verzeichnisses zu ermöglichen, ist zunächst ein geeignetes P2P-Regelwerk als Grundlage auszuwählen, das die speziellen Sicherheitsanforderungen einer solchen Anwendung erfüllt. Nach einer kurzen Vorstellung und Beurteilung existierender Peer-to-Peer-Dateisysteme werden die Sicherheitsanforderungen für ein Peer-to-Peer-Verzeichnis hergeleitet und die in Abschnitt 2.5 präsentierten Protokolle und Regelwerke diesbezüglich qualitativ verglichen und bewertet, um zu einer Entscheidung für ein konkretes Regelwerk zu gelangen.
Danach wird konkret die Datenverfügbarkeit in einem P2P-System, welches auf das gewählte Regelwerk aufbaut, theoretisch analysiert. Dabei werden mit stochastischen Methoden Aussagen über die Verfügbarkeitswahrscheinlichkeit von Datensätzen unter bestimmten Rahmenbedingungen (Angriffswahrscheinlichkeiten, Onlineverhalten, Replikationsfaktoren) zu beliebigen Zeitpunkten hergeleitet.
Weiterhin wird die unter Gesichtspunkten des Kommunikationsaufwandes optimale Anfragepolitik für Datensätze ermittelt.

Die Ergebnisse aus diesem Kapitel dienen als Basis für die in Kapitel 5 vorgestellten Regeln für die Speicherung und Anfrage von Zertifikaten und Statusnachrichten. Sie stellen außerdem den ersten Teil der Sicherheitsanalyse dar (siehe auch Kap. 7).

4.1 Existierende Peer-to-Peer-Datenverwaltungssysteme

Es gibt verschiedene Systeme auf Basis des P2P-Paradigmas, welche für die Verwaltung von Daten konzipiert sind. Man unterscheidet drei Arten von Systemen (angelehnt an [AnS04]):

- Ein **P2P-Filesharing-System** ermöglicht einfachen und direkten Austausch von Daten. Beispiele für Filesharing-Systeme, die auch strukturierte Overlays verwenden, sind neue eMule-Versionen [EMKAD], LimeWire [LIMEW] und BitTorrent-Implementierungen wie Kenosis [KENOS] oder eXeem [EXEEM].
  Sie setzen aber keine umfangreichen Techniken für Sicherheit, Verfügbarkeit, Vollständigkeit und Persistenz der verwalteten Daten ein und kommen daher nicht für ein verteiltes AAI-Verzeichnis in Frage.
- Ein **P2P-Dateisystem** hingegen ist ein System zur Veröffentlichung, Speicherung

und Verteilung von Daten mit den primären Zielen Sicherheit, Performanz und Datenpersistenz. Es werden bewusst Sicherheitstechniken integriert, um diese Anforderungen erfüllen zu können.

- Ein **P2P-Datenbanksystem** ist der Zusammenschluss multipler Datenbanken, die auf verschiedenen Knoten ausgeführt werden. Über ein P2P-Netzwerk werden Verbindungen zwischen diesen Systemen hergestellt, Daten auf verschiedenen Knoten angefragt, koordiniert und ausgetauscht. Beispiele sind LRM [SGM01] und PIER [HML03].
  Für den betrachteten Anwendungsfall kommen diese Systeme aber nicht in Frage, da die Organisation von Zertifikaten und Rückrufen in relationalen Datenbanken auf verschiedenen Knoten sehr aufwendig wäre.

Nur Peer-to-Peer-Dateisysteme passen also hinsichtlich ihrer Anforderungen und Eigenschaften zum Zielsystem des verteilten AAI-Verzeichnisses. Eine geeignete Auswahl wird im Folgenden beschrieben und kurz hinsichtlich ihrer Tauglichkeit als Basis eines P2P-Verzeichnisses für eine AAI diskutiert.

4.1.1 CFS

CFS (Cooperative File System) [DKK01] ist ein mittels Chord und dem Blockspeichersystem DHash (ebenfalls [DKK01]) realisiertes Dateisystem. Zum Load Balancing werden Dateien in Datenblöcke aufgeteilt und diese Segmente auf verschiedenen Knoten abgelegt. Die Identifizierung von Blocks erfolgt über den Hashwert ihres Inhalts (Content Hash). DHash realisiert Caching und die Replikation von Datenblöcken, indem jeder Block auf den ersten k Nachfolgern seines Bezeichners im Chord-Ring abgelegt wird. In neueren Versionen von DHash wird statt der reinen Replikation die hinsichtlich des Speicheraufwandes effizientere Technik des Erasure Coding [WeK02] verwendet.
Auf einem CFS werden mehrere Dateisysteme verschiedener Besitzer umgesetzt, indem spezielle Root Blocks, d.h. Einstiegspunkte, definiert werden, die über den öffentlichen Schlüssel des Inhabers gefunden werden können. Von einem solchen Root Block aus kann ein Benutzer das entsprechende Dateisystem erkunden.
Für ein AAI-Verzeichnis ist die Segmentierung der Datensätze eher hinderlich. Zudem sind AAI-Daten in der Regel kleine Datenmengen: ein X.509-Zertifikat umfasst beispielsweise nur ca. 1-2 Kilobyte [NyB99], die Aufteilung der Daten macht daher wenig Sinn. Auch die Erstellung verschiedener Dateisysteme mit unterschiedlichen Einstiegspunkten behindert den freien Datenfluss. Die Verteilung der authentischen öffentlichen Schlüssel, die zum Lesen der Daten anderer Benutzer notwendig sind, wäre im geplanten verteilten AAI-Verzeichnis bei einer möglicherweise großen Anzahl an publizierenden Instanzen sehr aufwendig bzw. nicht möglich, da hierfür bereits im Verzeichnis gespeicherte AAI-Daten zur Herleitung der Zertifizierungsketten benötigt würden.

4.1.2 FARSITE

FARSITE [ABC02][FARSI] ist eine Entwicklung von Microsoft und realisiert die Verteilung und Replikation von Daten auf vernetzten Desktop-PCs. Diese bilden gemeinsam ein verteiltes Dateisystem, welches einen realen Server oder die lokale Speicherung von Dateien auf den Festplatten der Benutzer ersetzen soll. Die Autorisierung der Teilnehmer

erfolgt auf Basis digitaler Signaturen. Die Zielumgebung ist jedoch nur eingeschränkt skalierbar, da das System für den Einsatz in großen Unternehmen oder Universitäten konzipiert ist. Die Obergrenze für die Netzgröße liegt bei ca. 10.000 Knoten [ABC02]. Zudem werden hohe Anforderungen an die Bandbreite und die ständige Verfügbarkeit der angeschlossenen Rechner gestellt.
Da in einer AAI mit verteiltem Verzeichnis von heterogenen Knoten mit verschiedenen Betriebssystemen, Netzanbindungen, Onlinezeiten usw. auszugehen ist, stellt FARSITE hierfür keine adäquate Technologie dar.

4.1.3 Ivy

Ivy [MMG02] ist ein an NFS angelehntes Dateisystem. Änderungen an der Datenbasis werden von jedem Benutzer in einer Log-Datei protokolliert, die im Blockspeichersytem DHash gespeichert, verteilt und repliziert wird. Bei der Suche nach Daten muss der suchende Teilnehmer selbst über die Vertrauenswürdigkeit anderer Benutzer entscheiden, indem er nur die Log-Dateien der Benutzer verwendet, denen er vertraut.
Ivy setzt den Fokus auf das Vertrauen gegenüber einzelnen Log-Dateien. Da jedem Benutzer der AAI mit verteiltem Verzeichnis die gleiche Datenbasis zur Verfügung stehen soll, ist der dazu orthogonale Ansatz von Ivy nicht passend für diesen Anwendungsfall.
Die Entscheidung über Vertrauenswürdigkeit von Authorities, welche Log-Dateien führen, wird ja erst im Laufe der Operationen auf der gesamten Datenbasis hergeleitet und kann den Benutzern nicht a priori bekannt sein. Der Ausschluss bestimmter Teile der Datenbasis schon *vor* der Suche nach Zertifikaten und Statusnachrichten ist daher nicht sinnvoll.

4.1.4 OceanStore

OceanStore [KBC00][OCEAN] ist ein kostenpflichtiger Verzeichnisdienst und basiert auf dem mit Pastry verwandten Protokoll Tapestry [ZKJ01]. Replikation wird räumlich nahe den erwarteten Endbenutzern durchgeführt [Gee02].
Jeder Benutzer ist bei einem Service Provider registriert, dessen Benutzer- und Speicherbereich eine Administrationsdomäne bildet. Interaktion mit anderen Domänen wird über den Kauf und Verkauf von Speicherplatz zwischen den Service Providern umgesetzt. OceanStore realisiert ein hybrides Peer-to-Peer-Netzwerk mit Einschränkungen bezüglich des freien Datenaustauschs zwischen den Knoten: Die Daten können nicht frei verteilt werden, sondern die Verteilung über Administrationsdomänen hinweg ist vom Verhalten privilegierter Instanzen abhängig. Diesen müssen die Benutzer vertrauen, damit der Dienst realisiert werden kann. Eine hybride Infrastruktur widerspricht aber der Forderung, dass die Verantwortung für ein verteiltes AAI-Verzeichnis bei allen Teilnehmern zu gleichen Teilen liegt, so dass OceanStore und verwandte Produkte[1] für unsere Zwecke ungeeignet sind.

4.1.5 PAST

PAST [RoD01b] ist ein P2P-Dateisystem auf Basis von Pastry mit dem expliziten Ziel der langfristigen Archivierung von Inhalten. Um diese Datenverfügbarkeit erreichen zu

[1] Pangaea [SKK02] ist konzeptuell eng mit OceanStore verwandt, implementiert aber weniger Sicherheitsaspekte.

können, werden alle eingestellten Dateien repliziert und auf verschiedenen Knoten abgelegt. Kontrollierte Löschung von Datensätzen ist nicht implementiert.
Das System wird durch SmartCards unterstützt, welche von einer zentralen Instanz ausgegeben werden und die Einhaltung von Speicherplatzbeschränkungen kontrollieren sowie ein Schlüsselpaar zum Einsatz von Signaturen bereitstellen. Auch die NodeID wird unter Verwendung der SmartCard berechnet.
Die Integrität von Datensätzen wird gesichert, indem jeder publizierende Knoten ein digitales Zertifikat für den Datensatz u.a. mit dem Hashwert über den Inhalt, der FileID und dem gewünschten Replikationsfaktor erzeugt und in der Publikationsphase verwendet. Da für jeden Speichervorgang Bestätigungen von allen Speicherknoten an den Einsteller verschickt werden, ist der Vorgang gut nachvollziehbar. Die Rückmeldungen schützen jedoch nicht vor böswilligem Verhalten von Speicherknoten.
Daten können auch verschlüsselt werden, allerdings ohne dass dabei die SmartCards zum Einsatz kommen. Jeder Benutzer kann ein Kryptosystem seiner Wahl verwenden.
Das SmartCard-Konzept zur Knotenauthentifizierung und zur Kontrolle des vereinnahmten Speicherplatzes ist allerdings für den Anwendungsfall der AAI mit P2P-Verzeichnis nicht anwendbar, da dafür immer eine zentrale, vertrauenswürdige Instanz für die Kartenvergabe vorausgesetzt wird. Ein wesentliches Argument für die Realisierung eines dezentralisierten Verzeichnisdienstes ist aber die Möglichkeit vollständiger organisatorischer Dezentralisierung. Bei Voraussetzung von privilegierten Entitäten könnten daher viele sinnvolle Einsatzszenarien nicht umgesetzt werden (siehe auch Kap. 8).
Insgesamt ist zu bemerken, dass PAST schwerpunktmäßig für die Speicherung und Sicherung nicht öffentlicher Datenbestände konzipiert ist. Um PAST für ein AAI-Verzeichnis brauchbar zu machen, müsste das System extrem reduziert und ohne SmartCards betrieben werden. Zudem ist das zugrunde liegende Protokoll Pastry hinsichtlich seiner Sicherheitseigenschaften gegenüber böswilligen Angriffen nicht optimal (siehe 4.2.3).

4.1.6 Fazit

Im Rahmen der Peer-to-Peer-Dateisysteme wäre PAST für den hier diskutierten Anwendungsfall am ehesten geeignet. PAST ist aber als System für benutzereigene und damit nicht-öffentliche Daten konzipiert und stellt hinsichtlich der SmartCard-Technologie hohe Anforderungen an die Teilnehmer . Es benötigt zudem eine zentrale, vertrauenswürdige Vergabestelle, um die implementierten Sicherheitsmechanismen nutzen zu können. Da das verteilte AAI-Verzeichnis auch für den Einsatz in organisatorisch vollständig dezentralisierten Umgebungen einsetzbar sein soll, ist auch PAST nicht geeignet.
Es kann also für ein verteiltes AAI-Verzeichnis nicht auf bestehende P2P-Dateisysteme zurückgegriffen werden, sondern es ist die Spezifikation einer eigenen Lösung notwendig.

4.2 Auswahl des optimalen Peer-to-Peer-Regelwerks

4.2.1 Basiskennzahlen

Die Vorstufe zur Untersuchung der Protokolle auf ihre Tauglichkeit für ein verteiltes Datenverwaltungssystem ist die Gegenüberstellung ihrer Basiskennzahlen: durchschnittlicher Aufwand für Join, Leave und Lookup sowie Maximalgröße der Routing-Tabellen.

	Aufwand für Join	**Aufwand für Leave**	**Aufwand für Lookup**	**Max. Größe der Routing-Tabelle**
CAN	$O(d)$	$O(d)$	$O(d\,N^{\frac{1}{d}})$	d
Chord	$O(log_2 N)$	$O(log_2^2\ N)$	$\frac{1}{2}log_2 N$	$log_2 N$
Kademlia	$O(log_2 N)$	-	$O(log_2 N)$ oder $O(log_{2^b}\ N)$	$k\,log_2 N$ oder $k\,2^b log_{2^b} N$
P-Grid	$O(log_2 N)$	-	$O(log_2 N)$	$log_2 N$
Pastry	$3 \cdot 2^b \cdot log_{2^b} N$	$O(log_{2^b} N)$	$O(log_{2^b} N)$	$log_{2^b} N + 2\|L\|$
Symphony	$O(log_2^2 N)$	$O(log_2^2 N)$	$O(\frac{1}{q}(log_2^2 N))$	$2q + 2$
Viceroy	$O(log_2 N)$	$O(log_2 N)$	$O(log_2 N)$	12

Tabelle 4.1: Basiskennzahlen für strukturierte Protokolle und Regelwerke

Die beiden unterschiedlichen Werte in Kademlia erklären sich durch eine Optimierung mit erweiterter Routing-Tabelle; in der Regel gilt hier $b = 5$.
Die Protokolle bewegen sich mit Ausnahme von CAN hinsichtlich des Aufwandes für Join, Leave und Lookup in der Größenordnung $O(logN)$. In den Kategorien Performanz und Skalierbarkeit der Netze gibt es also kaum relevante Unterschiede.

4.2.2 Angriffe in Peer-to-Peer-Datenverwaltungssystemen

Allgemein verfolgt ein Angreifer auf ein beliebiges System seine Ziele unter Ausnutzung der Gesamtheit seiner Fähigkeiten und Kenntnisse. In diesem Abschnitt wird speziell ein Angreifer betrachtet, welcher die Kompromittierung eines verteilten Datenverwaltungssystems bezweckt.
Sein Fähigkeitsprofil wird im Folgenden als Worst-Case-Szenario aufbauend auf vier grundlegenden Annahmen und Voraussetzungen definiert. Danach werden Motivation und konkrete Ziele des Angreifers hergeleitet.
Diese Angreiferspezifikation dient dann als Grundlage für die folgende Bewertung der in 2.5 vorgestellten Peer-to-Peer-Protokolle und -Regelwerke anhand von Sicherheitskriterien, die ein Peer-to-Peer-Netzwerk gegen solche Angriffe widerstandsfähig machen. Aus dieser Bewertung resultiert die Entscheidung für ein konkretes Regelwerk.

4.2.2.1 Annahmen und Voraussetzungen für die Sicherheitsbetrachtung

Annahme A1: Sicherheit der Netzwerkebene. Nachrichten zwischen Knoten werden mittels einer logisch unterhalb des Overlays angesiedelten Netzwerkebene, die bei-

spielsweise TCP/IP implementiert, übertragen. Das gilt sowohl für die Weiterleitung von Lookups zwischen Knoten als auch für den Austausch von Datensätzen.
Es wird angenommen, dass auf dieser Ebene geeignete Sicherheitstechniken wie z.B. SSL/TLS [DiA99] zur Verbindungsverschlüsselung zum Einsatz kommen und sie damit sicher gegen Angriffe ist. Konkret heißt dies, dass die direkte Kommunikation zwischen zwei Knoten auf der Netzwerkebene nicht von Dritten abgehört oder verändert werden kann. Jeder Peer kann nur solche Nachrichten lesen und verarbeiten, die *direkt* an seine Netzwerkadresse gerichtet sind. Diese Annahme abstrahiert somit von der Netzwerkebene, auf der prinzipiell verschiedene Technologien wie z.B. TCP/IP, UDP/IP oder ISO/OSI eingesetzt werden können.
Weiterhin wird diese Kommunikation als vollständig, fehlerfrei und nahezu verzugslos angesehen.

Annahme A2: Sicherheit der kryptographischen Methoden. Alle kryptographischen Methoden, die zur Ver- und Entschlüsselung, Signaturerzeugung und -prüfung sowie zur Berechnung von FileIDs und NodeIDs angewandt werden, sind als sicher vorausgesetzt. Ein Angreifer kann diese nicht brechen.

Annahme A3: Angriffswahrscheinlichkeit. Es wird angenommen, dass jeder Knoten mit Wahrscheinlichkeit $p_a > 0$ ein Angreifer ist und somit böswillig handelt. Der Erwartungswert für die Anzahl solcher böswilliger Peers in einem Netz mit N Knoten ist $p_a\, N$.

Annahme A4: Statisches Netzwerk. Vereinfachend wird angenommen, dass sich ein gegebenes Netzwerk mit seinen Knoten und den Verbindungen zwischen ihnen nicht verändert. Fluktuation (Joins und Leaves) von Knoten über die Zeit hinweg wird für die Auswahl der geeigneten P2P-Technik vernachlässigt.

4.2.2.2 Angreiferfähigkeiten

Allgemein kann ein Angreifer auf ein computerbasiertes System entweder als Teilnehmer oder als Außenstehender (externe Entität) versuchen die Datenbasis oder die Funktionsweise des Systems zu beeinträchtigen [BSI06].

Eine böswillige externe Entität kann aufgrund der Annahme *A1* nicht auf Netzwerkebene in den Datenverkehr zwischen den Knoten eines P2P-Netzwerks eingreifen. Knoten auf Overlay-Ebene sprechen diese nie direkt an, da sie keine NodeID besitzt. Weiterhin kann sie selbst keine Nachrichten an andere Knoten senden, so dass sie weder aktiv noch passiv am Datenverkehr im P2P-System beteiligt ist. Dieser Angreifer wird daher nicht weiter betrachtet.

Ein böswilliger Systemteilnehmer verfügt hingegen über wesentlich umfangreichere Möglichkeiten, da er als Peer uneingeschränkt am Nachrichtenverkehr partizipieren kann. Dementsprechend ist er in der Lage, Lookup-Nachrichten für die Suche nach beliebigen Bezeichnern zu erzeugen, an andere Teilnehmer versenden und somit Datensätze im Netzwerk zu publizieren und anzufragen.

Er kennt weiterhin alle Basisparameter des P2P-Regelwerks, beispielsweise die Berechnungsmethoden für NodeIDs und FileIDs sowie verwendete Ver- und Entschlüsselungsalgorithmen, Signatur- und Hashfunktionen. Über alle Datensätze, für die er zu den zuständigen Knoten gehört und die er dementsprechend auf seinem eigenen Computersystem gespeichert hat, besitzt er uneingeschränkte Kontrolle. Er kann sie also löschen, modifizieren oder gegenüber anfragenden Knoten zurückhalten.
Der Angreifer ist selbst Bestandteil von Routing-Pfaden und kann daher alle Lookup-Nachrichten, die ihn erreichen, lesen und untersuchen. Zudem kann er über den nächsten Knoten in einem Routing-Pfad entscheiden: aus seiner Routing-Tabelle kann er einen gemäß den Regeln des Lookup-Verfahrens passenden oder falschen Knoten entnehmen, aber auch von der Tabelle abweichen, beispielsweise um einen Lookup an einen anderen böswilligen Knoten weiterzuleiten.
Mehrere böswillige Systemteilnehmer können gemeinsam eine Verschwörung bilden.

4.2.2.3 Angreiferziele

Angriffe werden - angelehnt an die Arbeit von Srivatsa und Liu [SrL04] - in Angriffe auf die Datenbasis, auf den Lookup und auf die Netzwerkorganisation eingeteilt. Ein Angreifer kann ein oder mehrere der vorgestellten Ziele verfolgen.

Angriffe auf die Datenbasis
Ein Angreifer kann die Verfälschung oder Vernichtung von Datensätzen bzw. die Verschleierung von deren Existenz bezwecken.

- **Gezielter Angriff.** Ein Angreifer strebt die Schädigung konkreter Datensätze an.
- **Ungezielter Angriff.** Der Angreifer zielt auf die Manipulation der Datenbasis als solche ab. Schlimmstenfalls werden alle Datensätze, für die der Angreifer zuständig ist, gelöscht, modifiziert oder anderweitig unverfügbar gemacht.

Angriffe auf den Lookup
Ein Angreifer kann daran interessiert sein, den Lookup für eine bestimmte oder beliebige FileID zu stören.

- **Verzögerung**
  Der ungefährlichste Fall ist die Weiterleitung von Lookup-Nachrichten nicht an einen gemäß der Lookup-Spezifikation passenden Knoten, sondern an einen *zufällig* ausgewählten. Dies hat zur Folge, dass der Routing-Pfad zu zuständigen Knoten länger und somit die Antwort auf den Lookup verzögert wird.
- **Verhinderung**
  Ein Angreifer kann empfangene Lookup-Nachrichten löschen, zurückhalten oder an einen anderen *böswilligen* Knoten weiterleiten. Dadurch kann das Auffinden von zuständigen Knoten verhindert werden, bis der Anfrageknoten einen neuen Lookup startet [CDG02].
- **Verschleierung von Datenbeständen**
  Weiterhin kann ein Angreifer versuchen, bestimmte existierende Datensätze als un-

verfügbar erscheinen zu lassen, indem er die Bezeichner in empfangenen Lookup-Nachrichten so fälscht und weiterleitet, dass andere Knoten für zuständig gehalten werden. Dann wird entweder auf den falschen Knoten nach einem Datensatz gesucht, oder aber er wird an die falschen Knoten zur Speicherung übergeben und ist für Lookups nach seiner ursprünglichen FileID nicht auffindbar.
Ähnlich funktionieren auch Net-Split-Angriffe (Partitioning [SiM02]), im Rahmen derer Lookups in ein paralleles P2P-Netzwerk von Angreifern umgeleitet werden.

Angriffe auf die Netzorganisation
Der Angreifer verfolgt hier das Ziel, die Struktur des Peer-to-Peer-Systems zu unterwandern. Zwei wesentliche Herangehensweisen sind möglich:

- **Betrieb mehrerer Knoten**
  Betreibt ein Angreifer mehrere Knoten, erhöht sich die Wahrscheinlichkeit, dass er Teil eines beliebigen Routing-Pfades oder für einen beliebigen Datensatz zuständig wird. Zudem vervielfacht sich auch die Gesamtzahl der Datensätze, für die er verantwortlich ist. Dies erleichtert Angriffe auf den Lookup und die Datensätze selbst. Die Generierung verschiedener NodeIDs und damit das Auftreten einer Entität als multiple Knoten im Netzwerk wird auch als Sybil Attack[2] bzw. Sybil-Angriff [Dou02] bezeichnet.

- **Selbstpositionierung**
  Eine Entität, die gezielte Angriffe plant, kann versuchen ihren Knoten so im Overlay zu positionieren, dass dieser für einen konkret anvisierten Datensatz zuständig wird. Da ein böswilliger Systemteilnehmer die Verfahren zur Bestimmung von NodeIDs und FileIDs kennt, ist denkbar, dass er selbst (mit hoher Wahrscheinlichkeit mittels zahlreicher Versuche) die entsprechend passende NodeID für sich erzeugt. Es handelt sich hierbei um eine Variante der Sybil Attack.

4.2.3 Vergleich der Sicherheitseigenschaften

Es werden nun die Kriterien entwickelt, anhand derer die Protokolle und Regelwerke hinsichtlich ihrer Robustheit gegenüber den genannten Angriffen bewertet werden.

4.2.3.1 Sicherheit der Datenbasis

Replikation
Jeder publizierte Datensatz wird in Form von k identischen Replika auf verschiedenen Knoten abgelegt. Die meisten Regelwerke verfügen über solche Mechanismen. Damit wird primär das Ziel verfolgt, Datenverlust durch fehlerbedingte Ausfälle von Knoten zu verhindern. Der Oberbegriff *Replikation* wird im Folgenden ausschließlich verwendet, unabhängig davon, ob „klassische“ Replikation, d.h. Erzeugung von identischen Kopien

[2] Bei dieser Bezeichnung handelt es sich um eine Anlehnung an das bekannte Buch *Sybil* von Flora Retha Schreiber [Sch74]. Dieses dokumentarische Werk handelt von einer jungen Frau mit einer multiplen Persönlichkeitsstörung, welche 16 verschiedene Identitäten besitzt. Obwohl inzwischen angezweifelt wird, dass bei der Patientin tatsächlich eine solche Störung vorlag, ist der Name Sybil auch heute noch mit dem Begriff der multiplen Persönlichkeiten verknüpft.

eines Datensatzes, oder eine redundanzmindernde Technik wie Erasure Coding (siehe 4.1.1, [WeK02]) eingesetzt wird.
Tabelle 4.2 vergleicht die verwendeten Replikationstechniken. Der Knoten, der einen bestimmten Datensatz publiziert, wird auch als Einsteller bezeichnet.
Um einen fundierten Vergleich mit den anderen Regelwerken zu ermöglichen, werden die reinen Protokolle Chord und Pastry, für die in ihrer Grundform keine Replikation definiert ist zusammen mit den Replikationstechniken von DHash bzw. PAST betrachtet. Für Viceroy ist kein Replikationsmechanismus vorgesehen [MNR02], aufgrund der Definition des zuständigen Knotens als $succ(b)$ kann aber Replikation wie in DHash realisiert werden.
Bei der Implementierung von Replikationsmechanismen gilt dann jeweils $|\mathcal{N}_b| = k$ für Replikationsfaktor k: alle Replikainhaber sind gemeinsam für den entsprechenden Datensatz zuständig.

	Zuständige Knoten für Replika (Replikationsfaktor $k > 1$)	**Erzeuger**
CAN	k Realitäten Nachbarn halten Backups für den Fehlerfall	k Realitäten: Einsteller Sonst: Knoten mit geringster Distanz zu b transferiert Replika an Nachbarn
Chord + DHash	k erste Nachfolger von Bezeichner b	Erster direkter Nachfolger $succ(b)$
Kademlia	k Knoten mit zu b nächster NodeID	Einsteller
P-Grid	kein fester Wert k, Replikation auf Knoten mit gleichem Zuständigkeitspfad	Einsteller/Netzwerk (Routing mit verschiedenen Einstiegsknoten führt zu verschiedenen Speicherknoten)
Pastry + PAST	k Knoten mit zu b nächster NodeID	Knoten mit geringster Distanz zu b
Symphony	k Nachfolger von b	Erster direkter Nachfolger $succ(b)$
Viceroy	Möglich: k erste Nachfolger von b	Dann: erster direkter Nachfolger $succ(b)$

Tabelle 4.2: Kriterien für die Bewertung der Replikationstechniken

Für die Bewertung einer Replikationsstrategie ist insbesondere interessant, von welchem Knoten die Replika bei Publikation des Datensatzes erzeugt werden. Hierbei werden zwei Methoden unterschieden:

1. *Erstellung durch einen der zuständigen Knoten:* Hier ist ein einziger Knoten, in der Regel der erste Nachfolger der FileID oder derjenige mit minimaler Distanz seiner NodeID zur FileID, für die Erstellung aller weiteren Replika zuständig. Handelt

er böswillig, kann er den Datensatz bereits zum Publikationszeitpunkt dauerhaft unverfügbar machen, indem er ihn nicht weiter repliziert und auch selbst nicht speichert. Die Wahrscheinlichkeit des Datenverlustes bei einem Publikationsvorgang ist also p_a, da sie nur vom zuständigen Knoten abhängt.

2. *Erstellung durch den publizierenden Knoten:* Werden die Replika hingegen vom Einsteller einzeln an die zuständigen Knoten für die Replika übertragen, so besitzt dieser die alleinige Kontrolle über den Replikationsprozess. Ein böswilliger zuständiger Knoten kann dann nur ein Replikat vernichten. Damit ein Datensatz vollständig verloren geht, müssten alle Speicherknoten der Replika böswillig handeln. Die Wahrscheinlichkeit hierfür ist nur $p_a^k \ll p_a$.

Bei Kademlia liegt die Verantwortung für die Replikation immer beim publizierenden Knoten. Werden bei CAN multiple Realitäten eingesetzt, so behält auch hier der Einsteller die Kontrolle über den Replikationsvorgang.
In P-Grid erzeugt zwar auch der Einsteller die Replika, aber nicht im Rahmen eines kontrollierten Replikationsverfahrens: Die Anzahl der Replika ist abhängig davon, wie viele Knoten sich für einen Pfad bereit erklärt haben und wie viele dabei während des Publikationsprozesses gefunden werden (siehe 2.5.4). Feste Replikationsfaktoren können so nicht realisiert werden.
In den anderen Protokollen ist einer der zuständigen Knoten allein für die Replikation verantwortlich.

4.2.3.2 Sicherheit des Lookup

Kurze Routing-Pfade
Bei Publikation und Anfrage von Datensätzen ist es wichtig, dass der Lookup zur Ermittlung des zuständigen Knotens für den Datensatz korrekt ausgeführt wird. Da wegen $A3$ ein Knoten mit Wahrscheinlichkeit p_a böswillig handelt, ergibt sich die Wahrscheinlichkeit p_{lookup}, dass ein Lookup fehlerfrei gelingt, wie folgt:

$$p_{lookup} = (1 - p_a)^{l_{path}}$$

Die Variable l_{path} bezeichnet die Länge des Routing-Pfades. Die Erfolgswahrscheinlichkeit nimmt also bei konstantem p_a mit der Länge des Pfades drastisch ab (siehe Abb. 4.1). Für Lookup-Protokolle, die lange Pfade produzieren, ist somit die Wahrscheinlichkeit für einen Angriff auf den Lookup allgemein höher als für solche mit kurzen Pfaden.

Die durchschnittliche Pfadlänge ist identisch mit dem Aufwand für einen Lookup gemäß der dritten Spalte in Tabelle 4.1. CAN mit einer Pfadlänge $O(d\,N^{\frac{1}{d}})$ ist zwar komplexitätstheoretisch gegenüber den anderen Protokollen mit Pfadlängen der Größenordnung $O(logN)$ im Nachteil, in praktischen Untersuchungen ergab sich aber, dass sich die Pfadlänge in CAN insbesondere bei Verwendung multipler Realitäten nicht wesentlich von diesen unterscheidet [GGG03] [RFH01].

Authentifizierung von Knoten
Um den Lookup gegen Angriffe abzusichern ist die wechselseitige Authentifizierung der beteiligten Knoten sinnvoll. So kann verhindert werden, dass sich böswillige Entitäten

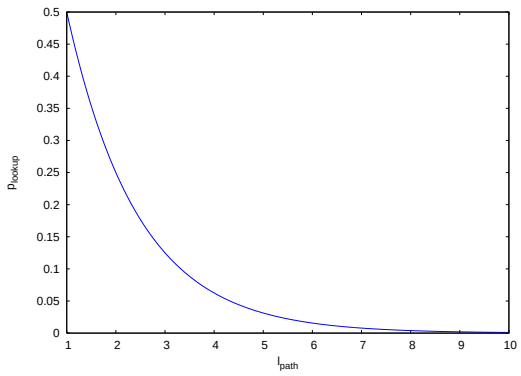


Abbildung 4.1: Wahrscheinlichkeit für den Erfolg eines Lookup in Abhängigkeit von der Pfadlänge, $p_a = 0.5$

als andere Knoten ausgeben, indem sie die eigene NodeID fälschen. Zudem wird die Realisierung von Reputationsmanagement, d.h. die wechselseitige Bewertung des Transaktionsverhaltens von Knoten möglich [KSG03].
Ein solcher Mechanismus auf Basis von Public-Key-Kryptographie ist nur in P-Grid implementiert. Die Authentifizierungsdaten, insbesondere die Zuordnung zwischen NodeID und PublicKey, werden dabei selbst als Datensätze im P2P-Netzwerk gespeichert. Die Entwickler von P-Grid betonen aber in [ADH03] selbst, dass dieses Authentifizierungskonzept auf andere Regelwerke portiert werden kann.

Plausibilitätsprüfung des zuständigen Knotens
Ist der Lookup abgeschlossen, erhält dessen Initiator NodeID und Netzwerkadresse eines zuständigen Knotens (bzw. mehrere solche Kontakte). Es liegt jetzt in seinem Interesse nachzuprüfen, ob diese Antwort plausibel ist, z.B. indem er prüft, ob die NodeID des zurückgelieferten Knotens ein Präfix bestimmter Länge mit FileID teilt. Die absolute Korrektheit kann er damit nicht beweisen, aber zumindest deutliche Fehlmeldungen ausschließen.
Prinzipiell ist eine solche Prüfung in allen P2P-Protokollen realisierbar, bei denen die NodeID Rückschluss auf den Zuständigkeitsbereich eines Knotens zulässt, d.h. in allen genannten mit Ausnahme von P-Grid.

Monitoring der Routing-Pfade
Böswillige Knoten im Routing-Pfad könnten versuchen, eine Anfrage gar nicht oder nicht korrekt weiterzuleiten. Daher liegt es im Interesse des Initiators den Routing-Prozess zu kontrollieren, um hinsichtlich der protokollgemäßen Durchführung des Lookup sicher zu sein. Er will dabei insbesondere prüfen, ob der jeweils nächste Knoten im Pfad den Kriterien des Protokolls genügt. In den meisten Protokollen liegt die Verantwortung für die Wahl des nächsten Knotens und die Weiterleitung der Lookup-Nachricht beim jeweils aktuellen Knoten des Lookup-Vorgangs, d.h. dem letzten Empfänger von $lookup_b$.
Der Initiator muss dann darauf vertrauen, dass die Antwort des Lookup korrekt ist. Auch ein Knoten innerhalb des Routing-Pfades erhält keine Rückmeldung, ob der Peer, an den

er die Lookup-Nachricht weitergeleitet hat, sich protokollgemäß verhält.
Für das Monitoring von Zwischenschritten durch Knoten im Pfad oder durch den Initiator wäre die aufwendige Erweiterung des Peer-to-Peer-Regelwerks unter Entstehung zusätzlicher Verkehrsdaten notwendig, welche den Lookup-Aufwand mindestens verdoppelt. In CAN ist eine solche Modifikation nicht möglich [SrL04]. Kademlia hingegen implementiert standardmäßig Rückmeldungen jedes Knotens an den Initiator des Lookup.

Eingriffsmöglichkeiten des Initiators
Das Monitoring von Pfaden durch den Initator ist nur dann sinnvoll, wenn er auf diese Information reagieren kann, indem er bei Problemen aktiv in den Routingprozess eingreift. Dies ist nur in Kademlia der Fall. Der initiierende Knoten gibt hier selbst in jedem Routingschritt seine Lookup-Nachricht an von ihm selbst ausgewählte Knoten aus der im letzten Schritt zurückgelieferten Menge der Kandidaten weiter.
In allen anderen Regelwerken ist keine Eingriffsmöglichkeit für den Initator vorgesehen; selbst wenn er den Lookup überwachen könnte, hätte er keine Möglichkeit zum Eingriff.

Tabelle 4.3 fasst die Sicherheitskonzepte gegen Routingangriffe zusammen.

	Pfadlänge	**Knotenauthentifizierung**	**Plausibilitätsprüfung**	**Monitoring & Eingriffsmöglichkeiten**
CAN	$O(d\,N^{\frac{1}{d}})$	nein	ja	nein
Chord	$O(log_2\ N)$	nein	ja	nein
Kademlia	$O(log_2\ N)$ / $O(log_{2^b}\ N)$	nein	ja	ja
P-Grid	$O(log_2\ N)$	ja	nein	nein
Pastry	$O(log_{2^b}\ N)$	nein	ja	nein
Symphony	$O(log_2^2\ N)$	nein	ja	nein
Viceroy	$O(log_2\ N)$	nein	ja	nein

Tabelle 4.3: Kriterien für die Bewertung des Routing

4.2.3.3 Sicherheit der Netzwerkorganisation

Regeln zur Berechnung von NodeIDs und Nachprüfbarkeit
In den meisten Protokollen hängt der Zuständigkeitsbereich eines Knotens von seiner NodeID ab. Ein Knoten kann Zuständigkeit für einen bestimmten Datensatz erlangen, wenn es seinem Betreiber möglich ist, die NodeID dieses Knotens selbst zu wählen.
Wird die NodeID jedoch mittels Hashing charakteristischer, nachprüfbarer Daten des Knotens erzeugt, ist die Selbstpositionierung signifikant erschwert. Da die Hashfunktion kollisionsresistent und nicht umkehrbar ist, können zu einer angestrebten NodeID nicht einfach passende charakteristische Eingabedaten erzeugt werden. Während in den Basisversionen einiger Systeme (z.B. CAN) frei wählbare NodeIDs propagiert wurden, was Selbstpositionierung ermöglichen würde, hat sich mittlerweile das Hashing der aktuellen IP-Adresse eines Knotens, ggf. in Kombination mit einer Portnummer, in Forschung und

Praxis für alle Protokolle mit Ausnahme von P-Grid durchgesetzt [CGP05].
P-Grid ist ein Spezialfall: hier kann ein Knoten zwar nicht seine NodeID, aber seinen Zuständigkeitsbereich selbst bestimmen, da dieser wie bereits erwähnt von der NodeID unabhängig ist. In die NodeID selbst gehen zusätzlich zur IP-Adresse der Erstanmeldung auch Datum, Zeit und eine Zufallszahl ein; allerdings bieten diese Daten keinerlei Schutz vor Angriffen, bei denen sich ein Knoten mit böswilliger Absicht einen bestimmten Zuständigkeitsbereich wählt.

Die Verwendung von IP-Adressen als Basis für die NodeID-Berechnung hat allerdings Nachteile, insbesondere wenn IPv6-Adressen verwendet werden: hier kann ein Teilnehmer standardmäßig eine große Anzahl an verschiedenen IP-Adressen besitzen, wodurch Sybil-Angriffe erleichtert werden [DiH06] [CDG02]. Die in einigen Regelwerk-Implementierungen propagierte[3] Kombination von Netzwerkadresse und Portnummer ist ebenfalls problematisch, da hierdurch unter Ausnutzung verschiedener Ports der Betrieb mehrerer Knoten auf einem einzigen Rechner möglich wird (erfolgreiche Sybil Attack). Daher ist die Entwicklung alternativer Möglichkeiten zur NodeID-Berechnung ein großes Ziel der aktuellen Peer-to-Peer-Forschung. Sie sind allerdings derzeit noch kein Bestandteil von P2P-Regelwerken und werden hier daher nur kurz angerissen.
Ein Ansatz zur Erschwerung von Sybil-Angriffen, der ohne eine zentrale Instanz zur NodeID-Vergabe [Dou02] auskommt, ist der Self Registration Approach von Dinger und Hartenstein [DiH06]: Ein Knoten berechnet seine NodeID aus IP-Adresse und Port[4] und registriert diese bei r bestimmten existierenden Knoten, welche die Korrektheit der NodeID prüfen und dann auf Anfragen anderer Peers nach dieser ID einen entsprechenden Registrierungsstatus liefern. Ein anderer Knoten akzeptiert diese NodeID als korrekt, wenn mindestens $\frac{r}{2}$ der Registrierungsknoten eine positive Rückmeldung liefern und somit als „Fürsprecher“ für den betreffenden Knoten agieren.
Ein weiterer Ansatz basiert auf Crypto Puzzles: hier muss jeder Knoten, um eine NodeID zu berechnen, zunächst eine bereitgestellte kryptographische Aufgabe lösen und die Lösung stets bereithalten. So ist die Erzeugung jeder zusätzlichen NodeID mit Aufwand verbunden [CDG02].

Zuweisung der Zuständigkeitsbereiche
In allen präsentierten Protokollen mit Ausnahme von P-Grid sind die Zuständigkeitsbereiche für die FileIDs an die NodeIDs der Knoten gebunden. Somit kann sich ein Knoten nur dann selbst im Overlay positionieren, wenn er die Regeln zur Berechnung von NodeIDs umgehen kann, wie im vorigen Abschnitt beschrieben.
Anders ist es allerdings bei P-Grid: Hier existiert keine Verbindung zwischen NodeID und Zuständigkeitsbereich, die Zuständigkeiten werden selbständig zwischen den Knoten ausgehandelt. Da diese Verhandlungen zwar festen Regeln, aber keinen Kontrollen unterliegen, und beliebig viele Knoten gleichzeitig für ein Präfix verantwortlich sein können, ist es für einen böswilligen Knoten leicht möglich, die Verhandlungen zu umgehen und sich selbst vorteilhaft zu positionieren.

[3] Die Hinzunahme der Port-Nummer wird dort vor allem mit der Motivation empfohlen, in Netzwerken mit NAT-Gateways, welche die IP-Adressen der verwalteten Rechner maskieren und z.B. in Firmennetzwerken auf eine einzige IP-Adresse abbilden, den Betrieb mehrerer Knoten zu ermöglichen.
[4] *Anmerkung:* Auch hier tritt wieder die bereits oben erwähnte Port-Problematik auf.

4.2.4 Bewertung und Entscheidung

4.2.4.1 Gewichtung

Die in den vorigen Abschnitten ermittelten und untersuchten Sicherheitskriterien sind unterschiedlich relevant für die Gesamtsicherheit eines Systems. Daher ist eine Gewichtung als Grundlage für eine fundierte Entscheidung notwendig.

Als wichtigste Eigenschaften werden die Möglichkeit zur Verfolgung und Korrektur des Routing-Pfades, also das *Monitoring* und die *Eingriffsmöglichkeiten* des Lookup-Initiators eingestuft, weil die konsequente, standardmäßige Überprüfung aller Schritte im Routing-Pfad falsche Weiterleitungen verhindern kann und die Prüfung der Plausibilität des zurückgelieferten Knoten hinfällig macht. Auch die Länge der Pfade ist nur dann eine kritische Größe, wenn der Lookup-Initiator Angriffe nicht erkennen und daher nicht auf sie reagieren kann.
Da Replikation die Vollständigkeit der Datenbestände sowohl bei Angriffen als auch bei Ausfällen sichert und daher ein unumgängliches Instrument zur Sicherung der Funktionsfähigkeit und Zuverlässigkeit eines Datenverwaltungssystems darstellt, wird die zweithöchste Gewichtung dem Kriterium *Erzeuger der Replika* zugeteilt. Kann der Einsteller die Replika selbst generieren, hat er aufgrund Annahme $A1$ Sicherheit darüber, dass tatsächlich k seiner Datensätze auf den zuständigen Knoten angekommen sind. Sie können dann ausschließlich durch böswilliges Verhalten der speichernden Knoten selbst zerstört werden[5].

	Monitoring & Eingriffsmöglichkeiten	**Erzeuger Replika**	**Verhinderung Selbstpositionierung**
CAN (k Realitäten)	-	+	+
Chord + DHash	-	-	+
Kademlia	+	+	+
P-Grid	-	(+) (ohne festes k)	-
Pastry + PAST	-	-	+
Symphony	-	-	+
Viceroy	-	-	+

Tabelle 4.4: Bewertung der zentralen Sicherheitskriterien

Die Möglichkeit zur *Selbstpositionierung* erleichtert Sybil-Angriffe und wird daher als Ausschlusskriterium gewertet.

[5]Wegen Annahme $A4$ wird die Gefahr durch Knoten, die zu einem späteren Zeitpunkt zuständig werden, hier ausgeblendet. Ihre Existenz wird erst in Abschnitt 4.3 ausführlich in die Betrachtungen einbezogen.

4.2.4.2 Ergebnis

Als einziges der Protokolle erhält Kademlia positive Bewertungen in allen drei Hauptpunkten. Zudem ist Kademlia durch seine Integration in aktuellen Filesharing-Systemen bereits erfolgreich in der Praxis erprobt.

Kademlia erfüllt also die wichtigsten Sicherheitsanforderungen auf Peer-to-Peer-Ebene und ist somit das am besten geeignete Regelwerk als Basis für ein verteiltes Datenverwaltungssystem im Allgemeinen und das verteilte Datenverwaltungssystem für eine Authentifizierungs- und Autorisierungsinfrastruktur im Speziellen.

4.3 Stochastische Analyse der Datenverfügbarkeit in Kademlia-basierten Netzwerken

Da bisher keine ausführlichen Untersuchungen zur zeitabhängigen Verfügbarkeit von Datensätzen in Kademlia existieren, wird nun ein stochastisches Formelwerk entwickelt, mit Hilfe dessen konkrete Parameter des Netzwerks wie die maximale Fluktuation von Knoten pro Zeiteinheit und der ideale Replikationsfaktor unter bestimmten Rahmenbedingungen hergeleitet werden.

Nach der Vorstellung grundlegender Definitionen wird die Formel für die Basisverfügbarkeit, d.h. die Verfügbarkeit in einem Netzwerk ganz ohne böswillig handelnde Knoten, nach einer Grundidee der Entwickler von Kademlia [MaM02] aufgestellt und untersucht. Dann wird die Betrachtungsgranularität verfeinert, indem die Periodenlänge auf Minuten verringert wird. So ist eine genauere Analyse der Verfügbarkeitsentwicklung möglich. Die in der Basisverfügbarkeit noch nicht berücksichtigten Angriffe durch böswillige Knoten werden integriert.
Die Betrachtung wird auf größere Zeiträume von mehreren Stunden ausgeweitet, wobei auch der Kademlia-eigene Republishing-Mechanismus, d.h. die automatische Wiederveröffentlichung von Datensätzen durch die Speicherknoten von Replika in die Analyse einbezogen wird. Das Ziel ist die Quantifizierung der Verfügbarkeitsentwicklung für gespeicherte Datensätze über lange Betrachtungszeiträume.

4.3.1 Definitionen

Definition 4.1 ***Knoten- und Datenbezeichner:***
Es werden zwei Arten von Kademlia-spezifischen Bezeichnern verwendet:
NodeID n_i***:*** *Bezeichner eines Knotens i, Bitstring der Länge 160 Bit*
FileID b***:*** *Bezeichner eines Datensatzes, Bitstring der Länge 160 Bit*

In der Analyse wird genau *ein* konkreter Datensatz und daher genau eine FileID betrachtet. Der Datensatz residiert in Form von identischen, d.h. inhaltsgleichen und auch bezeichnergleichen Replika auf verschiedenen Knoten des P2P-Netzwerks.

Definition 4.2 ***Replikationsfaktor:*** *Der (angestrebte) Replikationsfaktor für Datensätze wird mit k bezeichnet.*

Definition 4.3 ***Menge zuständiger Knoten*** $\mathcal{N}_b$*:*
$\mathcal{N}_b$ bezeichnet die Menge der Knoten, welche gemeinsam für den Datensatz mit Bezeichner b zuständig sind. Zu jedem beliebigen Zeitpunkt enthält die Menge $\mathcal{N}_b$ k verschiedene Elemente (Knoten), nämlich die k Knoten $\{n_1, n_2, \ldots, n_k\}$ mit den geringsten Distanzen $\Delta(n_i, b)$ zur FileID b. Jeder Knoten aus $\mathcal{N}_b$ verfügt idealerweise über ein Replikat des entsprechenden Datensatzes.

Im Rahmen der Analyse werden nur die Knoten in $\mathcal{N}_b$ und ggf. der publizierende bzw. anfragende Knoten betrachtet. Es ist zu beachten, dass $\mathcal{N}_b$ als abstrakte Menge die k Knoten mit aktuell nächster NodeID zu b umfasst, unabhängig davon, ob diese Knoten selbst Kenntnis davon haben, dass sie zu $\mathcal{N}_b$ gehören und davon, ob sie ein Replikat des Datensatzes mit Bezeichner b speichern.

Der im Folgenden definierte Begriff der Verfügbarkeit, wie er im Rahmen dieser stochastischen Analyse verwendet wird, ist für das Verständnis der Untersuchung zwingend erforderlich.

Definition 4.4 ***Verfügbarkeit****: Ein Datensatz wird als in einer Periode t verfügbar bezeichnet, wenn zum Ende von t mit einer Wahrscheinlichkeit $p_{av} \geq p_{av_{min}}$ mindestens ein Replikat des Datensatzes im Netz vorhanden ist.*

In der Regel ist $p_{av_{min}} = 0.99$ gesetzt (z.B. [BSV02]).

Weiterhin liegen der formalen Beschreibung und Analyse der Verfügbarkeitsentwicklung in Kademlia zwei durch die Eigenschaften des konkreten Netzwerks und dessen Teilnehmer festgelegte Parameter zugrunde:

Definition 4.5 ***Netzwerkparameter***

Angriffswahrscheinlichkeit p_a***:*** *(Durchschnittliche) Wahrscheinlichkeit, dass ein Knoten böswillig handelt. Es wird angenommen, dass er alle ihm zur Verfügung stehenden Möglichkeiten, die im Netz gespeicherten Daten und die Funktionsweise des Netzwerks negativ zu beeinflussen, ergreifen wird (ungezielter Angriff). Die wichtigsten Angriffspunkte sind die Löschung von Datensätzen, für die er zuständig wäre, und die Nicht-Durchführung von datenerhaltenden Maßnahmen wie Republishing oder Transfer on Join, d.h. die Übergabe von Datensätzen an einen Knoten, der $\mathcal{N}_b$ über einen Join neu beitritt.*

Fluktuationswahrscheinlichkeit p_o***:*** *(Durchschnittliche) Wahrscheinlichkeit, dass ein existierender Knoten in einer Periode das Netzwerk verlässt. Er wird verzögerungsfrei aus $\mathcal{N}_b$ entfernt und durch einen anderen Knoten ersetzt, so dass $|\mathcal{N}_b|$ immer gleich bleibt.*

Der Knoten, welcher an Stelle des ausgefallenen in $\mathcal{N}_b$ aufgenommen wird, ist in der Regel bereits vorher online gewesen, hat nur bislang nicht zu den k Knoten mit nächster NodeID zu b gehört. Er kann aber auch ein neuer Teilnehmer des Netzes sein, dessen NodeID im entsprechenden Bezeichnerbereich liegt.
Unterschieden werden im Speziellen p_{o_m} (minütliche Fluktuationswahrscheinlichkeit), p_{o_h}

(stündlich) und p_{o_d} (täglich). Die Umrechnungen zwischen minütlicher, stündlicher und täglicher Fluktuationswahrscheinlichkeit ergeben sich folgendermaßen:

$$p_{o_h} = 1 - (1 - p_{o_m})^{60} \tag{4.1}$$

$$p_{o_d} = 1 - (1 - p_{o_h})^{24} \tag{4.2}$$

Die stündliche Fluktuationswahrscheinlichkeit ist also als Gegenwahrscheinlichkeit dazu, dass ein Knoten über 60 Minuten online bleibt. Die Formel für tägliche und stündliche Fluktuationswahrscheinlichkeit ergibt sich analog.

4.3.2 Allgemeine Präliminarien der Analyse

Das Untersuchungsziel der Analyse ist die Feststellung, ob unter gegebenen Bedingungen die Verfügbarkeit eines Datensatzes über eine bestimmte Zeit hinweg garantiert werden kann. Diese Zeit hängt von der Lebensdauer der verwalteten Zertifikate und Statusinformationen ab.
Die Wahl einer geeigneten, realistischen Belegung für die Parameter aus Definition 4.5 ist notwendig, um zu brauchbaren quantitativen Aussagen zu gelangen. Dabei ist aber eine zu optimistische Schätzung zu vermeiden, um dem Paradigma der Worst-Case-Analyse Rechnung zu tragen.

4.3.2.1 Parameterbelegung Angriffswahrscheinlichkeit

Es wird vorausgesetzt, dass Methoden zur signifikanten Erschwerung von Sybil-Angriffen - wie in 4.2.3.3 erwähnt - eingesetzt werden, um den gleichzeitigen Betrieb mehrerer Knoten durch einen einzigen böswilligen Benutzer signifikant zu erschweren. Außerdem wird vorausgesetzt, dass mindestens die Hälfte aller Knoten regelkonform handelt.
Unter diesen Voraussetzungen wird eine Angriffswahrscheinlichkeit $p_a = 0.5$ als adäquat zur Abdeckung des Worst Case angesehen.

4.3.2.2 Parameterbelegung Fluktuationswahrscheinlichkeit

Gemäß unterschiedlicher Einsatzszenarien (siehe auch Kapitel 8) ist für die hier durchgeführte Analyse auch die Fluktuationswahrscheinlichkeit anzupassen.
Für ein System mit hoher Fluktuation, d.h. ein Netzwerk mit in hohem Maße heterogenen Benutzerprofilen und einer hohen Quote an privaten PCs, wird eine Fluktuationswahrscheinlichkeit $p_{o_h} = 0.8 \, (\Leftrightarrow p_{o_m} \approx 0.0265)$ angesetzt. Dies bedeutet, dass ein Knoten mit der Wahrscheinlichkeit $1 - p_{o_h} = 0.2$ von einer beliebigen Stunde zur nächsten *nicht* offline geht. Die Wahrscheinlichkeit, dass er über 24 Stunden ohne Unterbrechung online bleibt, ist dann verschwindend gering (ca. $1.68 \cdot 10^{-17}$).
Höhere Fluktuationswahrscheinlichkeiten werden nicht betrachtet: es ist Prämisse des gegebenen Anwendungsfalls, dass zumindest einige der Knoten im Netz - auch beim Einsatz des Systems im Internet- hohe Onlinezeiten und geringe Fluktuation aufweisen. Es ist nicht davon auszugehen, dass ausschließlich PCs mit sehr kurzen Online-Phasen an dem System partizipieren. Die länger verfügbaren Knoten verringern die durchschnittliche Fluktuationswahrscheinlichkeit, selbst wenn der Großteil der Knoten eine kurze Onlinezeit und damit eine hohe Fluktuationswahrscheinlichkeit je Periode aufweist.

Weiterhin werden Netze mit moderater Fluktuation betrachtet. Hier ist der Anteil an stabileren, über längere Zeiträume verfügbaren Peers, z.B. in Form von PCs und Workstations an Universitäten, höher als im obigen Fall. Als Fluktuationswahrscheinlichkeit wird hier $p_{o_h} = 0.5\,(\Leftrightarrow p_{o_m} \approx 0.0115)$ angesetzt.

Praktische Untersuchungen zum Onlineverhalten von Knoten in P2P-Netzwerken lassen vermuten, dass die reale Fluktuationswahrscheinlichkeit geringer als hier angenommen ausfallen wird. Im P2P-Filesharing, das fast nur von Privatnutzern betrieben wird, sind die Sitzungslängen in der Regel kurz, z.B. im Schnitt eine Stunde bei Napster [CLL02]. In P2P-Netzwerken, die sich nicht nur aus Privat-PCs zusammensetzen, ist aber die erwartete Sitzungsdauer (Onlinedauer) pro Knoten wesentlich höher, z.B. wurden durchschnittlich 28.6 Stunden für Overnet [BTC04] und sogar 109 Stunden in Firmenetzwerken gemessen [BDE00]. Für ein verteiltes AAI-Verzeichnis wird mit einer durchschnittlichen Online-Dauer gerechnet, die zwischen der in extrem stark fluktuierenden Filesharing-Systemen wie Napster und der in wesentlich geringer fluktuierenden Netzwerken wie Overnet liegt.
Problematisch ist die Umrechnung der durchschnittlichen Onlinezeiten in adäquate Werte für die Fluktuationswahrscheinlichkeit, da diese von weiteren Faktoren wie der konkreten Verteilung der Onlinedauern abhängt. Eine grobe Approximation ergibt sich wie folgt: Der Erwartungswert für die Onlinedauer pro Knoten wird als fixe Größe angenommen, d.h. jeder Knoten geht genau y Perioden nachdem er online gegangen ist wieder offline. Nun betrachten wir die Menge aller Knoten N zu einem beliebigen Zeitpunkt t. Jeder Knoten ist vor einer nicht bekannten Anzahl an Minuten $\leq y$ online gegangen, wobei jede mögliche bisherige Onlinedauer als gleich wahrscheinlich gilt. Im Zeitschritt $t \rightarrow t+1$ sind nun $\frac{1}{y}N$ Knoten am Ende ihrer erwarteten Onlinedauer angelangt und verlassen das Netz. Diese Quote $\frac{1}{y}$ ist demnach zugleich die Approximation der stündlichen Fluktuationswahrscheinlichkeit pro Knoten ($p_{o_h} \approx 0.035$ für erwartete Online-Zeit 28.6 Stunden bzw. $p_{o_h} \approx 0.009$ für 109 Stunden). Aus einer weiteren Untersuchung [TaV06] der erwarteten Fehlerzahl pro Knoten pro Stunde ergibt sich als Näherung $p_{o_h} \approx 0.196$.
Diese Approximationen belegen, dass die für die Untersuchung gewählten Werte für p_{o_h} die in der Realität erwartete Fluktuationswahrscheinlichkeit deutlich übertreffen und somit den Worst Case abdecken.

4.3.3 Basisverfügbarkeit

4.3.3.1 Annahmen und Voraussetzungen

Als Basisverfügbarkeit wird im Rahmen dieser Arbeit die in [MaM02] herangezogene Datenverfügbarkeit in einem idealisierten Netzwerk ohne Angreifer verstanden. Für ihre Berechnung werden von Maymounkov und Mazières die folgenden vereinfachenden Annahmen getroffen:

- **Replikation mit festem Faktor:** Es entstehen bei der Publikation eines Datensatzes Replika auf allen k Knoten aus $\mathcal{N}_b$.
- **Diskrete Betrachtung:** Es wird nur ein diskreter Zeitschritt untersucht, in welchem vom Zustand zum Anfang einer Periode zum Zustand am Periodenende über-

gegangen wird, ohne Zwischenzustände innerhalb dieses Zeitraums zu berücksichtigen.

- **Stündliche Betrachtung:** Ein Zeitschritt deckt eine Stunde ab.
- **Keine Neuerstellung von Replika:** Jedes Replikat, das auf einem Knoten aus $\mathcal{N}_b$ gespeichert ist, der in der betrachteten Periode in den Status offline wechselt, ist verloren. Der betrachtete Zeitschritt beinhaltet insbesondere *kein Republishing*, d.h. keine selbstorganisierende Neuerstellung von Replika durch die verbleibenden zuständigen Knoten.
- **Keine Angreifer:** Alle Knoten verhalten sich protokollgemäß - es gibt kein böswilliges Verhalten, für die Angriffswahrscheinlichkeit gilt daher hier zunächst $p_a = 0$.

Es ist einsichtig, dass die Basisverfügbarkeit keine ausreichende Maßzahl für die Verfügbarkeit in einem nicht idealen System darstellen kann. Daher werden in den späteren Abschnitten ab 4.3.4 erweiterte Formeln entwickelt, welche auf realistischere Annahmen aufbauen (siehe auch 4.3.3.4).

4.3.3.2 Formalisierung

Die Wahrscheinlichkeit, dass die - unter den obigen Voraussetzungen - am Ende der Periode noch vorhandene Replikazahl $\tilde{k}$ genau i mit $0 \leq i \leq k$ entspricht[6], bestimmt sich wie folgt:

$$p(\tilde{k} = i)_{(p_{o_h},k)} = \binom{k}{i} \cdot (1 - p_{o_h})^i \cdot (p_{o_h})^{k-i} \tag{4.3}$$

Die Werte der Zufallsvariable $\tilde{k}$ sind binomialverteilt; $(1 - p_{o_h})^i$ bezeichnet die Wahrscheinlichkeit, dass i Knoten in dieser Periode online bleiben, $(p_{o_h})^{k-i}$ die Wahrscheinlichkeit, dass alle weiteren Knoten aufgrund von Fluktuation das Netz verlassen.

Die Wahrscheinlichkeit, dass *mindestens* i Replika am Ende der Periode existieren, ergibt sich dann wie folgt:

$$p(\tilde{k} \geq i)_{(p_{o_h},k)} = \sum_{j=i}^{k} p(\tilde{k} = j)_{(p_{o_h},k)} \tag{4.4}$$

Insbesondere gilt für die Wahrscheinlichkeit, dass *mindestens ein* Replikat am Ende der Periode existiert:

$$p(\tilde{k} \geq 1)_{(p_{o_h},k)} = 1 - p(\tilde{k} = 0)_{(p_{o_h},k)} \tag{4.5}$$

Der Erwartungswert für die Anzahl der vorhandenen Replika am Ende der Periode ist dann:

$$\begin{aligned} E(\tilde{k})_{(p_{o_h},k)} &= \sum_{i=0}^{k} i \cdot p(\tilde{k} = i)_{(p_{o_h},k)} \\ &= (1 - p_{o_h}) \cdot k \end{aligned} \tag{4.6}$$

[6]Entgegen der gängigen Nomenklatur (z.B. [FKP04]) werden Zufallsvariablen hier ausschließlich in Kleinbuchstaben notiert.

4.3.3.3 Analyse

Die Basisformel gibt Aufschluss über die Wirksamkeit des Replikationsmechanismus von Kademlia unter der Annahme der Fluktuation von Knoten. Sie ist somit die Grundlage für alle weiteren Analyseschritte.
Abbildung 4.2 zeigt die Verteilung der Wahrscheinlichkeit für i Replika zum Perioden-

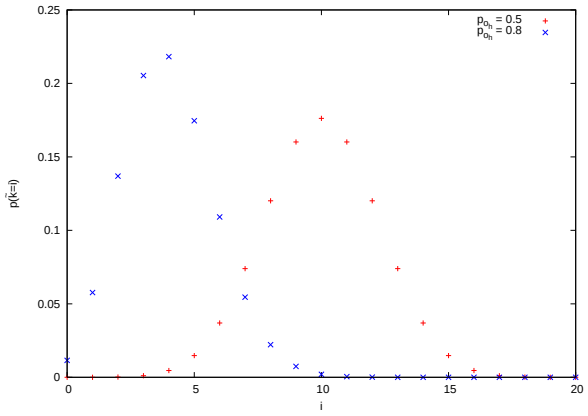


Abbildung 4.2: Basisformel: Wahrscheinlichkeitsverteilungen $p(\tilde{k} = i)_{(p_{o_h},k)}$ für $p_{o_h} = 0.5$, 0.8; $k = 20$

ende gemäß Formel 4.3 in Abhängigkeit von i. Der Erwartungswert $E(\tilde{k})$ ist auch aus der Abbildung ablesbar: Die Wahrscheinlichkeitsverteilung bildet stetig approximiert eine Normalverteilungskurve mit Maximum bei $\tilde{k} = 4$ für $p_{o_h} = 0.8$ bzw. $\tilde{k} = 10$ für $p_{o_h} = 0.5$.

Abbildung 4.3 zeigt die Wahrscheinlichkeit, dass bei dem in [MaM02] empfohlenen Replikationsfaktor $k = 20$ zum Ende der Periode mindestens ein Replikat im Netz vorhanden und der Datensatz somit verfügbar bleibt, in Abhängigkeit von der Fluktationsrate p_{o_h}. Es ist zu sehen, dass bei Fluktuationsraten $p_{o_h} < 0.8$ der Datensatz verfügbar ist, da $p(\tilde{k} \geq 1)_{(p_{o_h},k)} \geq 0.99$. Die Verfügbarkeitswahrscheinlichkeit unterschreitet bei ca. $p_{o_h} = 0.795$ die Grenze $p_{av_{min}} = 0.99$.

4.3.3.4 Bewertung der Basisverfügbarkeit

Die vorgestellte Basisverfügbarkeit ist die einzige von den Kademlia-Entwicklern in ihrem Proof of Concept [MaM02] implizierte Formel, mit der die Sicherheit des Protokolls untermauert werden soll.
Sie weist aber *wesentliche* Defizite in ihrer Aussagekraft auf, da bestimmte Eigenschaften des Kademlia-Regelwerks ebenso wie solche von P2P-Systemen allgemein nicht berücksichtigt werden:

- **Zeitaspekte:** Die Basisformel betrachtet den Knotenausfall und die damit einhergehende Verfügbarkeit von Datensätzen nur in einem diskreten Abschnitt von einer Stunde. Es wird dabei angenommen, dass jeweils zum Beginn der nächsten Stunde

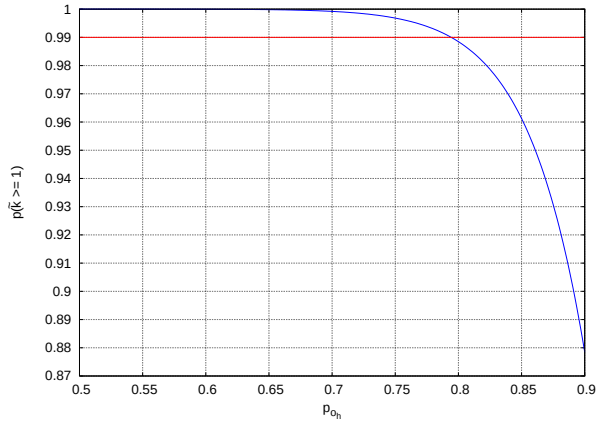


Abbildung 4.3: Basisformel: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{(p_{o_h},k)}$ in Abhängigkeit von p_{o_h}; $k = 20$

durch das Republishing wieder k Replika vorhanden sind, so dass die Betrachtung über mehrere Stunden hinweg als nicht notwendig angesehen wird. Diese Annahme kann in einem realen System aber *nicht* erfüllt werden, wie in den folgenden Punkten erläutert wird.
Zudem kann die Entwicklung *innerhalb* jedes Zeitraums von einer Stunde mit Hilfe der Basisformel nicht untersucht werden - dies ist aber für die Analyse interessant, z.B. um minutengenaue Verfügbarkeitswahrscheinlichkeiten, feingranulare Verfügbarkeitsentwicklung oder variierende Republishing-Zeiträume untersuchen zu können.
Zeitaspekte sind daher mit feinerer Granularität und über längere Zeitspannen, insbesondere im Hinblick auf die gesamte Gültigkeitsdauer eines Datensatzes, zu berücksichtigen.

- **Transfer on Join:** Betritt ein neuer Knoten n_i, der aufgrund seiner NodeID zu $\mathcal{N}_b$ gehört, das Netz und erlangt der Peer n_e mit minimaler Distanz $\Delta(n_e, b)$ im Zuge dieses Join-Verfahrens von n_i Kenntnis[7], so sendet n_e eine Speicheraufforderung (`STORE`-Request) für ein Replikat an den neuen Knoten. Durch diesen sogenannten *Transfer on Join* entstehen neue Replika, die aber in der Basisformel keine Berücksichtigung finden.

- **Existenz von Angreifern:** Die Annahme, dass alle Knoten protokollgemäß handeln, ist in einem realen P2P-System nicht haltbar, da hier stets böswillige Teilnehmer existieren. Für die Angriffswahrscheinlichkeit gilt im Folgenden: $p_a > 0$. Es werden verschiedene Angriffszeitpunkte unterschieden:

  - **Angriffe zum Publikationszeitpunkt:** Jeder der k Knoten, die ein `STORE`-Request für ein Replikat erhalten, wird mit Wahrscheinlichkeit p_a den Da-

[7] Da n_i beim Join einen Lookup nach seiner eigenen NodeID durchführt, ist es wahrscheinlich, dass n_e als Knoten mit geringer Distanz $\Delta(n_e, n_i)$ die Lookup-Nachricht erhält und somit von dem neuen Knoten erfährt.

tensatz nicht speichern, obwohl er gegenüber dem Erzeuger der Replika die Speicherung bestätigt. Damit ist es wahrscheinlich, dass schon zu Beginn der Lebensdauer des Datensatzes weniger als k Replika existieren.

- **Angriffe in der Pre-Republishing-Phase:** Knoten, die zum Publikationszeitpunkt „gut“ waren, bleiben auch in dieser Phase „gut“. Angriffe können nur stattfinden, indem Knoten beim Transfer on Join böswillig handeln und dadurch weniger neue Replika als vorgesehen entstehen.
- **Angriffe zum Republishing-Zeitpunkt:** In regelmäßigen Zeitabständen übermittelt jeder Knoten `STORE`-Requests an die anderen $k-1$ Knoten aus $\mathcal{N}_b$, um denjenigen neu hinzugekommenen Knoten in $\mathcal{N}_b$, welche bisher noch kein Replikat besitzen, den Datensatz zur Speicherung zu übergeben. Auch hier werden mit hoher Wahrscheinlichkeit nicht alle diese Knoten Replika erzeugen. Weiterhin ist es auch möglich, dass der für das Republishing verantwortliche Knoten böse ist und daher die Wiederveröffentlichung des Datensatzes unterlässt.

In den folgenden Abschnitten werden die Formeln für die Basisverfügbarkeit um die skizzierten Aspekte erweitert, um eine detaillierte Analyse von Kademlia unter realistischen Annahmen zu ermöglichen.
Zunächst wird das Formelwerk auf eine feinere Granularität (minütliche Betrachtungsweise) angepasst und um Angriffe zum Publikationszeitpunkt erweitert. Die Methode des Transfer on Join wird diskutiert.
Es folgt die formale Beschreibung des Republishing am Ende jeder Stunde und damit die Vervollständigung des Formelwerks. Damit kann die Verfügbarkeitswahrscheinlichkeit zu jedem beliebigen Zeitpunkt seit Publikation eines Datensatzes berechnet werden.
Im Anschluss erfolgt die ausführliche Analyse auf Basis der entwickelten Formeln. Insbesondere wird die Langzeitverfügbarkeit unter Berücksichtigung multipler Republishing-Zeitpunkte quantifiziert.

4.3.4 Zeitverhalten bei minütlicher Betrachtung

Es gelten für die Untersuchung des Zeitverhaltens folgende Annahmen:

- Die bisher monolithisch betrachtete Periode von einer Stunde wird in T wiederum diskrete Zeitschritte aufgeteilt, wobei die „Länge“ jeder Einheit $\frac{1}{T}$ ist. Als grundlegende Zeiteinheiten werden Minuten betrachtet, d.h. $T = 60$.
- Die Fluktuationswahrscheinlichkeit pro Knoten wird folglich minütlich angegeben, d.h. es wird p_{o_m} verwendet.
- Knoten ändern ihre „Gesinnung“ nicht: waren sie einmal Angreifer, handeln sie über ihre gesamte Online-Zeit hinweg böswillig, ansonsten bleiben sie gut, handeln also regelkonform.

Die Granularität der zeitlichen Betrachtung wird also verfeinert, indem Minuten als neue Zeiteinheit zugrunde gelegt werden. Ein Zeitpunkt t bezeichnet das Ende der t. Minute und damit zugleich den Beginn der $t+1$. Minute.

Der Reihe nach werden drei Phasen im Lebenszyklus der Replika untersucht:

1. *Publikation:* In $t = 0$ wird ein Datensatz veröffentlicht, indem der Einsteller k `STORE`-Requests an die Knoten aus $\mathcal{N}_b$ versendet. Die Anzahl $\tilde{k}$ der tatsächlich entstehenden Replika ist in Abhängigkeit von p_a wahrscheinlichkeitsverteilt.
2. *Fluktuation:* In den jeweils 60 Minuten zwischen der Publikation eines Datensatzes und dem ersten Republishing bzw. zwischen zwei Republishing-Zeitpunkten gehen Replika in Abhängigkeit von der Fluktuationswahrscheinlichkeit p_{o_m} verloren.
3. *Re-Publikation (Republishing):* Alle 60 Minuten wird der Datensatz von den verbleibenden Replikatbesitzern wieder veröffentlicht, indem an alle Knoten $\in \mathcal{N}_b$ ein `STORE`-Request übermittelt wird.

Die Phasen 2 und 3 werden nach dem ersten Republishing alternierend durchlaufen.

4.3.4.1 Nomenklatur

Alle Formeln werden in einer Indexnotation als Liste geschrieben. Dabei werden folgende Indices verwendet:

- Bezeichner $fine$: Formel berücksichtigt die feinere (minütliche) Granularität
- Bezeichner rp: Formel berücksichtigt Republishing
- Bezeichner $init$: Formel betrifft Publikationszeitpunkt
- Parameter p_a, p_{o_m}, k
- Zeitpunkt t

Sie dienen der Spezifikation der Funktion und ihren zugrunde liegenden Parametern, ggf. mit deren Belegung.

4.3.4.2 Phase 1 und 2: Publikation und Fluktuation

4.3.4.2.1 Formalisierung Phase 1: Veröffentlichung von Datensätzen

Im Gegensatz zur ersten Annahme aus 4.3.3.1 entstehen in einem System mit Angreifern nur mit einer geringen Wahrscheinlichkeit k Replika. Die Formel zur Beschreibung der Verteilung der tatsächlich erzeugten Replika $\tilde{k}$ in Abhängigkeit von p_a lautet folglich:

$$p(\tilde{k} = i)_{init,(p_a,k)} = \binom{k}{i}(1 - p_a)^i \cdot (p_a)^{k-i} \tag{4.7}$$

Der Parameter p_{o_m} geht hier noch nicht ein, da nur ein einziger Zeitpunkt - der Publikationszeitpunkt - betrachtet wird.
Der Erwartungswert für die entstandenen Replika ergibt sich dann wie folgt:

$$E(\tilde{k})_{init,(p_a,k)} = \sum_{i=0}^{k} i \cdot p(\tilde{k} = i)_{init,(p_a,k)} \tag{4.8}$$

$$= (1 - p_a) \cdot k \tag{4.9}$$

4.3.4.2.2 Formalisierung Phase 2: Knotenfluktuation

Ein Datensatz wird zum Zeitpunkt $t = 0$ publiziert. Während der ersten Stunde, d.h. $t = 1, \ldots, 60$, gehen Knoten aus $\mathcal{N}_b$ mit einer gewissen Wahrscheinlichkeit offline und werden durch neue Peers, welche noch keine Replika besitzen, ersetzt.
Allgemein gilt folgende rekursive Formel für die Wahrscheinlichkeit von i Replika im Zeitpunkt t:

$$p(\tilde{k} = i)_{fine,(p_{o_m},t,k)} = \sum_{j=0}^{k-i} \binom{i+j}{i} \cdot (1 - p_{o_m})^i \cdot p_{o_m}^j \cdot p(\tilde{k} = i + j)_{fine,(p_{o_m},t-1,k)} \tag{4.10}$$

Die Formel setzt sich wie folgt zusammen:

- $p(\tilde{k} = i + j)_{fine,(p_{o_m},t-1,k)}$: Wahrscheinlichkeit, dass es in der Vorperiode $i + j$ Replika gegeben hat
- $\binom{i+j}{i} \cdot (1 - p_{o_m})^i \cdot p_{o_m}^j$: Wahrscheinlichkeit, dass von diesen $i + j$ Replika am Ende der aktuellen Periode noch i Replika übrig bleiben, d.h. j gehen durch Fluktuation verloren
- $\sum_{j=0}^{k-i}$: Die Wahrscheinlichkeit, dass aus $i + j$ Replika der Vorperiode jetzt noch i existieren, wird für alle j bestimmt, wobei $i + j = i..k$. Diese Werte werden addiert, um die Gesamtwahrscheinlichkeit für i Replika in der aktuellen Periode zu bestimmen.

Formel 4.10 greift damit stets auf die Wahrscheinlichkeitsverteilung für die Zufallsvariable $\tilde{k}$ aus der Vorperiode zurück. Insbesondere gilt für $t = 1$ unter direkter Verwendung von Formel 4.7:

$$p(\tilde{k} = i)_{fine,(p_{o_m},p_a,t=1,k)} = \sum_{j=i}^{k} p(\tilde{k} = j)_{init,(p_a,k)} \cdot \binom{j}{i} \cdot (1 - p_{o_m})^i \cdot (p_{o_m})^{j-i} \tag{4.11}$$

Die Angriffswahrscheinlichkeit p_a geht also nur im Zeitpunkt $t = 0$ und später zu den Republishing-Zeitpunkten, siehe Abschnitt 4.3.4.3.2, in die Berechnung ein.

4.3.4.2.3 Transfer on Join

In der Phase zwischen Publikation des Datensatzes und der ersten Wiederveröffentlichung findet standardmäßig Transfer on Join statt. Bemerkt ein bestehender Peer $n_e \in \mathcal{N}_b$, dass ein Knoten n_i im ID-Bereich der k nächsten Knoten zur FileID b durch einen Join hinzugekommen ist, so führt er zwei Schritte aus:

1. n_e prüft, ob er der absolut nächste Knoten zur FileID b ist (der neue Knoten n_i wird dabei nicht berücksichtigt, auch wenn er eine geringere Distanz zu b haben sollte). Nur dann initiiert n_e einen Transfer on Join.
2. n_e sendet einen `STORE`-Request für den Datensatz mit FileID b an n_i.

Dadurch ändert sich die Wahrscheinlichkeitsverteilung von $\tilde{k}$ und auch der Erwartungswert zum Positiven. Der Erfolg dieses Verfahrens hängt davon ab, mit welcher Wahrscheinlichkeit und wie schnell neu hinzukommende Knoten vom für den Transfer on Join zuständigen Peer entdeckt werden und ob dieser sowie der jeweilige neue Knoten gut sind. Wenn durch Fluktuation *alle* Replikainhaber das Netz verlassen haben, ist keine Entstehung neuer Replika und damit kein Transfer on Join mehr möglich.

Transfer on Join wird aber in die vorliegende Analyse nicht explizit einbezogen, sondern nur das Republishing berücksichtigt. Transfer on Join verlegt nur die ohnehin beim Republishing stattfindende Übertragung eines Replikats an einen neu hinzugekommenen Knoten zeitlich nach vorne, sofern dieser vor dem nächsten Republishing-Zeitpunkt entdeckt wird. Knoten, welche bereits online sind und erst durch den Ausfall von Peers mit geringerer Distanz zu b in $\mathcal{N}_b$ aufgenommen werden, erhalten immer erst zum Republishing-Zeitpunkt ein Replikat. Für sie existiert also keine Entsprechung des Transfer on Join.
Beim Republishing werden mit Sicherheit *alle* Peers aus $\mathcal{N}_b$ erreicht, da ein re-publizierender Knoten vor dem Versand der `STORE`-Requests eine Aktualisierung seiner Routing-Tabellen (Bucket Refresh, siehe 2.5.3) durchführt und damit die aktuelle Menge $\mathcal{N}_b$ ermittelt.
Es ist nun also möglich, die Betrachtung des Transfer on Join aus Gründen der Vereinfachung des Formelwerks auszulassen und stattdessen nur das Republishing, in dem *alle* $k-1$ nächsten Knoten zu einem Datensatz *auf einmal* Replika vom re-publizierenden Knoten angeboten bekommen, zu betrachten.

Die durch Analyse des Republishing erlangten quantitativen Aussagen legen also eine *untere Grenze* für die Verfügbarkeitswahrscheinlichkeit fest: Durch die Knotenfluktuation zwischen den Republishing-Zeitpunkten in einem Szenario *ohne* Berücksichtigung von Transfer on Join verschlechtert sich die Verfügbarkeitswahrscheinlichkeit eines Datensatzes schneller als in einem realen Szenario *mit* Transfer on Join.

4.3.4.3 Phase 3: Republishing

4.3.4.3.1 Ablauf

Ein Knoten, der zu einem bestimmten Zeitpunkt das Republishing des bei ihm gespeicherten Datensatzes initiiert, führt folgende Schritte aus:

1. Er aktualisiert über einen Bucket Refresh seine Routing-Tabellen, so dass er sich über die k nächsten Knoten zu b sicher ist.

2. Er sendet ein `STORE`-Request an alle weiteren Knoten aus $\mathcal{N}_b$, d.h. an $k-1$ Peers. Der Fall, dass er durch neu hinzugekommene Knoten mit kleinerer Distanz zu b aus $\mathcal{N}_b$ „gedrängt“ wurde, wird vernachlässigt.

Jeder Knoten aus $\mathcal{N}_b$, der kein Angreifer ist, republiziert jeweils in 60-Minuten-Abständen. Das erste Republishing erfolgt 60 Minuten nach Empfang des Replikats durch einen `STORE`-Request.

Alle Knoten, die zum Publikationszeitpunkt $t = 0$ zu $\mathcal{N}_b$ gehören, empfangen in diesem Zeitpunkt ein `STORE`-Request des Einstellers. Da Transfer on Join nicht berücksichtigt

ist, wird angenommen, dass alle Replikatbesitzer dieselbe Taktung haben, d.h. übereinstimmende Republishing-Zeitpunkte, da sie ihre Replika zum gleichen Zeitpunkt erhalten haben. Das erste Republishing findet damit in $t = 60$ statt, das zweite in $t = 120$ usw.

4.3.4.3.2 Formalisierung

Der erste Republishing-Zeitpunkt nach der Publikation wird mit t_r bezeichnet, die weiteren mit $t_{(r+1)}, t_{(r+2)}$ usw. Es wird idealisierend angenommen, dass das Republishing verzögerungsfrei erfolgt, d.h. das Verschicken und Empfangen der `STORE`-Requests geschieht gleichzeitig zum Ende der t_r-ten, $t_{(r+1)}$-ten, ... Minute.

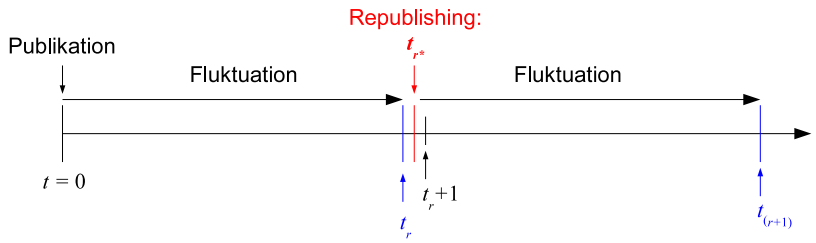


Abbildung 4.4: Zeitstrahl für die Republishing-Zeitpunkte

Als Hilfestellung wird der Zeitpunkt des Versendens und Empfangens der `STORE`-Requests, von t_r getrennt betrachtet und als t_{r^*} bezeichnet. Von t_{r^*} bis zum Zeitpunkt $t_r + 1$ findet dann normale Knotenfluktuation statt, wobei als Ausgangspunkt die neue Wahrscheinlichkeitsverteilung aus t_{r^*} verwendet wird. In t_r (und analog in späteren Republishingzeitpunkten) muss zunächst eine Fallunterscheidung vorgenommen werden:

- Sind zum Zeitpunkt t_r keine Replika mehr vorhanden, kann nicht mehr republiziert werden. Der Datensatz ist verloren.
- Ist in t_r noch mindestens ein Replikat im Netz vorhanden, wird das Republishing durchgeführt - prinzipiell einmal durch jeden Besitzer eines Replikats. Da aber aufgrund gleicher Rahmenbedingungen das Ergebnis bei jedem dieser zeitgleichen Re-Publikationsvorgänge identisch ist, genügt es, das Republishing *eines einzigen* dieser Knoten zu betrachten.

Sind in t_r keine Replika mehr vorhanden, gilt also:

$$p(\tilde{k} = i)_{fine,rp,(p_{om},p_a,t_{r^*},k)} = \begin{cases} 1, & \text{wenn i} = 0 \\ 0 & \text{sonst} \end{cases} \tag{4.12}$$

Dieses Ereignis tritt mit der Wahrscheinlichkeit $p(\tilde{k} = 0)_{fine,(p_{om},p_a,t_r,k)}$ ein.
Im gegenteiligen Fall (Replikazahl in t_r ist > 0) wird ein Republishing stattfinden. Die Wahrscheinlichkeit hierfür ist:

$$p(\tilde{k} \geq 1)_{fine,(p_{om},p_a,t_r,k)} = 1 - p(\tilde{k} = 0)_{fine,(p_{om},p_a,t_r,k)}$$

Für die Analyse des Republishing ist die Aufteilung der Menge $\mathcal{N}_b$ in vier disjunkte Teilmengen notwendig:

- $\mathcal{N}_{old,good}$: Menge aller Knoten aus $\mathcal{N}_b$, die mindestens seit dem letzten Republishing-Zeitpunkt online geblieben und keine Angreifer sind. Die Mächtigkeit von $\mathcal{N}_{old,good}$ ist zugleich die Anzahl der zum Zeitpunkt t_r existierenden[8] Replika in der Menge der zuständigen Knoten.
- $\mathcal{N}_{old,adversary}$: Menge aller Knoten aus $\mathcal{N}_b$, die seit dem letzten Republishing-Zeitpunkt online geblieben und Angreifer sind.
- $\mathcal{N}_{new,good}$: Menge aller Knoten aus $\mathcal{N}_b$, die seit dem letzten Republishing-Zeitpunkt zu $\mathcal{N}_b$ hinzugekommen und keine Angreifer sind.
- $\mathcal{N}_{new,adversary}$: Menge aller Knoten aus $\mathcal{N}_b$, die seit dem letzten Republishing-Zeitpunkt zu $\mathcal{N}_b$ hinzugekommen und Angreifer sind.

Alle Knoten $\in \mathcal{N}_{old,good}$ republizieren in t_{r^*}, wobei jeder dieser Knoten auch nach dem Republishing weiterhin ein Replikat besitzen wird. Eine Änderung in der Replikazahl hängt also vom Verhalten der Knoten in der Menge

$$\mathcal{N}_{\neg(old,good)} = \mathcal{N}_b \setminus \mathcal{N}_{old,good}$$

ab. Die Wahrscheinlichkeit, dass ein zufällig ausgewählter Knoten n_i aus $\mathcal{N}_{old,good}$ ist, beträgt

$$p(n_i \in \mathcal{N}_{old,good}) = (1 - p_{o_m})^{60} \cdot (1 - p_a),$$

nämlich die Wahrscheinlichkeit, dass der Knoten in den letzten 60 Perioden online geblieben ist, multipliziert mit der Wahrscheinlichkeit, dass er kein Angreifer ist.
Folglich ist die Wahrscheinlichkeit dafür, dass ein zufällig aus $\mathcal{N}_b$ ausgewählter Knoten aus $\mathcal{N}_{\neg(old,good)}$ ist:

$$p(n_i \in \mathcal{N}_{\neg(old,good)}) = 1 - p(n_i \in \mathcal{N}_{old,good}).$$

Die betrachtete Menge ist also nun $\mathcal{N}_{\neg(old,good)}$, da die Knoten aus $\mathcal{N}_{old,good}$ die Anzahl der Replika beim Republishing nicht beeinflussen: sie besaßen vorher Replika und behalten sie auch weiterhin.
Um die Wahrscheinlichkeit der Neuentstehung von Replika zu quantifizieren, muss nun mit *bedingten Wahrscheinlichkeiten* gerechnet werden - die Wahrscheinlichkeit, dass ein zufällig ausgewählter Knoten zu $\mathcal{N}_{new,good}$ gehört, wird unter der Prämisse bestimmt, dass er *nicht* zu der Knotenmenge gehört, welche bereits Replika besitzt.

Jeder Knoten aus $\mathcal{N}_{\neg(old,good)}$ erhält nun in t_{r^*} definitiv ein `STORE`-Request vom republizierenden Knoten. Er wird daraufhin ein Replikat erzeugen, sofern er $\in \mathcal{N}_{new,good}$ ist, sonst nicht.
Um diese Wahrscheinlichkeit zu berechnen, ist die allgemeine Formel für bedingte Wahrscheinlichkeiten heranzuziehen:

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

[8] Nicht auf Knoten aus $\mathcal{N}_b$ existierende Replika, die beispielsweise durch Caching oder durch alte Knoten, die mit neuer FileID online kommen, aber ihre vorherigen Datensätze weiterhin anbieten, können nicht sicher über einen Lookup aufgefunden werden und sind daher hier nicht von Belang.

mit den Ereignissen A: $n \in \mathcal{N}_{new,good}$ und B: $n \in \mathcal{N}_{\neg(old,good)}$. Es ergibt sich dann:

$$\begin{aligned} p(n \in \mathcal{N}_{new,good} | n \in \mathcal{N}_{\neg(old,good)}) = & \; p(n \; adds \; 1) \\ = & \; \frac{(1-(1-p_{o_m})^{60}) \cdot (1-p_a)}{1-((1-p_{o_m})^{60} \cdot (1-p_a))}. \end{aligned} \tag{4.13}$$

$p(n \; adds \; 1)$ bezeichnet also die Wahrscheinlichkeit, dass ein zufällig ausgewählter Knoten n aus der Menge der Knoten ohne Replika $\mathcal{N}_{\neg(old,good)}$ durch das Republishing ein Replikat erzeugen wird und somit die Gesamtzahl der Replika um 1 erhöht.
Die Formel für die Wahrscheinlichkeit von $i > 0$ Replika in t_{r^*} setzt sich damit wie folgt zusammen:

$$\begin{aligned} p(\tilde{k} = i)_{fine,rp,(p_{o_m},p_a,t_{r^*},k)} = & \overbrace{\sum_{j=1}^{i} p(\tilde{k} = j)_{fine,(p_{o_m},p_a,t_r,k)}}^{I} \cdot \\ & \underbrace{\binom{k-j}{i-j} \cdot p(n \; adds \; 1)^{i-j} \cdot (1-p(n \; adds \; 1))^{k-i}}_{II} \end{aligned} \tag{4.14}$$

Es wird also summiert über alle Replikazahlen j, $1 \leq j \leq i$ aus Zeitpunkt t_r. Die Wahrscheinlichkeit, dass mit j Replika in t_r nach dem Republishing i Replika existieren, ergibt sich zu der Wahrscheinlichkeit, dass $i-j$ von den $k-j$ Knoten aus $\mathcal{N}_{\neg(old,good)}$ nach Empfang der `STORE`-Requests jeweils ein neues Replikat erzeugen.
In Teil I der Formel 4.14 werden nur Knoten aus $\mathcal{N}_{old,good}$ betrachtet (die aktuellen Replikainhaber), in Teil II nur die Knoten, die *nicht* aus $\mathcal{N}_{old,good}$ sind. Die in den beiden Formelteilen betrachteten Teilmengen von $\mathcal{N}_b$ sind somit disjunkt.

Wenn man nun die Verteilung für $i = 0$ hinzunimmt, ergibt sich als Gesamtformel:

$$p(\tilde{k} = i)_{fine,rp,(p_{o_m},p_a,t_{r^*},k)} = \begin{cases} p(\tilde{k} = 0)_{fine,rp,(p_{o_m},p_a,t_r,k)}, & \text{wenn i = 0} \\ \sum_{j=1}^{i} p(\tilde{k} = j)_{fine,(p_{o_m},p_a,t_r,k)} \cdot & \\ \binom{k-j}{i-j} \cdot p(n \; adds \; 1)^{i-j} \cdot & \\ (1-p(n \; adds \; 1))^{k-i} & \text{sonst} \end{cases} \tag{4.15}$$

Die entstehende Wahrscheinlichkeitsverteilung aus t_{r^*} ist danach noch der normalen Fluktuation ausgesetzt. Damit ergibt sich die Verteilung in $t_r + 1$ einfach gemäß Formel 4.10 unter Einsetzung der Werte aus t_{r^*} statt derer aus t_r:

$$p(\tilde{k} = i)_{fine,rp,(p_{o_m},p_a,t,k)} = \begin{cases} \sum_{j=0}^{k-i} \binom{i+j}{i} \cdot (1-p_{o_m})^i \cdot p_{o_m}^j \cdot & \\ p(\tilde{k} = i+j)_{fine,rp,(p_{o_m},p_a,t_{r^*},k)}, & \text{wenn } t \, mod \, 60 = 1, t \neq 1 \\ p(\tilde{k} = i)_{fine,(p_{o_m},p_a,t,k)} & \text{sonst} \end{cases} \tag{4.16}$$

Für jeden Zeitpunkt nach Republishing $t_{(r+q)} + 1$, $q \in \mathbb{N}_0$ wird also die Verteilung aus dem jeweiligen Republishing-Zeitpunkt $t_{(r+q)^*}$ verwendet.

4.3.4.4 Analyse

Details zur Umsetzung der für die Analyse verwendeten Formeln in Berechnungsroutinen eines Computerprogramms finden sich in Appendix A.
Abbildung 4.5 stellt die Wahrscheinlichkeitsverteilung von $\tilde{k}$ zum Publikationszeitpunkt für verschiedene Angriffswahrscheinlichkeiten dar (Phase 1).

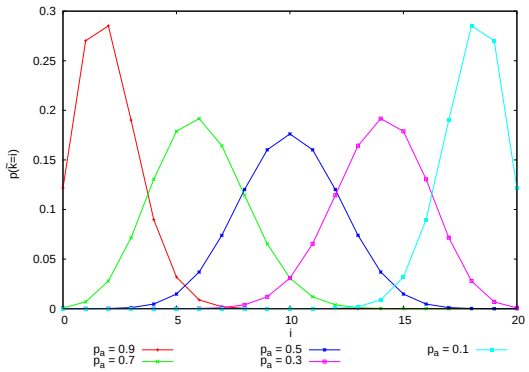
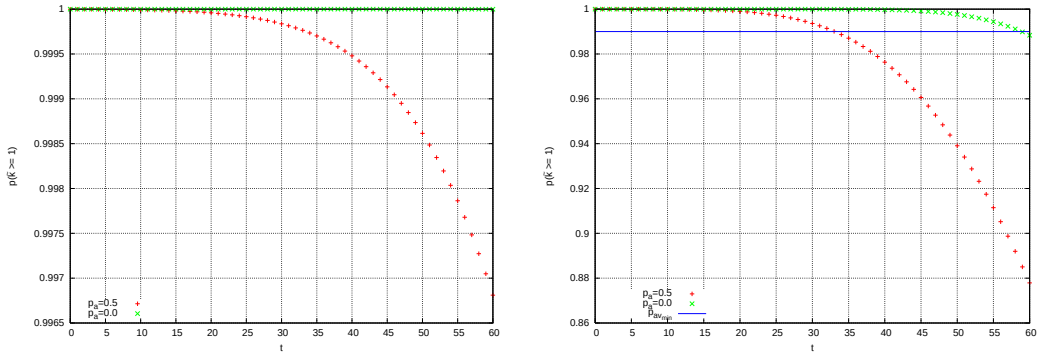


Abbildung 4.5: Wahrscheinlichkeitsverteilung im Publikationszeitpunkt $p(\tilde{k} = i)_{init,(p_a,k)}$ bei variiertem p_a in $t = 0$, zur besseren Lesbarkeit stetig interpoliert; $k = 20$

In Abbildung 4.6 wird die Entwicklung der Verfügbarkeit des Datensatzes in einem System mit und ohne Angreifer bei hoher und moderater Fluktuationswahrscheinlichkeit über die ersten 60 Minuten hinweg dargestellt (Phase 2).

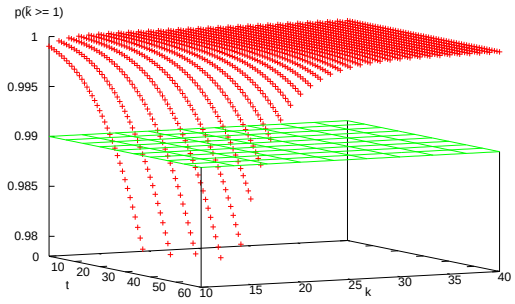


(a) Moderate Fluktuation: $p_{o_m} = 0.0115$ (b) Starke Fluktuation: $p_{o_m} = 0.0265$

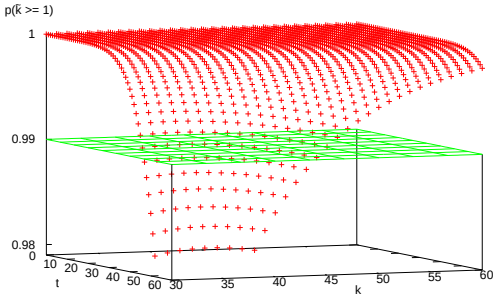
Abbildung 4.6: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{fine,(p_{o_m},t,k)}$ mit und ohne Angreifer in $t = 0..60$; $k = 20$

Bei moderater Fluktuation bleibt die Verfügbarkeitswahrscheinlichkeit auch bei $p_a = 0.5$ über die vollen 60 Minuten deutlich größer als $p_{av_{min}}$. Bei hoher Fluktuation hingegen ist die Verfügbarkeit des Datensatzes unter Verwendung des in [MaM02] propagierten Replikationsfaktors $k = 20$ selbst bei einer Angriffswahrscheinlichkeit von 0 nicht für eine volle Stunde gegeben (vgl. hierzu auch Abb. 4.3), sondern nur für 58 Minuten. Bei

$p_a = 0.5$ wird die Verfügbarkeitsschwelle von $p_{av_{min}} = 0.99$ sogar schon bei 34 Minuten unterschritten.
In Abbildung 4.7 wird die Verfügbarkeitswahrscheinlichkeit eines Datensatzes über eine Stunde hinweg für verschiedene k aufgetragen. Offenbar ist bei starker Fluktuation eine ausreichende Verfügbarkeitswahrscheinlichkeit am Ende der Stunde nur bei hohen Replikazahlen (gemäß Abb. 4.7(b) ab $k = 44$) gegeben. Bei moderater Fluktuation genügt ein geringerer Replikationsfaktor, bereits für $k = 17$ bleibt die Verfügbarkeitswahrscheinlichkeit in der betrachteten Stunde ausreichend hoch (Abb. 4.7(a)).



(a) Moderate Fluktuation: $p_{o_m} = 0.0115$

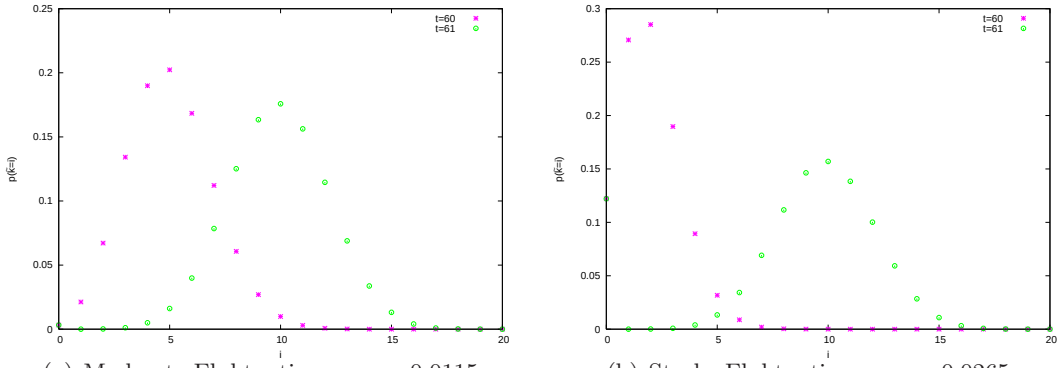


(b) Starke Fluktuation: $p_{o_m} = 0.0265$

Abbildung 4.7: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{fine,(p_{o_m},t,k)}$ bei variiertem k in $t = 0..60$; $p_a = 0.5$

In $t = 60, 120, \ldots$ liegen die festgelegten Republishing-Zeitpunkte. In diesen ändert sich die Wahrscheinlichkeitsverteilung von $\tilde{k}$. Abbildung 4.8 zeigt dies am Beispiel der Wahrscheinlichkeitsverteilung in $t = 60$, d.h. unmittelbar vor dem Republishing, und in $t = 61$, d.h. unmittelbar danach, für die exemplarische Belegung $k = 20$. Es ist zu beachten, dass die Wahrscheinlichkeit für $\tilde{k} = 0$ in $t = 60$ und $t = 61$ stets identisch ist (siehe Formel 4.15). Abgesehen davon ist die Wahrscheinlichkeitsverteilung in $t = 61$ beinahe deckungs-

gleich mit der Verteilung zum Publikationszeitpunkt (vgl. Abb. 4.5 für $p_a = 0.5$).



(a) Moderate Fluktuation: $p_{o_m} = 0.0115$ (b) Starke Fluktuation: $p_{o_m} = 0.0265$

Abbildung 4.8: Wahrscheinlichkeitsverteilung $p(\tilde{k} = i)_{fine,rp,(p_{o_m},p_a,t,k)}$ in $t = 60$, $t = 61$; $k = 20, p_a = 0.5$

Durch diese Verschiebung in der Wahrscheinlichkeitsverteilung steigt nach dem Republishing-Zeitpunkt die Wahrscheinlichkeit für 0 Replika zunächst langsamer an als zuvor. Durch Fluktuation verschiebt sich die Wahrscheinlichkeitsverteilung stetig in Richtung geringerer Replikazahlen, d.h. es wird immer wahrscheinlicher, dass nur geringe Replikazahlen existieren, und unwahrscheinlicher, dass viele Replika vorhanden sind.
Unmittelbar nach dem Republishing ist die Wahrscheinlichkeit für die Existenz vieler Replika nun wieder höher als unmittelbar zuvor. Verschiebt sich nun diese neue Verteilung mit der Zeit wieder in Richtung geringerer Replikazahlen, steigt die Unverfügbarkeitswahrscheinlichkeit erst sehr langsam, später immer schneller an. Abbildung 4.9 zeigt die Auswirkungen dieser Entwicklung auf die Gesamtverfügbarkeit exemplarisch für $k = 20$ und $k = 40$ bei starker Fluktuation.

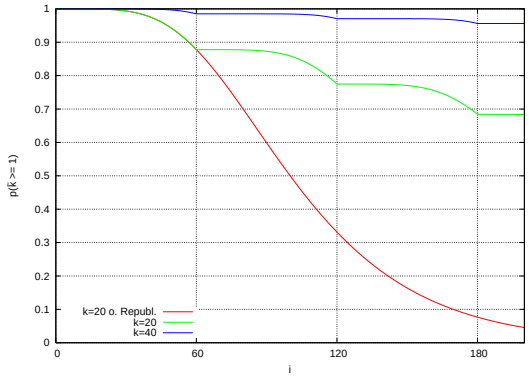


Abbildung 4.9: Verfügbarkeitswahrscheinlichkeit für $t = 0..200$ mit Republishing: $p(\tilde{k} \geq 1)_{fine,rp,(p_{o_m},p_a,t,k)}$ und ohne: $p(\tilde{k} \geq 1)_{fine,(p_{o_m},p_a,t,k)}$; $k = 20, 40$, $p_a = 0.5$, $p_{o_m} = 0.0265$, Verlauf stetig interpoliert

In den Republishing-Zeitpunkten wird der Verfall der Verfügbarkeitswahrscheinlichkeit

stark verlangsamt, es entsteht ein stufenförmiger Verlauf. Die Abbildung zeigt zum Vergleich auch die Verfügbarkeitsentwicklung im gleichen Zeitfenster ohne Republishing.
Datensätze in einer AAI haben unterschiedliche Gültigkeitsdauern, für die ihre Verfügbarkeit gesichert werden muss. Bei der Kombination von Zertifikaten mit Statusinformationen wie in CRS müssen letztere nur 24 Stunden lang verfügbar bleiben, Zertifikate aber länger. Werden hingegen Rückrufe eingesetzt, sind diese genauso lange vorzuhalten wie die zugehörigen Zertifikate.
In beiden Fällen ist die maximale Vorhaltezeit durch die Lebensdauer der Zertifikate bestimmt. Im Folgenden wird daher der Bestimmung des geeigneten Wertes für k nachgegangen, wenn Zertifikate langfristig verfügbar bleiben sollen. Dafür ist die Frage zu beantworten, wie genau Replikationsfaktor k und Fluktuationswahrscheinlichkeit p_{o_m} zusammenhängen. Nur so kann ein geeigneter Replikationsfaktor für ein Netz mit einer bestimmten Fluktuationswahrscheinlichkeit bestimmt werden.
Als zumutbar wird eine Vorhaltezeit der Datensätze im Netzwerk von 30 Tagen angesehen. Nach Ablauf dieser Frist muss der Einsteller des Datensatzes (Zertifizierungsstelle, Inhaber, Status Authority) ihn neu publizieren. Abbildung 4.10 trägt die Verfügbarkeitswahrscheinlichkeit nach 30 Tagen, d.h. 43200 Minuten in Abhängigkeit von den Parametern k und p_{o_m} auf.

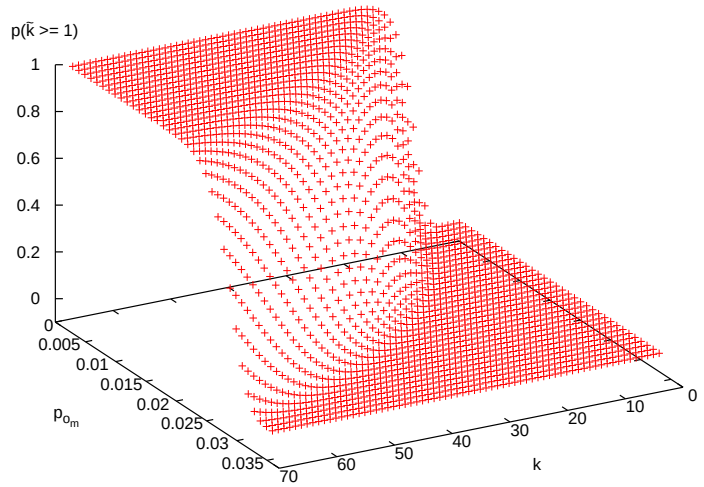


Abbildung 4.10: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{fine,rp,(p_{om},p_a,t,k)}$ in $t = 43200$ mit k, p_{o_m} variiert; $p_a = 0.5$

Unter der Voraussetzung, dass für unterschiedliche Anwendungsfälle die Fluktuationswahrscheinlichkeit variieren wird, kommt die Frage auf, welcher Replikationsfaktor k für die gegebenen Rahmenbedingungen optimal ist. Es ist unmittelbar einsichtig, dass der durch hohe Replikationsfaktoren entstehende Aufwand für die Selbstverwaltung des Netzes - insbesondere bezüglich des Bandbreiten- und Speicherplatzbedarfs - zu minimieren

ist. Daher ist es wenig sinnvoll, in Netzen mit geringer Fluktuation hohe Werte für k einzusetzen. Umgekehrt wiederum ist es aber auch riskant, mit einem zu geringen Replikationsfaktor nicht die geforderte Verfügbarkeit gewährleisten zu können.
Dementsprechend ist der Replikationsfaktor in Abhängigkeit von der Fluktuationswahrscheinlichkeit zu optimieren. Hierfür wird zunächst nur der Bereich betrachtet, in dem die Verfügbarkeitswahrscheinlichkeit ≥ 0.99 liegt, wie in Abbildung 4.11 dargestellt. Die eingezogene Ebene ist die Verfügbarkeitsgrenze $p_{av_{min}} = 0.99$.

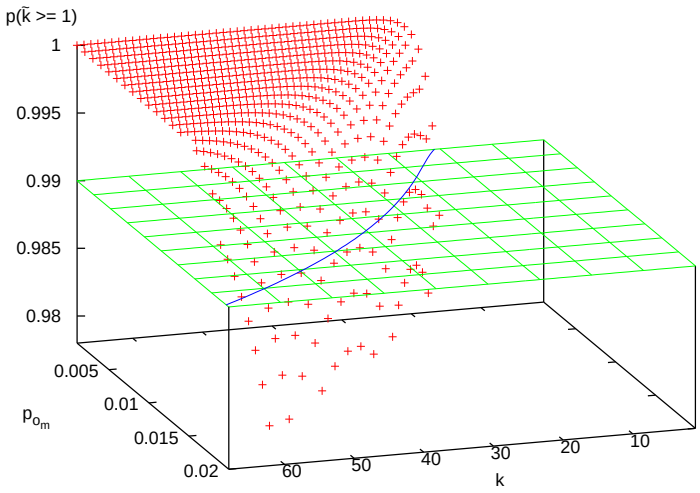


Abbildung 4.11: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{fine,rp,(p_{o_m},p_a,t,k)}$ in $t = 43200$ mit k, p_{o_m} variiert; $p_a = 0.5$

Nun wird der Bereich betrachtet, in welchem sich die 0.99-Ebene und die Funktion

$$\chi : [0,1] \times \mathbb{N} \rightarrow [0,1] \tag{4.17}$$

$$\chi(p_{o_m}, k) = p(\tilde{k} \geq 1)_{fine,rp,(p_{o_m},p_a=0.5,t=43200,k)} \tag{4.18}$$

schneiden. Da die Funktion nicht stetig ist, muss der Verlauf dieser Höhenlinie (blaue Linie in Abb. 4.11) approximiert werden[9]. Dafür wird für jedes k die kleinste Fluktuationswahrscheinlichkeit p_{o_m} bestimmt, mit der gerade noch $\chi(p_{o_m} \geq 0.99)$ gilt. Die Genauigkeit der Berechnung beträgt 5 bzw. 6 Nachkommastellen für p_{o_m}.
Nach der Ermittlung der approximierten Punkte wird der Verlauf der Höhenlinie in einem zweidimensionalen Koordinatensystem aufgetragen, siehe Abb. 4.12. Alle Wertepaare (k, p_{o_m}) unterhalb der Linie führen nach 30 Tagen zu einer Verfügbarkeitswahrscheinlichkeit > 0.99. Die Werte auf der Linie zeigen an, welche Fluktuationswahrscheinlichkeit von dem gegebenen k gerade noch ausgeglichen wird. Anhand der Graphik kann somit

[9] siehe Appendix A.2

auch im Gegenzug für jedes gegebene p_{o_m} der kleinste ausreichende Wert für den Replikationsfaktor gefunden werden: dies ist das Optimum für k.

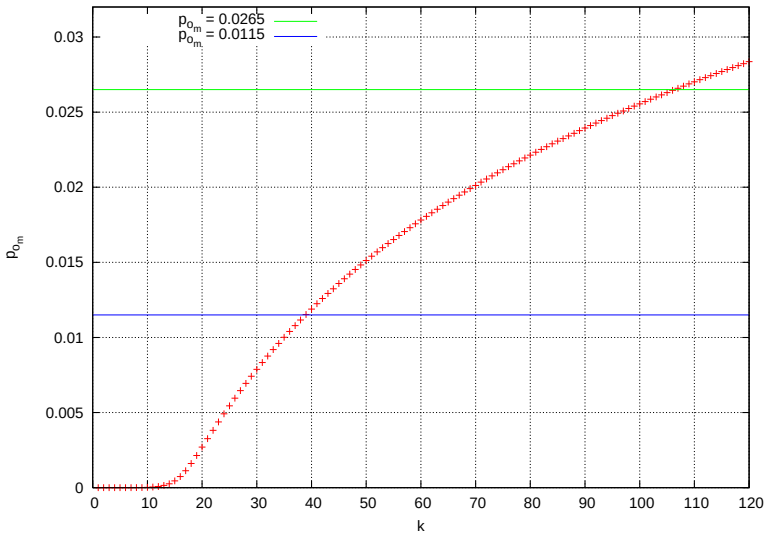
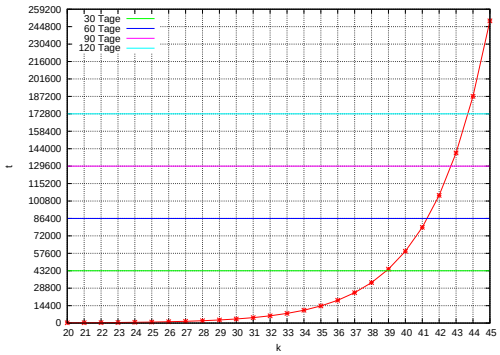
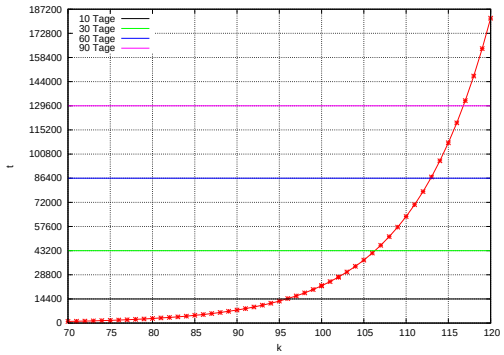


Abbildung 4.12: Approximierte 0.99-Höhenlinie von $\chi(k, p_{o_m})$ für $t = 43200$

Für ein moderat fluktuierendes Netzwerk liegt der optimale Replikationsfaktor für 30 Tage Vorhaltezeit also bei $k = 39$, für ein stark fluktuierendes Netz bei $k = 107$.
Weiterhin kann der optimale Replikationsfaktor für eine feste Fluktuationsrate auch in Abhängigkeit von der gewünschten Vorhaltezeit bestimmt werden. Abbildung 4.13(a) stellt diesen Zusammenhang für $p_{o_m} = 0.0115$ dar. Der Kurvenverlauf beschreibt die von k abgedeckte Zeitspanne zwischen Publikation und Unterschreiten der minimal vorausgesetzten Verfügbarkeitswahrscheinlichkeit $p_{av_{min}} = 0.99$.



(a) Moderate Fluktuation: $p_{o_m} = 0.0115$



(b) Starke Fluktuation: $p_{o_m} = 0.0265$

Abbildung 4.13: Abgedeckte Zeitspanne t in Abhängigkeit von k, stetig interpoliert

Mit einem Replikationsfaktor von $k = 44$ (Zusatzaufwand gegenüber Verfügbarkeit für 30 Tage $\approx 13\%$) bleibt ein Datensatz bei moderater Fluktuation beispielsweise für mehr als 120 Tage verfügbar.

Für Netzwerke mit starker Fluktuation hingegen ist der Einsatz hoher Replikationsfaktoren (z.B. $k = 107$ bei Vorhaltezeit von 30 Tagen, siehe Abbildung 4.13(b)) notwendig. Alternativ können aber auch die Republishing-Intervalle verkürzt werden: Wie Abbildung 4.9 auf Seite 96 illustriert, sinkt nach jedem Republishing die Verfügbarkeitswahrscheinlichkeit erst langsam, dann immer schneller. Dieser Umstand kann ausgenutzt werden, indem die Abstände zwischen Republishing-Zeitpunkten verringert werden, z.B. auf 30 Minuten. Abbildung 4.14 stellt die Verfügbarkeitswahrscheinlichkeiten für die Republishing-Intervalle von 30 und 60 Minuten für verschiedene k bei starker Fluktuation gegenüber. Bei 30-Minuten-Intervallen genügt bereits ein Replikationsfaktor von $k = 47$.

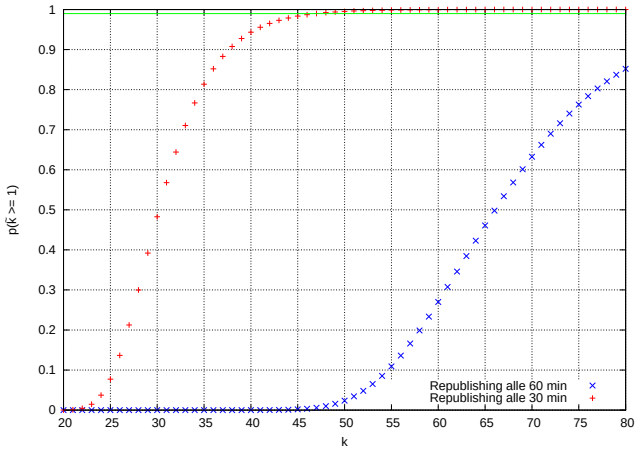


Abbildung 4.14: Verfügbarkeitswahrscheinlichkeit $p(\tilde{k} \geq 1)_{fine,rp,(p_{o_m},p_a,t,k)}$ für verschiedene Republishing-Intervalle, k variiert, $p_{o_m} = 0.0265$

4.3.4.5 Variable Republishing-Zeitpunkte

Bisher wurde vorausgesetzt, dass nur *genau ein* Knoten nach Ablauf von *genau* einer, zwei, ... Stunde(n) seit der ersten Veröffentlichung eines Datensatzes für die Initiierung des Republishing verantwortlich ist. Maymounkov und Mazières beschreiben aber in [MaM02] das Republishing wie folgt:

[... W]hen a node receives a `STORE` *RPC [Remote Procedure Call, hier: Request] for a given key-value pair, it assumes the RPC was also issued to the other* $k-1$ *closest nodes, and thus the recipient will not republish the key-value pair within the next hour. This ensures that as long as republication intervals are not exactly synchronized, only one node will republish a given key-value pair every hour.*

Jeder Knoten hat also einen internen Zeitgeber, der jeweils den Ablauf einer Stunde seit Empfang des `STORE`-Requests für einen Datensatz meldet und das Republishing anstößt. Die Asynchronität der internen Zeitgeber wird insbesondere durch Transfer on Join verursacht. In diesem Fall erhält nur ein einziger Knoten, nämlich der neu hinzugekommene, in diesem Zeitpunkt ein Replikat, so dass seine interne Zeitrechnung bis zum nächsten

Republishing sich von der aller anderen Replikatbesitzer unterscheidet. In der Realität bedeutet dies, dass jeder der k Knoten aus $\mathcal{N}_b$ zu *irgend*einem Zeitpunkt innerhalb der Stunde ein Republishing durchführen müsste.
Es ist zu erwarten, dass die Republishing-Zeitpunkte dann näher beieinander liegen als in der vorliegenden Analyse, in keinem Fall aber können sie weiter als 60 Zeitschritte voneinander entfernt sein: solange es noch mindestens ein Replikat gibt, wird mindestens einmal pro Stunde republiziert.
Die Analyse legt also auch hier eine *untere Grenze* für die Entwicklung der Wahrscheinlichkeitsverteilung über $\tilde{k}$ und damit einhergehend für die Verfügbarkeit von Replika im Zeitverlauf fest.

4.3.5 Fazit

Die durchgeführte Analyse zeigt, wie sowohl moderat als auch für stark fluktuierende Netze mit hoher Angriffswahrscheinlichkeit über den Replikationsfaktor und die Wahl des Republishing-Intervalls die Verfügbarkeit der Datensätze gesichert werden kann.

Mit hoher Wahrscheinlichkeit reicht in einem Netzwerk mit moderater Fluktuation ein Replikationsfaktor von 39 (für starke Fluktuation von 107 bzw. 47 bei Halbierung der Republishing-Intervalle), um die Verfügbarkeit der Daten über 30 Tage zu garantieren. Mit einem nur geringen Zusatzaufwand kann diese Spanne bei Bedarf auch weiter verlängert werden.

4.4 Optimale Anfragepolitik bei Existenz böswilliger Speicherknoten

Kademlia bietet zwei grundlegende Lookup-Methoden: Die Methode `FIND_NODE` dient nur dem Auffinden der k nächsten Knoten zu einem Bezeichner. Ein Knoten, welcher eine Lookup-Nachricht mit `FIND_NODE` erhält, liefert stets die k ihm bekannten nächsten Knoten zum gesuchten Bezeichner zurück.
Anders verhält es sich mit der Methode `FIND_VALUE`, die dem Auffinden eines Datensatzes dient. Empfängt ein Knoten eine Lookup-Nachricht mit `FIND_VALUE`, so prüft er zunächst, ob er einen Datensatz mit dem gesuchten Bezeichner selbst gespeichert hat. Ist dies der Fall, liefert er direkt den Datensatz zurück. Besitzt er den gesuchten Datensatz nicht, so antwortet er wie bei `FIND_NODE` mit einer Liste von k besser passenden Knoten.

Es wird angenommen, dass ein Teil der Knoten böswillig handelt. Neben den in 4.3 untersuchten Effekten von Angriffen auf die Verfügbarkeit von Datensätzen ist auch damit zu rechnen, dass Daten von Speicherknoten modifiziert und/oder falsche Antworten geliefert werden (Integritätsangriff, siehe auch Kapitel 7).
Unter Berücksichtigung dieser Annahmen ergeben sich nun zwei praktikable Alternativen zum Bezug eines Datensatzes mit Bezeichner b:

1. Über die Anwendung von `FIND_NODE` werden die k Knoten aus $\mathcal{N}_b$ bestimmt. Der Lookup-Initiator sendet dann an jeden davon ein `FIND_VALUE`. Alle Knoten, welche

einen oder mehrere Datensätze mit Bezeichner b besitzen, liefern diese(n) zurück.

2. Bei Verwendung der Methode `FIND_VALUE` wird der Lookup beendet, sobald die erste Antwort (in Form eines oder mehrerer Datensätze) von einem kontaktierten Knoten gesendet wird. Erfüllt diese Antwort nicht die Integritätsanforderungen, so muss eine weitere Anfrage gestartet werden.
   Diese wird dann wie in 1. durchgeführt, indem alle k Knoten aus $\mathcal{N}_b$ befragt werden. Letzteres ist notwendig, da die Wahrscheinlichkeit sehr hoch ist, dass bei einer erneuten `FIND_VALUE`-Anfrage wieder derselbe Knoten wie zuvor kontaktiert wird, der inkorrekte Antworten liefert.

Es ist unmittelbar einsichtig, dass sich die beiden Varianten hinsichtlich ihres Kommunikationsaufwandes unterscheiden. Dabei seien folgende Annahmen getroffen:

- Eine Lookup-Nachricht hat die Länge l_{LOOKUP} Byte.
- Der Transfer eines Datensatzes entspricht l_{DS} Byte.
- In einem vollständig belegten Adressraum der Größe $N = 2^{160}$ (Adresslänge 160 Bit in Kademlia) ist die maximale Anzahl der in einem Lookup kontaktierten Knoten $log_2\ N = 160$. In einem Netzwerk, in dem nicht alle Adressen belegt sind ($N < 2^m$), ist die Anzahl x aller kontaktierten Knoten:

  $$x = O(log_2\ N) = a \cdot log_2\ N + b < m$$

  Untersuchungen von Ratnasamy et al. [GGG03] an einem Netzwerk der Größe 2^{16} ergaben die Pfadlänge $l_{path} \approx \frac{1}{2}\ log_2\ N$ pro Anfrage in einem Netzwerk ohne Knotenausfälle, wobei der Ausfall von über 50% der Knoten die durchschnittliche Pfadlänge um ca. 50% (also $l_{path} \approx \frac{3}{4}\ log_2\ N$) erhöht. Diese Analyse bezieht aber den Parallelisierungsfaktor α für Anfragen nicht ein. Unter Berücksichtigung von α ergäbe sich dann als Maximalwert für die Anzahl der pro Lookup kontaktierten Knoten $x = \alpha \cdot l_{path} \approx \alpha \cdot \frac{3}{4}\ log_2\ N$. Es ist also wahrscheinlich, dass x für $\alpha = 3$ und eine maximal angenommene Netzgröße $N = 2^{21}$ (über 2 Mio. Knoten) ≤ 48 ist.
- $\hat{k}$ ist die Gesamtzahl der Datensätze, die unter dem gesuchten Bezeichner b auf den Knoten aus $\mathcal{N}_b$ verfügbar sind. Dies beinhaltet *alle* Replika *aller* Datensätze mit identischem Bezeichner b (zur Existenz von gleichen Bezeichnern vgl. 5.3.2.1).
- $\bar{k}$ ist die Anzahl der verschiedenen unter identischem Bezeichner b publizierten Datensätze. Dabei gilt dann für den Anfragezeitpunkt: $\hat{k} = \bar{k} \cdot \tilde{k}$, d.h. die Gesamtzahl der verfügbaren Datensätze unter Bezeichner b auf allen Knoten $\in \mathcal{N}_b$ entspricht in etwa der Anzahl der unter diesem Bezeichner publizierten Datensätze multipliziert mit der erwarteten Anzahl der aktuell vorhandenen Replika pro Datensatz.
  Unterschiedliche Publikationszeitpunkte der Datensätze werden vernachlässigt, d.h. es wird vereinfachend angenommen, dass alle Datensätze in Form einer festen Anzahl $\tilde{k}$ an Replika vorliegen.
- Caching wird nicht berücksichtigt, d.h. der erste Knoten, der auf ein `FIND_VALUE` einen Datensatz zurückliefern kann, muss bereits $\in \mathcal{N}_b$ sein.

Der Kommunikationsaufwand in Byte (bezeichnet mit c) für die beiden verfügbaren Varianten kann damit wie folgt berechnet werden:

Variante I. Aufwand für `FIND_NODE` mit `FIND_VALUE` an jeden Knoten der Menge $\mathcal{N}_b$:

$$\begin{aligned} c_{\text{Var.I}} &= (1+k)\cdot x \cdot l_{LOOKUP} + \hat{k}\cdot l_{DS} && |\hat{k} = \bar{k}\cdot\tilde{k} \\ \Leftrightarrow c_{\text{Var.I}} &= \underbrace{(1+k)\cdot x \cdot l_{LOOKUP}}_{I} + \underbrace{\bar{k}\cdot\tilde{k}\cdot l_{DS}}_{II} \end{aligned}$$

Summand I repräsentiert dabei den Aufwand für das Auffinden der Knoten aus $\mathcal{N}_b$. Der Multiplikator $(1+k)$ ergibt sich daraus, dass pro kontaktiertem Knoten der Aufwand für eine Lookup-Nachricht vom Anfrageknoten hin zum kontaktierten Knoten plus der Aufwand für k Lookup-Nachrichten (Liste der besser passenden Knoten) in die entgegengesetzte Richtung anfällt. Summand II quantifiziert den Aufwand für den Transfer von $\hat{k}$ Datensätzen an den Initiator der Anfrage.

Variante II. Aufwand für standardmäßiges `FIND_VALUE` mit Umschwenken auf Variante I bei Misserfolg:

$$\begin{aligned} c_{\text{Var.II}} &= (1-p_a)\cdot((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS}) \\ &\quad + p_a\cdot(((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS}) \\ &\quad + (1+k)\cdot x\cdot l_{LOOKUP} + \hat{k}\cdot l_{DS}) && |\hat{k} = \bar{k}\cdot\tilde{k} \\ \Leftrightarrow c_{\text{Var.II}} &= \overbrace{(1-p_a)\cdot((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS})}^{I} \\ &\quad II \quad \begin{cases} + p_a\cdot(((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS}) \\ +(1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot\tilde{k}\cdot l_{DS}) \end{cases} \end{aligned}$$

Der Summand I stellt den Idealfall dar: der erste Knoten, der den Datensatz besitzt, ist kein Angreifer (Wahrscheinlichkeit $1-p_a$) und liefert die bei ihm gespeicherten $\bar{k}$ korrekten Datensätze zurück. Summand II repräsentiert den Fall, dass auf die erste Anfrage falsche Daten zurückgeliefert werden, da der Knoten ein Angreifer ist und die bei ihm gespeicherten Daten modifiziert hat. Sobald dies bei der Verarbeitung der Antwort bemerkt wird, muss eine zweite Anfrage gestellt werden. In diesem zweiten Durchgang wird dann wie in Variante I. vorgegangen, der Kommunikationsaufwand summiert sich.

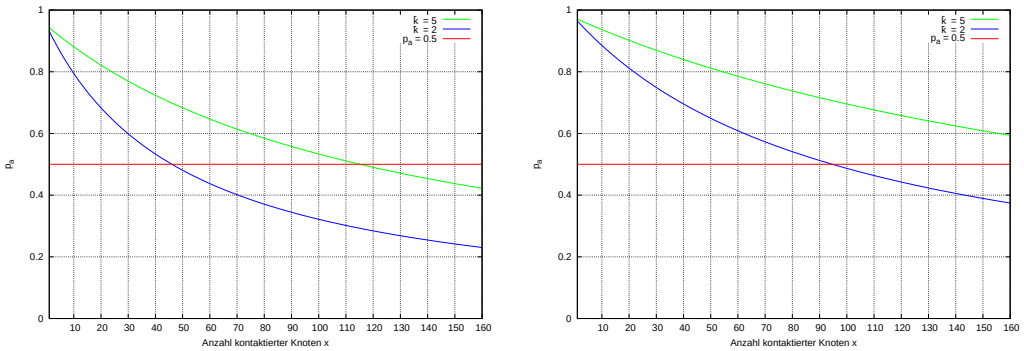
Durch Gleichsetzen $c_{\text{Var.I}} = c_{\text{Var.II}}$ und Lösen nach p_a wird die Angriffswahrscheinlichkeit berechnet, bei welcher der Kommunikationsaufwand für beide Varianten identisch ist.

$$\begin{aligned} (1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot\tilde{k}\cdot l_{DS} = \quad & (1-p_a)\cdot((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS}) \\ & + p_a\cdot(((1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot l_{DS}) \\ & + (1+k)\cdot x\cdot l_{LOOKUP} + \bar{k}\cdot\tilde{k}\cdot l_{DS}) \\ \Leftrightarrow & \\ p_a = \quad & \frac{(\tilde{k}-1)\cdot\bar{k}\cdot l_{DS}}{(1+k)\cdot x\cdot l_{LOOKUP} + \tilde{k}\cdot\bar{k}\cdot l_{DS}} \end{aligned}$$

Die Platzhalter l_{LOOKUP} und l_{DS} werden nun mit plausiblen Werten ersetzt. Da ein Lookup neben dem gesuchten 20-Byte-Bezeichner noch einige weitere Informationen (z.B. die verwendete Methode: `FIND_NODE` oder `FIND_VALUE`) enthält, wird hierfür von 40 Byte Gesamtgröße ausgegangen[10].
Die Größe eines Datensatzes hängt vom konkreten Anwendungsfall ab. Dementsprechend wird hier die Größe eines AAI-Zertifikats[11] verwendet (ca. 2 kB, also 2048 Byte [NyB99]). Unter weiterer Einsetzung des in 4.3 ermittelten optimalen Replikationsfaktors $k = 39$ für moderat fluktuierende Netzwerke ergibt sich dann:

$$p_a = \frac{2048 \cdot (\tilde{k} - 1) \cdot \bar{k}}{1600 \cdot x + 2048 \cdot \tilde{k} \cdot \bar{k}} \tag{4.19}$$

Der Wert für p_a, bei dem der Aufwand für Variante I und II identisch ist, hängt also von drei Parametern ab: der tatsächlichen Anzahl der kontaktierten Knoten x in einem Lookup (d.h. des exakten Lookup-Aufwandes), den zum gegebenen Zeitpunkt im Durchschnitt vorhandenen Replika pro Zertifikat $\tilde{k}$ und der Anzahl verschiedener Zertifikate, die unter gleichem Bezeichner gespeichert ($\bar{k}$) sind. Der Wert für x bewegt sich wegen $x = O(log_2\ N)$ mit $a \leq 1$ im Intervall $[0, 160]$, wobei $x \leq 48$ als wahrscheinlich gilt (siehe S. 102). Für $\tilde{k}$ gilt: $\tilde{k} \in [1, k]$.



(a) $\tilde{k} = 20; \bar{k} = 2,5$ (b) $\tilde{k} = 39; \bar{k} = 2,5$

Abbildung 4.15: Break-Even des Kommunikationsaufwandes zwischen Anfragevariante I und II (stetig interpoliert)

Abbildung 4.15(a) zeigt den Verlauf der Schnittkurve gemäß Formel 4.19 zwischen $c_{\text{Var.I}}$ und $c_{\text{Var.II}}$ aufgelöst nach p_a für eine beispielhafte Belegung der Parameter $\tilde{k} = 20 \approx \frac{1}{2}k$ (Average Case) bzw. $\tilde{k} = 39 = k$ (Worst Case hinsichtlich des Kommunikationsaufwandes) und $\bar{k} = 2$ bzw. $\bar{k} = 5$. Für Punkte (x_i, p_{a_i}) unterhalb einer Kurve ist Variante II vorteilhafter, für Punkte oberhalb Variante I.
Die angenommene Angriffswahrscheinlichkeit $p_a = 0.5$ ergibt für $k = 20$ und $\bar{k} = 2$ $x \approx 47$, d.h. die Kommunikationsaufwendungen für Variante I und II sind bei 47 kontak-

[10] Bei allen Angaben wird der auf den unteren Schichten des Netzwerks entstehende Overhead für UDP-Header usw. ausgeklammert.
[11] Zertifikate werden im verteilten AAI-Verzeichnis die größten Datensätze darstellen, s. Abschnitt 5.2.

tierten Knoten identisch. Wie bereits erwähnt, sind größere Werte für x in einem realen Kademlia-System unwahrscheinlich, so dass Variante II Szenarien vorzuziehen ist[12].

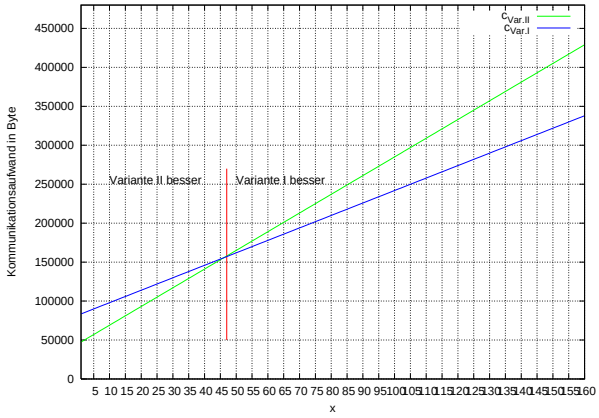
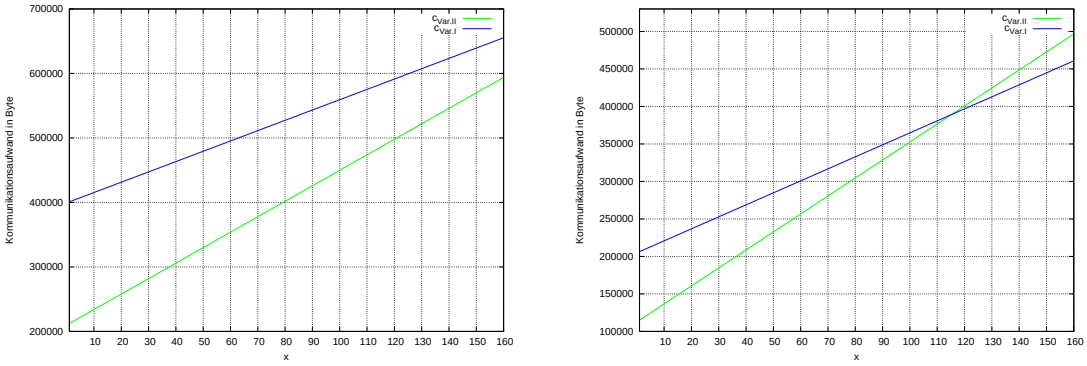


Abbildung 4.16: Kommunikationsaufwand für Variante I und II, stetig interpoliert; $\tilde{k} = 20, \bar{k} = 2$

Diese Entscheidung wird für größere $\bar{k}$ und $\tilde{k}$ untermauert (siehe Abb. 4.15(b) und obere Kurve in 4.15(a)): hier ist Variante I für $p_a = 0.5$ erst für sehr große x bzw. in keinem Fall (obere Kurve in Abb. 4.15(b)) vorteilhafter. Dies illustrieren auch Abb. 4.16 sowie Abb. 4.17(a) und 4.17(b) Zur Minimierung des Kommunikationsaufwandes ist also die



(a) $\tilde{k} = 39$ (b) $\tilde{k} = 20$

Abbildung 4.17: Kommunikationsaufwand für Variante I und II, stetig interpoliert; $\bar{k} = 5$

optimale Anfragepolitik unter den gegebenen Rahmenbedingungen stets Variante II.

[12]Für höhere Replikationsfaktoren k wird wegen der zugleich erwarteten höheren Werte für $\tilde{k}$ die Entscheidung bestätigt. So ergibt sich beispielsweise für $k = 100, \tilde{k} = 50, \bar{k} = 2, p_a = 0.5$ Aufwandsgleichheit bei $x \approx 49$.

Kapitel 5

System zur Speicherung von Zertifikaten und Statusinformationen : *P2P-ZuSI*

Ein zentraler Punkt dieser Arbeit ist die Spezifikation eines verteilten Informationssystems, welches die Daten einer AAI dezentral im Rahmen eines strukturierten P2P-Verzeichnisses speichert, verwaltet und für Anfragen bereithält. Hierfür muss ein Regelwerk entwickelt werden, welches von allen Teilnehmern des Systems verwendet wird und die Details der Formatierung, Speicherung und Anfrage von Datensätzen festlegt.
Die Kombination aus diesem Regelwerk (im Folgenden *P2P-ZuSI*-Regelwerk genannt) und einem konkreten Kademlia-basierten Peer-to-Peer-System wird als *P2P-ZuSI* (P2P Zertifikat- und Status-Informationssystem) bezeichnet und realisiert das verteilte Verzeichnis einer AAI.
Zunächst wird ein Grobentwurf von *P2P-ZuSI* vorgestellt. Dann wird die geeignete Status- bzw. Rückruftechnik für die durch den Entwurf vorgegebenen Rahmenbedingungen bestimmt und darauf das Regelwerk für Adressierung und Speicherung der Statusnachrichten aufgebaut.
Im Anschluss werden die Regeln für die Verwaltung von Zertifikaten und dabei insbesondere die Behandlung der Zertifikatinhalte hinsichtlich des Sicherheitsziels Vertraulichkeit diskutiert und formuliert.
Weiterhin wird ein Schichtenmodell vorgestellt, welches die Ergebnisse illustriert und als Basis für die formale Modellierung des spezifizierten Systems (siehe Kapitel 6) dient.

5.1 Grobkonzept

P2P-ZuSI ist ein Informationssystem, welches unter Verwendung eines festen Regelwerks ein Kademlia-basiertes P2P-Netzwerk nutzt, um als Verzeichnis zur Speicherung, Verwaltung und Bereitstellung von Daten einer AAI zu agieren.
Daten aus dem Verzeichnis können von jedem Teilnehmer des P2P-Systems angefragt werden. Weiterhin ist es jedem Knoten erlaubt, beliebige Daten zu veröffentlichen. Es ist somit Voraussetzung, dass jede Entität der AAI zugleich auch Teilnehmer des Netzwerks ist oder zumindest unbeschränkten Zugriff auf einen Netzwerkknoten hat. Damit

ist die Differenzierung zwischen Entität im Sinne der AAI und Knoten im Sinne der P2P-Infrastruktur im wesentlichen aufgehoben.

Es ist zu beachten, dass keine bestimmte AAI-Architektur besonders unterstützt werden soll, sondern eine möglichst allgemeine Eignung für verschiedene Architekturen zu erreichen ist. Somit bestehen insbesondere keine Beschränkungen dahingehend, welche AAI-Rollen von einem P2P-Systemteilnehmer eingenommen werden dürfen.

5.2 Verwaltung von Informationen zum Zertifikatstatus

5.2.1 Auswahl der optimalen Technik zur Statusverwaltung

Die in Abschnitt 3.9.2 vorgestellten Techniken sind in unterschiedlichem Maße für *P2P-ZuSI* geeignet.

5.2.1.1 Online-Techniken

In einem P2P-Verzeichnis findet die Verteilung von Informationen stets unter zeitlicher Verzögerung und unter Beteiligung zahlreicher Knoten statt. Eine direkte Verbindung und damit ein Online-Datenaustausch zwischen zwei AAI-Entitäten ist nicht möglich und auch nicht gewollt. Im Gegenteil, die dezentrale Datenspeicherung soll ja den Organisations- und Kommunikationsaufwand für die Issuing und Status Authorities verringern. Eine Online-Technik zur Kommunikation des Zertifikatstatus ist also nicht einsetzbar.

5.2.1.2 Offline-Techniken

Das strukturierte Peer-to-Peer-Verzeichnis kann prinzipiell zur Verwaltung von Offline-Rückrufinformationen genutzt werden. Die Status Authorities fügen dann regelmäßig aktualisierte, signierte Rückruflisten in das Netzwerk ein. Diese Certificate Revocation Lists bzw. Trees sind entweder vollständig oder in mehrere Teile gespalten speicherbar.
Um ein Zertifikat verifizieren zu können, muss ein Prüfer nach den zuletzt veröffentlichten Rückrufdaten suchen, ein Exemplar dieser Daten von einem gerade für diesen Datensatz zuständigen Peer herunterladen und ermitteln, ob das betreffende Zertifikat in dieser Datenstruktur als zurückgerufen deklariert wurde.
Die Integrität der Datei kann leicht mittels Signaturverifikation überprüft werden, indem hierfür der authentische öffentliche Schlüssel des Ausstellers verwendet wird. Damit wird Gewissheit darüber erlangt, ob Manipulationen durch P2P-Knoten durchgeführt wurden oder die Datei anderweitig beschädigt ist. Der Prüfer wird in einem solchen Fall die enthaltenen Informationen nicht mehr akzeptieren.

Allerdings ergäben sich beim Einsatz von Offline-Techniken im Rahmen von *P2P-ZuSI* wesentliche Nachteile:
Bei nicht fragmentierten Dateien wie vollständigen CRLs oder CRTs sind die einzelnen Dateien potentiell sehr groß, so dass hoher Kommunikations- und Speicheraufwand bei der Publikation entsteht. Somit wären aber Knoten mit geringen Ressourcen prinzipiell benachteiligt und möglicherweise sogar von der Teilnahme am System ausgeschlossen.

Die Replikation gestaltet sich ebenfalls schwierig, da mit den notwendigen Replikationsfaktoren sehr viel Speicherplatz benötigt wird und - insbesondere beim Republishing - hoher Kommunikationsaufwand entsteht. Lebenszyklusmanagement solcher Daten, insbesondere die Löschung z.B. bei Auftauchen einer neuen CRL ist u.a. aufgrund der möglicherweise differierenden Gültigkeitszeiträume und des notwendigen Aufwandes zur regelmäßigen Prüfung, ob Updates publiziert wurden, schwierig.
Das wesentlichste Problem liegt aber in der bereits in Abschnitt 3.9.2.2 beschriebenen Gefahr, dass ein Prüfer ein Zertifikat unzutreffenderweise als gültig ansieht, weil er die Rückrufliste oder Teile davon nicht auffinden oder aufgrund einer Beschädigung nicht verwenden kann. Da ein falsches Positivergebnis im Rahmen einer AAI als wesentlich schlimmer als ein falsches Negativergebnis einzustufen ist, ist die Verwendung von Offline-Techniken für *P2P-ZuSI* nicht zu empfehlen.

5.2.1.3 Rückruffreie Systeme

Trusted Directory
Ein Trusted Directory verlangt per Definition die Existenz einer vertrauenswürdigen, kontrollierten Umgebung, in der Zertifikate sicher und stetig verfügbar vorgehalten werden. In einem Peer-to-Peer-System wird der Dienst aber von einer Gruppe nicht notwendigerweise vertrauenswürdiger Teilnehmer angeboten. Eine weitere Voraussetzung ist die Existenz einer Instanz, welche für die Integrität der Daten organisatorisch und technisch verantwortlich ist. Beides kann nicht garantiert werden (siehe auch Kap. 8), so dass eine Kombination des Trusted-Directory-Ansatzes mit einem P2P-Verzeichnis unmöglich ist.

CRS/Novomodo und HCRS
Alle Statusnachrichten können einzeln im Netzwerk gespeichert werden. Diese kleinen, voneinander unabhängigen Datensätze halten den Kommunikationsaufwand allgemein gering und ermöglichen so auch die problemlose Einbindung von Knoten mit geringer Bandbreite und Speicherkapazität in das System. Auch entsteht kein Overhead, da ein Anfrageknoten immer nur genau die Information erhält, die ihn interessiert, und beispielsweise keine Rückrufinformationen über viele andere Zertifikate wie beim Einsatz von CRLs.
HCRS setzt allerdings eine gewisse „Intelligenz“ des Verzeichnisdienstes oder nicht zertifikatgebundene Adressierung der Datensätze voraus (siehe 3.9.2.3 und 5.2.2.2). Die Umsetzung dieser Anforderungen wäre schwierig, da Suchanfragen der Form „Liefere eine aktuelle SI für einen beliebigen Knoten aus $\mathcal{D}_i$“ nur mit wesentlichem Zusatzaufwand[1] realisierbar sind (zumal die Zusammensetzung von $\mathcal{D}_i$ den Prüfern nicht bekannt ist).
Die Statusnachrichten aus CRS sind allerdings sehr gut für die Speicherung in einem P2P-Netzwerk und damit für den Einsatz in *P2P-ZuSI* geeignet.

NewPKI
Da innerhalb des Peer-to-Peer-Netzwerks alle Knoten gleichberechtigt sind, ist es unerheblich, von welcher AAI-Entität eine Statusnachricht publiziert ist. Damit bietet NewPKI die gleichen Vorteile wie CRS und ist im skizzierten System sehr gut einsetzbar.

[1] z.B. mittels separat gespeicherter und über Metadaten adressierter Zuordnungsdatensätze, welche die konkreten FileIDs der Statusinformationen enthalten

Fazit: Aus der Menge der vorgestellten Techniken für die Bereitstellung von Rückrufen und Statusinformationen sind die rückruffreien Techniken CRS und NewPKI optimal für das skizzierte Peer-to-Peer-basierte Informationssystem.

5.2.2 Regelwerk für Statusinformationen

5.2.2.1 Paralleler Einsatz von IA- und EE-zentrierten Statusinformationen

Wie im vorigen Abschnitt gezeigt, sind CRS und NewPKI gleichermaßen für die Realisierung von Statusnachrichten im skizzierten *P2P-ZuSI* geeignet. CRS implementiert eine IA-zentrierte Sicht auf die Statusverwaltung, d.h. die IA kontrolliert den Status ihrer ausgestellten Zertifikate. NewPKI realisiert ein EE-zentriertes System mit der Verantwortung für die Statuskontrolle auf Seite der End-Entität. Es können also sowohl End-Entitäten als auch Issuing Authorities in der Rolle SA auftreten.
Da in einer AAI - wie in Abschnitt 3.9.1 und insbesondere Tabelle 3.1 angesprochen - sowohl IA-zentrierte als auch EE-zentrierte Gründe für den Rückruf von Zertifikaten vorliegen können, ist die alleinige Integration eines der beiden Konzepte in *P2P-ZuSI* nicht ratsam und auch nicht notwendig: Da alle Teilnehmer des P2P-Netzwerks beliebige Daten publizieren dürfen (vollständige Gleichberechtigung), werden in *P2P-ZuSI* Statusnachrichten aus allen möglichen Quellen gespeichert und verfügbar gemacht. Der Urheber einer Statusnachricht ist sogar Völlig unerheblich, sie muss nur gültig sein.
Negativnachrichten, d.h. explizite Rückrufe wie bei CRS/Novomodo in 3.9.2.3 vorgestellt, werden für *P2P-ZuSI* nicht verwendet. Es reicht aus, dass die Entität SA bei einem Rückrufereignis keine weiteren Positivnachrichten ausstellt. Zusätzliche Nachrichten zu publizieren erzeugt nur zusätzlichen Aufwand, bietet aber keinen Informationsvorteil und ist daher unnötig. Es kommen also nur Statusinformationen, d.h. positive Statusnachrichten zum Einsatz.

5.2.2.2 Publikation von Statusinformationen

Die Status Authority erzeugt in jeder Periode[2], in der das Zertifikat gilt, einen Datensatz der folgenden Form:

$$si_i(cert) = H^{d-i}(r)$$

Dabei ist i die Gültigkeitsperiode des Zertifikats, wobei mit Gültigkeitsdauer d (in Perioden) gilt: $1 \leq i \leq d$.
Anschließend bestimmt SA die FileID für den Datensatz aus dem Hashwert über das gesamte Zertifikat konkateniert mit der aktuellen Periode i:

$$fid_{si_i} = H(H(cert) \dot{+} i)$$

Da jeder Prüfer eine Statusinformation erst dann anfragt, wenn er das zugehörige Zertifikat bereits besitzt, kann er leicht dessen Hashwert bestimmen. Die Periode errechnet er aus Gültigkeitsbeginn des Zertifikats und dem aktuellen Datum (ggf. inkl. Uhrzeit).
Mit dieser Berechnungsmethodik für die FileID ist gewährleistet, dass jeder Prüfer diese leicht berechnen und damit auch jede Statusinformation zu einem bekannten Zertifikat

[2]Es ist generell keine Synchronisierung der Perioden, weder in ihrer Länge noch in ihrer Anzahl, notwendig. Es muss lediglich im Zertifikat angegeben werden, wie sich die Perioden errechnen.

anfragen kann.
Die Status Authority publiziert dann den Datensatz $(fid_{si_i}, si_i(cert))$ durch `STORE`-Requests an die k zuständigen Knoten.

5.2.2.3 Speichern von Statusinformationen

Erhält ein Knoten eine Statusinformation mit einem `STORE`-Request, speichert er diese und stellt sie auf Anfragen hin zur Verfügung. Nach Ablauf einer allgemein bekannten Mindestspeicherdauer löscht er sie, da die Statusinformation dann ungültig geworden ist.

5.2.2.4 Abfragen von Statusinformationen

Um eine aktuelle Statusinformation zu beziehen, muss ein Prüfer aus dem ihm vorliegenden Zertifikat *cert* die Werte $H(cert)$ und i berechnen und daraus die passende FileID erzeugen.
Er startet dann einen Lookup (gemäß den Ergebnissen aus Abschnitt 4.4 zunächst mit `FIND_VALUE`) nach einem Datensatz mit fid_{si_i}. Wird die Statusinformation von keinem Knoten zurückgeliefert, muss er annehmen, dass das Zertifikat nicht mehr gültig ist.
Erhält er eine Statusinformation, prüft er sie auf Korrektheit, indem er den i-fachen Hashwert über si_i berechnet und mit dem Validierungsziel aus dem Zertifikat vergleicht. Ist die Validierung erfolglos, fordert er den Datensatz nochmals von allen Peers aus $\mathcal{N}_{fid_{si_i}}$ an. Erst nach Abarbeitung der dabei empfangenen Ergebnisse entscheidet sich dann, ob das Zertifikat als gültig oder ungültig aufgefasst wird.

5.3 Verwaltung von Zertifikaten

Zertifikate sind im Hinblick auf ihre Verwaltung im P2P-Verzeichnis anders zu behandeln als Statusinformationen. Dies resultiert aus wesentlichen inhaltlichen und technischen Unterschieden wie in Tabelle 5.1 dargestellt.

Anmerkung: Die Länge der Statusinformationen wird in [Mic96] auf 160 Bit festgelegt. Es ist eine zufällige Koinzidenz, dass Bezeichner in Kademlia und Statusinformationen die gleiche Länge aufweisen.

	Statusinformationen	**Zertfikate**
Datensatzgröße	160 Bit = 20 Byte	im Mittel 1-2 kB
Inhalte	Hashwert einer Zufallszahl	signierte Klartextdaten
Art der Inhalte	alle vom gleichen Typ (Bitstring fester Länge)	Deskriptive Attributzertifikate (inkl. Schlüsselzertifikate); Privilegzertifikate

Tabelle 5.1: Unterschiede zwischen Statusinformationen und Zertifikaten

Zudem muss für Zertifikate eine andere Adressierungsmethodik gefunden werden, da sich die FileID für Statusinformationen unter Verwendung des Hashwerts über das Zertifikat ergibt, auf welches sich die Statusinformation bezieht.

Aus den gezeigten Differenzen gegenüber Statusinformationen resultieren Unterschiede in der Behandlung von Zertifikaten, die im Folgenden aufgegriffen werden.

5.3.1 Aufbau von Zertifikaten

In Anlehnung an das Zertifikatformat X.509 enthalten Privileg- und Deskriptive Attributzertifikate mindestens die folgenden Daten:

1. Inhaber-ID
2. Attribut- bzw. Privilegtyp
3. Attributwert bzw. Delegationsstufe
4. Gültigkeitszeitraum mit Start- und Enddatum
5. ggf. Erweiterungen
6. Aussteller-ID
7. Verwendeter Signaturalgorithmus
8. Signatur über den Hashwert des Zertifikatinhalts (1.-7.)

5.3.2 Regelwerk für Zertifikate

5.3.2.1 FileID-Berechnung

Ein Prüfer V, der ein Zertifikat sucht, besitzt zu diesem Zeitpunkt nur sehr wenige Informationen über Inhalte und Eigenschaften dieses Datensatzes.
Es wird angenommen, dass V über ein laufendes Authentifizierungs- und/oder Autorisierungsverfahren (A&A-Verfahren) in direktem Kontakt zu einer End-Entität E steht. Autorisierung bezieht sich - als Einschränkung der weiter gefassten Autorisierungsdefinition 3.2 - im Folgenden stets auf einen konkreten Privilegtyp, d.h. es wird geprüft, ob E ein von ihr beanspruchtes Privileg tatsächlich ausüben darf.
V benötigt dann aus dem Verzeichnis verschiedene Zertifikate, um eine Zertifizierungskette zu generieren. Die Typen der anzufragenden Zertifikate unterscheiden sich jeweils nach dem aktuellen Prozess-Schritt (Authentifizierung oder Autorisierung) und dem Typ des Initialzertifikats der Kette. Dieses kann entweder von E präsentiert werden oder aber wird von V auf Basis von durch E bereitgestellten Metadaten (z.B. Entitätsbezeichnung, beanspruchter Privilegtyp) aus dem Verzeichnis angefordert.

1. *Initialzertifikat ist Deskriptives Attributzertifikat mit Attributtyp pk:*
   Dieser Fall kann sowohl in der Authentifizierung (bei der Herleitung des authentischen öffentlichen Schlüssels für E) als auch in der Autorisierung (zur Herleitung des authentischen öffentlichen Schlüssels des Ausstellers eines Privilegzertifikats) auftreten. V wird dann eine Zertifizierungskette aus Schlüsselzertifikaten erstellen, d.h. alle Zertifikate sind vom Attributtyp pk.
   Im Falle der expliziten Trust-Modellierung ist außerdem parallel zu jeder Kette von Schlüsselzertifikaten eine Kette von Privilegzertifikaten des Typs $ci(pk)$ (siehe 3.7) zu erstellen.

2. *Initialzertifikat ist Privilegzertifikat mit Privilegtyp pt:*
   V fragt dann für die Erstellung einer Zertifizierungskette Privilegzertifikate vom

Typ *pt* mit höherer Delegationsstufe *pd* an. Weiterhin prüft er die Signaturen jedes erhaltenen Privilegzertifikats mittels des authentischen öffentlichen Schlüssels des Ausstellers. Um diesen Schlüssel zu ermitteln, muss jeweils eine Zertifizierungskette von Schlüsselzertifikaten generiert werden.

3. *Initialzertifikat ist ein Deskriptives Attributzertifikat mit Attributtyp at:*
   Die Signatur des Ausstellers wird über eine Kette von Schlüsselzertifikaten validiert (ggf. inkl. Privilegzertifikatkette $ci(pk)$). Zudem wird eine Kette von Privilegzertifikaten für das Privileg der Ausstellung von Zertifikaten des Typs a, $ci(at)$, generiert.

Damit hat V in den verschiedenen Fällen auch unterschiedliches Wissen über die Zertifikate, die er im Rahmen der Kettenerzeugung iterativ aus dem Verzeichnis anfordern muss. Es ergeben sich also nur zwei Typen von Zertifizierungsketten:

1. *Ketten von Schlüsselzertifikaten: V* sucht ausschließlich Deskriptive Attributzertifikate vom Typ *pk*. Der konkrete Attributwert, nämlich der Public Key selbst, ist ihm nicht a priori bekannt. Der Inhaber des jeweils nächsten zu suchenden Zertifikats ist der Aussteller des aktuell betrachteten Zertifikats (begonnen mit dem Initialzertifikat).

2. *Ketten von Privilegzertifikaten: V* sucht Zertifikate vom Typ *pt*, der direkt aus dem Initialzertifikat entnommen ist und sich über die gesamte Zertifizierungskette hinweg nicht ändert. Die Delegationsstufe muss für das jeweils nächste Zertifikat größer als die des aktuell betrachteten sein, die exakte Stufe ist allerdings nicht bekannt. Der Inhaber des nächsten anzufragenden Zertifikats ist wie auch bei Schlüsselzertifikatketten der Aussteller des aktuellen.

Weitere Informationen über die gesuchten Zertifikate liegen V also zu keinem Zeitpunkt vor. Damit kann die FileID nur aus diesen vorliegenden Informationen, nämlich dem Inhaber und dem Attribut- bzw. Privilegtyp des gesuchten Zertifikats erzeugt werden. Für die Berechnung der FileIDs von Zertifikaten ergibt sich daher:

$$\begin{aligned} fid_{cert_p} &= \psi(Inhaber, Privilegtyp) \\ fid_{cert_a} &= \psi(Inhaber, Attributtyp) \end{aligned} \tag{5.1}$$

Dabei ist ψ eine Einwegfunktion, die auf einen Bitstring fester Länge $m = 160$ abbildet:

$$\psi : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^m,$$

Im Folgenden wird ψ als konsistente Hashfunktion angenommen, die auf die Konkatenation von Inhaber und Privileg- bzw. Attributtyp angewandt wird:

$$\psi(Inhaber, Typ) = H(Inhaber \dot{+} Typ)$$

5.3.2.2 Behandlung der Zertifikatinhalte

Es ist nun festzulegen, in welcher Form die Datensätze selbst, also die Zertifikate, technisch vorgehalten werden sollen.

Angreifer. Bereits für die stochastische Verfügbarkeitsanalyse von Kademlia wurde angenommen, dass böswillige Systemteilnehmer negativen Einfluss auf die Datenbasis ausüben können. Ein Angreifer kann nun aber auch die Ermittlung möglichst vieler Informationen über die Datenbasis der gesamten AAI oder die für einen bestimmten Inhaber oder Aussteller publizierten Zertifikate anstreben (Angriff auf die Vertraulichkeit, siehe auch Kapitel 7).
Der Angreifer gewinnt dadurch nicht nur Informationen (z.B. über nicht-öffentliche Privilegien einer Entität), sondern kann diese auch als Basis für andere Angriffe verwenden, z.B. indem er nur Zertifikate eines bestimmten Inhabers von seinem System löscht.

Daher stellt sich die Frage, ob und wie die Vertraulichkeit von Zertifikatinhalten gesichert werden kann. Hierfür sind grundsätzlich zwei Herangehensweisen denkbar:

1. *Verschlüsselung bei Zertifikaterzeugung:* Bei der Erzeugung eines Zertifikats werden zunächst alle oder ausgewählte Felder verschlüsselt. Erst danach wird der Datensatz signiert und das fertige Zertifikat veröffentlicht.

2. *Verschlüsselung bei Publikation:* Das Zertifikat wird durch eine *P2P-ZuSI*-Methode im Zuge des Publikationsvorgangs vollständig verschlüsselt.

Verschlüsselung bei Zertifikaterzeugung. Hier werden die Zertifikatinhalte zunächst in unkritische und in zu schützende Felder aufgeteilt. Letztere werden mit einem symmetrischen Schlüssel chiffriert. Für die Schlüsselerzeugungsfunktion

$$\xi : \{0,1\}^l \rightarrow \{0,1\}^m$$

können nur die bereits in Abschnitt 5.3.2.1 vorgestellten Daten, nämlich Inhaber und Attribut- bzw. Privilegtyp verwendet werden, da sonst ein Prüfer die Daten nicht entschlüsseln kann. Als zu schützende Felder gelten insbesondere der Inhaber sowie Attributtyp und -wert bzw. Privilegtyp und Delegationsstufe. Erst nach der Chiffrierung erfolgt die digitale Unterschrift.
Ein Speicherknoten kann dann unkritische, aber für das Lebenszyklusmanagement interessante Daten wie das Gültigkeitsende des Zertifikats im Klartext lesen. Allerdings benötigt er zur Verifikation der Authentizität dieser Angaben den authentischen öffentlichen Schlüssel des Ausstellers. Die Prüfung der Signatur ist somit möglich, aber mit Aufwand für den Speicherknoten verbunden. Bei Weitergabe des Datensatzes beim Transfer on Join oder Republishing müsste jeder empfangende Knoten wiederum die Signatur des Ausstellers prüfen, damit der Datensatz nur dann gelöscht wird, wenn er tatsächlich nicht mehr gültig ist.
Es ist außerdem zu beachten, dass das entstehende Zertifikatformat aufgrund der Verschlüsselung nicht mehr gängigen Standards, z.B. X.509, entsprechen kann. Ein Wechsel der Algorithmen für Schlüsselerzeugung oder Ver- und Entschlüsselung würde zudem dazu führen, dass alle Zertifikate neu ausgestellt werden müssen.

Verschlüsselung bei Publikation. Für die Verschlüsselung bei der Publikation des Zertifikats kommt ebenfalls nur ein symmetrisches Verschlüsselungsverfahren in Frage, da die Entitäten, welche den Datensatz später entschlüsseln sollen, dem Einsteller nicht

bekannt sind. Er generiert also zunächst einen Schlüssel unter Verwendung einer Erzeugungsfunktion ξ wie oben und chiffriert damit das gesamte Zertifikat. Damit ist keine Möglichkeit zum Lebenszyklusmanagement gegeben, da die speichernden Knoten keinerlei Informationen über die Gültigkeitszeiträume der bei ihnen gespeicherten Zertifikate erhalten.
Die Erzeugung eines *zusätzlichen* Klartextteils, welcher nur zur Information der speichernden Knoten die Gültigkeitszeiträume der Zertifikate enthält, wäre vom Standpunkt der Sicherheit aus nicht vertretbar, da für einen speichernden Knoten die inhaltliche Verkettung zwischen diesem Klartext und dem Chiffretext nicht möglich wäre. Daher könnte auch nicht validiert werden, ob der angegebene Verfallszeitpunkt tatsächlich dem im Zertifikat angegebenen entspricht.
Bei dieser Variante zur Vertraulichkeitssicherung können die Verschlüsselungsmechanismen im *P2P-ZuSI*-Regelwerk gekapselt werden und nehmen damit keinen Einfluss auf die Zertifikaterzeugung seitens der IA-Entitäten. Die Zertifikate bleiben auf AAI-Schicht in einem standardkonformen Format. Weiterhin bleibt so eine gewisse Flexibilität gewahrt, z.B. kann dasselbe Zertifikat unter Verwendung verschiedener Schlüsselerzeugungs- und Verschlüsselungsfunktionen chiffriert werden.

Der Vorteil der Integritätssicherung des Chiffretextes bei der ersten vorgestellten Verschlüsselungsvariante verliert einen Großteil seines Wertes, wenn man bedenkt, dass bei jedem Transfer eines Zertifikats der speichernde Knoten eine aufwendige Signaturprüfung durchführen muss. Wird hingegen Verschlüsselung bei Publikation angewandt, besitzt das Chiffrat selbst zwar keine Integritätssicherung, aber nach der Dechiffrierung liegt - bei Erfolg der Entschlüsselung - dem Prüfer ein signierter Datensatz vor, dessen Integrität dann validiert werden kann. Daher wird die Verschlüsselung bei Publikation als Methode für die Vertraulichkeitssicherung in *P2P-ZuSI* eingesetzt.

5.3.3 Sicherheit

Erhält ein Knoten eine Speicheraufforderung für ein Zertifikat, so kann er nicht prüfen, ob FileID und Inhalt zusammenpassen. Er muss sich also darauf verlassen, dass eine korrekte Zuordnung vom publizierenden Knoten erzeugt wurde. In Kapitel 7 wird später gezeigt, dass die Publikation von Zertifikaten mit falschen FileIDs zwar unnötig Speicherplatz belegt, aber zu keinen erfolgreichen Angriffen führt.

Die FileID stellt sich dem speichernden Knoten als reiner Bitstring fester Länge dar. Aufgrund der Einwegeigenschaften der zu ihrer Erzeugung verwendeten Hashfunktion kann er daraus unmittelbar keine Informationen über den zugehörigen Datensatz herleiten. Dasselbe gilt für den symmetrischen Schlüssel, mit dem ein Zertifikat chiffriert ist.
Ist ein Angreifer auf der Suche nach Zertifikaten eines bestimmten Inhabers, so kann er die bei ihm gespeicherten Zertifikate einem Brute-Force-Angriff unterziehen, indem er alle möglichen Kombinationen von diesem Inhaber mit einem Attribut- oder Privilegtyp als Parameter für die Schlüssel- und FileID-Erzeugungsfunktion verwendet. Ein kluger Angreifer wird dabei mit dem häufigsten Typ (erwartungsgemäß pk) beginnen und in Richtung absteigender Häufigkeit vorgehen. Da Zertifikate für den gleichen Inhaber und Attributtyp - z.B. bei Schlüsselzertifikaten verschiedener Aussteller für einen Inhaber -

darüber hinaus die gleiche FileID und den gleichen Schlüssel besitzen, kann ein Angreifer dann direkt alle diese Zertifikate dechiffrieren.
Ein Angreifer, der in seiner Datenbasis Zertifikate anhand anderer Kriterien (z.B. Aussteller) suchen oder generell möglichst umfassende Information über seine Datenbasis erlangen will, muss im für ihn aufwendigsten Fall sogar *alle* im gesamten System möglichen Kombinationen von Inhaber und Attribut- bzw. Privilegtyp ausprobieren, um die Kombinationen, die für die bei ihm vorliegenden Datensätze verwendet wurden, zu ermitteln.
Generell kann keine vollständige Sicherheit gegenüber böswilligen Knoten gewährleistet werden: Da jeder Knoten zugleich auch Entität der AAI sein kann und daher das Regelwerk zur Schlüssel- und FileID-Erzeugung kennt, kann er immer versuchen, die Kombination von Inhaber und Typ zu erraten - gelingt dies, kann er auch die chiffrierten Inhalte entschlüsseln.
Je nach Größe des Netzwerks kann der Angriff mittels Brute Force jedoch aufwendig sein. In einem realistischen Szenario ist zudem nicht davon auszugehen, dass ein Angreifer über vollständige Kenntnis aller Zertifikatinhaber und Attributtypen verfügt, wodurch diese Form des Angriffs erschwert wird.

Die Vertraulichkeitssicherung mit Hilfe der Verschlüsselung bei Publikation ist nur als schwacher Schutz einzustufen. Die Schwäche basiert auf der Notwendigkeit, alle Zertifikate für AAI-Prüfer anfrag- und entschlüsselbar zu machen, während jeder P2P-Speicherknoten zugleich auch als AAI-Prüfer auftreten kann. Für diese Problematik existiert also keine ideale Lösung.
Dies hat zwei Konsequenzen für die Sicherheitseigenschaften von *P2P-ZuSI*:

1. Vertrauliche Zertifikatinhalte können nicht vor dem Zugriff von im Sinne einer konkreten AAI unbefugten Entitäten (also Speicherknoten) geschützt werden. Damit sollte bei Zertifikaten mit sensiblen Inhalten (z.B. Attributzertifikate für biometrische Muster, Privilegzertifikate für Berechtigung zur Verfügung über gewisse Geldbeträge eines Firmenkontos) in Erwägung gezogen werden, diese außerhalb des Systems zu speichern.
   Bei Privilegzertifikaten ist kein Vertrauen in die einzelnen Zertifikataussteller nötig. Es ist nur von Belang, dass eine valide Zertifizierungskette existiert. Da diese also nicht vom Vertrauen des Prüfers abhängt, ist es denkbar, für sensible Inhalte diese Zertifizierungskette offline bei der End-Entität zu speichern und bei der Autorisierung bereitzustellen.
   Ebenso könnte bei Deskriptiven Attributzertifikaten verfahren werden, indem das betreffende Zertifikat (Typ at) und die Zertifizierungskette von Privilegzertifikaten, welche die Ausstellungsberechtigung für den Inhaber belegt (Privilegtyp $ci(at)$), außerhalb des Verzeichnisses gespeichert wird. Dies sollte allerdings nur als Ausweichmöglichkeit verstanden und im Regelfall vermieden werden.
   Auch Pseudonymisierung von Privilegien ist in gewissen Szenarien (insbesondere bei hierarchischer oder hybrider AAI-Architektur) denkbar, um den Brute-Force-Angriff zu erschweren und die Bedeutung der Privilegien gegenüber Entitäten, die nicht zu einer definierten Gruppe autorisierter Prüfer gehören, zu verschleiern.
   Im Kapitel 8 wird auf die sinnvollen Einsatzszenarien für *P2P-ZuSI* ausführlich

eingegangen. Eine AAI, in welcher in signifikantem Umfang vertrauliche Privilegien und Attribute vorkommen, ist jedenfalls nicht der ideale Anwendungsfall für ein verteiltes Verzeichnis, sondern es ist die Datenspeicherung in einer vertrauenswürdigen, von den Authorities der AAI kontrollierten Umgebung vorzuziehen. Je nach Vertraulichkeitsbedarf ist weiterhin eine *nicht* zertifikatbasierte AAI mit zentralen Dienstanbietern hier möglicherweise die bessere Wahl, sofern die vertraulichen Inhalte auch den Prüfern nicht ungefiltert zur Verfügung stehen sollen.

2. Das Brechen der Vertraulichkeit von Inhalten darf nicht zu erfolgreichen Angriffen auf weitere Sicherheitsziele führen. Insbesondere darf die Integrität von Datensätzen nicht dadurch gefährdet werden, dass ein böswilliger Speicherknoten Zugriff auf den Klartext der Zertifikate erlangt. Die erfolgreiche Modifikation von Zertifikaten darf also nicht möglich sein. In Kapitel 7 wird dieser Umstand anhand eines formalen Modells des Systems verifiziert.
   Die in 4.3 quantifizierten Analyseergebnisse stützen sich auf das Angreifermodell eines *ungezielt* löschenden Angreifers, in welchem der hier betrachtete *gezielt* löschende Angreifer vollständig enthalten ist. Die Verfügbarkeitswahrscheinlichkeit kann sich somit gegenüber den Analyseergebnissen nicht verschlechtern.

Es ist aber stets zu bedenken, dass für einen erfolgreichen Angriff auf einen bestimmten Datensatz ein Knoten nicht nur die richtige Kombination von Inhaber und Typ kennen, sondern den Datensatz auch selbst gespeichert haben muss. Dazu muss er einer der k aktuell zuständigen Knoten sein oder zu einem früheren Zeitpunkt zu $\mathcal{N}_b$ gehört haben. Ist ersteres gegeben und löscht der Angreifer den Datensatz, ist gemäß der Analyse aus Abschnitt 4.3 dessen Verfügbarkeit dennoch mit sehr hoher Wahrscheinlichkeit weiterhin gegeben.

5.3.4 Erweiterung der Speichermethodik zur Verbesserung des Load Balancing

Gemäß der FileID-Berechnung mittels Formel 5.1 sind die FileIDs für alle Zertifikate, welche den gleichen Inhaber und Attribut- bzw. Privilegtyp besitzen, identisch. Damit sind für diese Daten aber auch die gleichen Knoten zuständig, was Probleme verursachen kann, wenn für eine Kombination von Inhaber und Typ sehr viele Zertifikate existieren, z.B. in einem Web of Trust mit multiplen Schlüsselzertifikaten für einen einzigen Teilnehmer.
Für solche Anwendungsfälle kann die Adressierung und Speicherung der Zertifikate im Netzwerk wie folgt angepasst werden:

1. *Speicherung:* Der Einsteller erzeugt eine FileID nach einem spezifischen Verfahren, z.B. als $fid_{cert} = H(Inhaber \dot{+} Typ \dot{+} Zufallszahl)$. Dann veröffentlicht er den Datensatz mit dieser FileID.

2. *Zuordnung:* Um den Datensatz auffindbar zu machen, erzeugt der Einsteller einen Datensatz mit FileID $fid_{mapping} = H(Inhaber \dot{+} Typ)$ wie in Formel 5.1 und mit fid_{cert} als Inhalt. Er sendet dann eine Speicheraufforderung an die Knoten aus $\mathcal{N}_{fid_{mapping}}$.

Ein Prüfer sucht dann nach $fid_{mapping}$ und verwendet die dabei zurückgelieferten tatsächlichen FileIDs, um die Datensätze zu beziehen.

Es ist unmittelbar einsichtig, dass die Knoten, welche die Zuordnungsdatensätze speichern, die ausschließliche Kontrolle über die Verfügbarkeit der Zertifikate für eine bestimmte Inhaber-Typ-Kombination haben. Daher verbessert diese Erweiterung die Robustheit des Systems gegenüber Angriffen auf die Verfügbarkeit von Datensätzen nicht. Die Vorteile dieser optionalen Erweiterung beziehen sich ausschließlich auf die gleichmäßigere Lastverteilung auf mehrere Knoten hinsichtlich des belegten Speicherplatzes.
Allerdings verlangt die Erweiterung eine Anpassung des Replikationsfaktors. Die Verfügbarkeitswahrscheinlichkeit eines Datensatzes setzt sich nun nämlich zusammen aus den Wahrscheinlichkeiten, dass der Zuordnungsdatensatz *und* das Zertifikat selbst verfügbar sind.
Dementsprechend ergibt sich mit aktueller Replikazahl $\tilde{k}_m$ des entsprechenden Zuordnungsdatensatzes und aktueller Replikazahl $\tilde{k}_c$ für die Wahrscheinlichkeit, dass mehr als ein Replikat eines so gespeicherten Datensatzes bezogen werden kann:

$$p(\tilde{k} \geq 1) = p(\tilde{k}_m \geq 1) \cdot p(\tilde{k}_c \geq 1)$$

Für angestrebtes $p(\tilde{k} \geq 1) \geq 0.99$ gilt also unter der Voraussetzung, dass sich die Verfügbarkeitswahrscheinlichkeit der Zertifikate und der Verwaltungsdatensätze gleich entwickelt[3], also $p(\tilde{k}_m \geq 1) = p(\tilde{k}_c \geq 1)$:

$$\begin{aligned} & p(\tilde{k} \geq 1) && \geq\ 0.99 \\ \Leftrightarrow\ & p(\tilde{k}_m \geq 1) && =\ \sqrt{0.99} \approx 0.995 \end{aligned}$$

Die Verfügbarkeitswahrscheinlichkeit jedes Zertifikats und jedes Zuordnungsdatensatzes muss also höher sein als in einem System ohne die vorgestellte Erweiterung. Dies ist durch entsprechende Erhöhung des Replikationsfaktors (z.B. von 39 auf 42 in einem moderat fluktuierenden Netzwerk) realisierbar.

5.3.5 Fazit

Das vorgestellte System zur Speicherung und Adressierung von Zertifikaten ermöglicht das Auffinden aller Zertifikate durch einen Prüfer. Zertifikate mit identischer Kombination von Inhaber und Typ werden dabei unter derselben FileID, also auch auf denselben Knoten abgelegt. Load Balancing ist durch die Implementierung der in 5.3.4 präsentierten Methodik realisierbar.
Das System bietet zusätzlich einen Grundschutz insbesondere gegen ungezielte Angriffe, bei welchen ein Knoten seine gespeicherten Daten analysieren will. Gegen gezielte Angriffe, bei denen Zertifikate bestimmter Inhaber dechiffriert werden sollen, wird nur ein geringer Schutz geboten. Dabei ist aber stets zu berücksichtigen, dass ein böswilliger Systemteilnehmer nur Datensätze angreifen kann, auf die er Zugriff hat, d.h. die er speichert.

[3] aufgrund gleichen Publikationszeitpunktes, gleichen Replikationsfaktors usw.

5.4 Realisierbarkeit von Reputationsmanagement

Der Einsatz von Reputationsmanagement in einem P2P-Netzwerk ermöglicht die Bewertung des Verhaltens einzelner Knoten. Jedes Knotenpaar, das eine Transaktion beliebiger Art ausführt, kann sich nach deren Ende gegenseitig bewerten. Die Reputation eines Knotens wird dann nach einem vorgegebenen Algorithmus aus den einzelnen Bewertungen gebildet. Ein Knoten, der eine Transaktion mit einem anderen durchführen will, kann dann auf Basis von dessen Reputationswert entscheiden, ob er dem Transaktionspartner vertraut oder lieber einen anderen wählt.

Es existiert eine Vielzahl von Reputationsmanagement-Ansätzen speziell für P2P-Umgebungen, beispielsweise EigenTrust [KSG03], FuzzyTrust [SHZ05] oder der Ansatz von Anceaume [AnR06], welcher auf Anreizsysteme für regelkonformes Verhalten basiert. Allerdings sind weiterhin viele Fragen und Schwächen ungeklärt, wie beispielsweise die Frage, mit welchem Initialwert für die Reputation bei einem neu hinzugekommenen Knoten gestartet werden soll: beginnt man mit einem neutralen Wert, kann es für einen böswilligen Knoten mit schlechten Bewertungen sinnvoll sein, offline und mit neuer NodeID wieder online zu gehen. Beginnt man hingegen mit einem negativen Wert, haben es neue Knoten schwer, überhaupt an Transaktionen teilnehmen zu dürfen und somit ihre Reputation zu steigern. Durch wechselnde IP-Adressen von Knoten und damit nichtpermanente NodeIDs wird diese Problematik noch verschärft.

Implementierte man Reputationsmanagement in *P2P-ZuSI*, so könnten sich speichernde, publizierende und anfragende Knoten nach Ende jedes Publikations-, Republishing- oder Anfragevorgangs gegenseitig bewerten. Allerdings würden sich hier wesentliche Schwächen zeigen. Ein Speicherknoten kann nicht ermitteln, ob ein empfangener Datensatz den Spezifikationen des Regelwerks genügt:

- *Statusinformation:* Da eine Statusinformation nur aus einem Hashwert besteht, kann nicht geprüft werden, ob dieser der passenden FileID zugeordnet ist.
- *Zertifikat:* Da Zertifikate vollständig verschlüsselt sind, kann auch hier keine Verbindung zwischen Inhalt des Datensatzes und FileID hergestellt werden.
  Wäre die Wahl auf Verschlüsselung bei Zertifikaterstellung gefallen, so könnte - mit entsprechendem Aufwand - diese Zuordnung geprüft werden. Damit würde sich die Bewertung von Transaktionen aber nur auf einen Teil der transferierten Daten beziehen, da die Zuordnung von Statusinformationen zu ihren FileIDs dennoch unprüfbar wäre. Dies wäre nicht sinnvoll, zumal Statusinformationen einen signifikanten Teil der gespeicherten Daten ausmachen.

Damit kann durch das Reputationsmanagement bei Publikations-Transaktionen nicht ermittelt werden, ob ein Knoten *böswillig* (im Sinne eines Angriffs auf das Gesamtsystem) handelt. Es kann lediglich bewertet werden, ob er sich hinsichtlich der Regeln des Peer-to-Peer-Netzwerks *konform* verhält, z.B. indem eine negative Bewertung ausgestellt wird, wenn ein Knoten keine Rückmeldung auf einen `STORE`-Request sendet.
Ein Speicherknoten, der im Rahmen von Republishing oder Transfer on Join ein Zertifikat erhält, kann diese Prüfung ebenfalls nicht durchführen. Es darf also kein Speicherknoten für eine falsche Antwort auf eine Anfrage verantwortlich gemacht werden, d.h. eine

schlechte Bewertung erhalten, da nicht nachvollzogen werden kann, welcher Knoten den Fehler verursacht hat. Neben den Speicherknoten könnte nämlich auch ein publizierender Knoten als Angreifer gehandelt haben, indem er mit Absicht falsche Datensätze veröffentlicht.

Reputationsmanagement kann also nur zur Bewertung von Knotenverhalten rein auf der Peer-to-Peer-Schicht eingesetzt werden, *nicht* aber, um Angriffe auf Inhalte zu entdecken und böswillige Teilnehmer auszuschließen.

5.5 Schichtenmodell

Nachdem nun das Regelwerk von *P2P-ZuSI* definiert ist, wird ein Schichtenmodell eingeführt, welches das *P2P-ZuSI*-Regelwerk als Zwischenschicht zwischen AAI- und P2P-Schicht platziert. Dieses Basismodell dient als Grundlage für das in Kapitel 6 detailliert ausgearbeitete formale Modell.

Die *P2P-ZuSI*-Regelwerk-Schicht bietet für die AAI-Schicht grundlegende Methoden an, welche von jeder AAI-Entität im Rahmen ihrer Aktionen aufgerufen werden können. Sie verwendet zu deren Ausführung Methoden der P2P-Schicht und realisiert somit die Vermittlung zwischen diesen beiden Schichten.
Die folgenden öffentlichen Methoden werden von der *P2P-ZuSI*-Regelwerk-Schicht bereitgestellt:

- `putCert(cert)` mit Rückgabetyp `boolean` (Publikation des Zertifikats `cert` erfolgreich oder nicht)
- `putSi(si)` mit Rückgabetyp `boolean` (Publikation der Statusinformation `si` erfolgreich oder nicht)
- `getCert(cert_info, request_method)` mit Rückgabetyp `cert_list` (Liste von Zertifikaten) oder `void` (bei Nichtexistenz eines passenden Datensatzes)
- `getSi(si_info, request_method)` mit Rückgabetyp `si_list` (Liste von Statusinformationen) bzw. `void`

Die Typen `cert_info` und `si_info` repräsentieren Informationen, welche die AAI-Entität über den gesuchten Datensatz mitliefern muss, wobei `cert_info = (holder,type)` sowie `si_info = (certhash,i)` gilt. Die Anfragemethode `request_method` in den `get`-Methoden kann basierend auf den Ergebnissen des Abschnitts 4.4 `getFirst` (Anfrage nur mit `FIND_VALUE`) oder `getAll` (Suche nach den Knoten aus $\mathcal{N}_b$ und dann Anfrage mit `FIND_VALUE` an jeden davon) sein.

Weiterhin besitzt die *P2P-ZuSI*-Regelwerk-Schicht interne Methoden, die nicht von außen zugreifbar sind:

- `createFid(cert_info)` und `createFid(cert)` mit dem Rückgabetyp `fid` (FileID) für Zertifikate

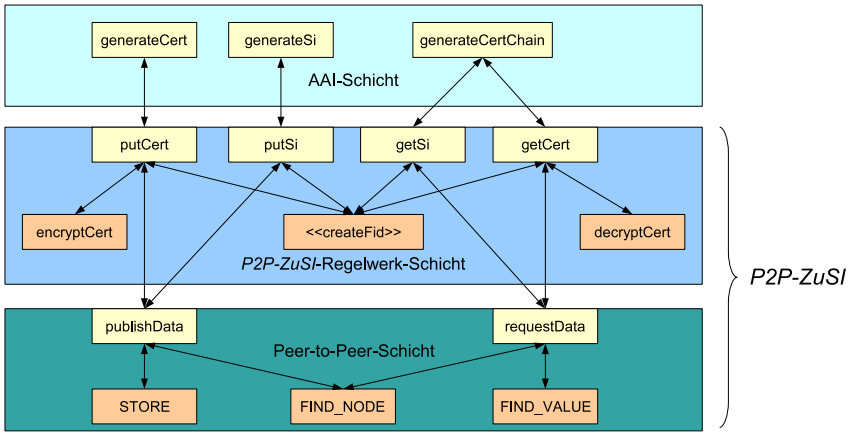


Abbildung 5.1: Schichtenmodell der AAI mit verteiltem Peer-to-Peer-Verzeichnis

- `createFid(si_info)` und `createFid(si)` mit dem Rückgabetyp `fid` für Statusinformationen

In einer `put`-Methode wird auf der *P2P-ZuSI*-Regelwerk-Schicht zunächst `createFid` mit dem Zertifikat bzw. der Statusinformation als Parameter aufgerufen. Dann wird der Datensatz `(fid,content)` an die Peer-to-Peer-Schicht und damit an den lokal betriebenen P2P-Knoten übergeben. Dieser veröffentlicht den Datensatz gemäß Kademlia-Spezifikation (vgl. 2.5.3).

Bei einer `get`-Methode wird ebenfalls zunächst `createFid` aufgerufen, um aus den gelieferten Informationen der AAI-Schicht die FileID des gesuchten Datensatzes zu bestimmen. Das Ergebnis wird als Parameter an die `FIND_NODE`- bzw. `FIND_VALUE`-Methode der Peer-to-Peer-Schicht übergeben.

Abbildung 5.1 visualisiert den Aufbau der Schichten und das Zusammenspiel der wesentlichen Methoden. Für die AAI-Schicht sind nur die Methoden aufgeführt, welche eine Interaktion mit dem Verzeichnis einschließen. Interne Methoden dieser Schicht wie beispielsweise die zur Validierung von erzeugten Zertifizierungsketten oder zur Durchführung eines Authentifizierungsverfahrens sind ausgeblendet.
Die dargestellten Schichten sind auf den Schichten 5, 6 und 7 des ISO/OSI-Referenzmodells [Zim80] anzusiedeln: Das Peer-to-Peer-Regelwerk baut als Overlay der Schicht 5 (Sitzungsschicht) direkt auf UDP (Transportschicht, Layer 4). Das *P2P-ZuSI*-Regelwerk organisiert auf der Präsentationsschicht (Schicht 6) die Formatierung von Daten und verbirgt die Details der P2P-Schicht vor der Anwendung (AAI auf Schicht 7).

Kapitel 6

Modellierung

Im aktuellen Kapitel wird ein formales Modell vorgestellt, welches das Zusammenspiel der Entitäten auf AAI- und Peer-to-Peer-Schicht unter Verwendung von *P2P-ZuSI* abbildet. Dabei liegt der Entwicklung das Schichtenmodell aus 5.5 zugrunde. Es wird ein rollenbasierter Fokus gewählt, d.h. die Methoden des Schichtenmodells werden in explizit modellierten AAI- und P2P-Rollen implementiert, wobei auch das rollenbasierte AAI-Modell aus 3.8 einbezogen wird.

6.1 Modellierungsmethode

6.1.1 Einführung in gefärbte Petrinetze

Gefärbte Petrinetze (Coloured Petri Nets, CPN) gehören zur Klasse der High-Level-Petrinetze. Diese basieren auf dem klassischen Petrinetzmodell [Pet62], implementieren aber spezifischen Erweiterungen.

Petrinetze allgemein eignen sich besonders zur formalen Modellierung von Informations- und Kontrollflüssen in nebenläufigen oder verteilten Systemen mittels einer klaren mathematischen Syntax. Die Modellierungsmethode ist dabei aber graphisch orientiert, d.h. komplexe Vorgänge aus der Realität werden mit einfachen graphischen Mitteln abgebildet.

Jedes Petrinetz ist aus vier Grundbausteinen aufgebaut:

- *Marken (Tokens)*: Eine Marke ist eine abstrakte Repräsentation von im Netz übertragenen und verarbeiteten Daten.
- *Stellen (Places)*: Eine Stelle ist Speicherort für eine oder mehrere Marken.
- *Transitionen (Transitions)*: Eine Transition ist die abstrakte Repräsentation für einen Daten verarbeitenden Prozess. Eine Transition ist aktiviert, wenn jede ihrer Eingabestellen (Stellen, von denen aus ein Bogen zur Transition führt) die durch die Gewichtung vorgegebene Markenanzahl beinhaltet. Die Verarbeitung einer aktivierten Transition wird Schalten oder Feuern genannt.
- *Bögen (Arcs)*: Ein Bogen ist ein gerichtete Verbindung von einer Stelle zu einer

Transition oder umgekehrt. Die Gewichtung eines Bogens gibt an, wie viele Marken in einem Übergang zwischen den beiden Endpunkten transferiert werden.

Der Zustand eines Petrinetzes ist die aktuelle Belegung aller Stellen mit Marken. Der Anfangszustand, auch als Initial Marking oder Anfangsmarkierung bezeichnet, ist der a priori vorgegebene Zustand eines Netzes, in welchem noch keine Transitionen ausgelöst wurden.
Ein Zustandsübergang ist die Umformung eines bestehenden Zustandes in einen neuen Zustand durch Schalten genau einer Transition.

Definition 6.1 ***Petrinetz*** *(nach [Pet62], [Jen92])*
Ein Petrinetz ist ein gerichteter, bipartiter Graph, bestehend aus zwei verschiedenen Arten von Knoten (Stellen und Transitionen) sowie gerichteten Kanten zwischen diesen. Es ist ein 5-Tupel

$$PN = (P, T, A, w, M_0),$$

welches die folgenden Anforderungen erfüllt:

1. *P ist die endliche Menge der Stellen.*
2. *T ist die endliche Menge der Transitionen.*
3. *A ist die endliche Menge der Bögen mit*

$$A \subseteq (P \times T) \cup (T \times P)$$

4. *w wird als Gewichtungsfunktion (Weight Function) bezeichnet und ist eine Abbildung*

$$w : A \to \mathbb{N}$$

5. *M_0 wird als Anfangsmarkierung (Initial Marking) bezeichnet und wird von einer Initialisierungsfunktion $I : P \to \mathbb{N}_0$ wie folgt erzeugt:*

$$\forall p \in P : M_0(p) = I(p).$$

6. *Es gilt $P \cap T = \emptyset$, $P \cup T \neq \emptyset$.*

In klassischen Petrinetzen können allerdings keine konkreten Daten gezielt modelliert, transportiert und verarbeitet werden. Da dies für die Formalisierung einer AAI mit verteiltem Verzeichnis aber unerlässlich ist, werden hierfür gefärbte Petrinetze eingesetzt. In diesen wird jeder Marke ein einfacher oder zusammengesetzter Datentyp zugewiesen, welcher als Colour Set oder Farbtyp bezeichnet wird. Die Farbe eines Tokens (Token Colour) ist dann die aktuelle Belegung der Felder des Datentyps mit konkreten Werten.

Als Äquivalent zur Gewichtung von Bögen in klassischen Petrinetzen werden in gefärbten Petrinetzen Multisets [Jen92] von Marken zwischen Stellen und Transitionen transferiert; Markierungen von Stellen werden ebenfalls in Form von Multisets dargestellt. Ein Multiset basiert dabei auf einer Menge S, wobei sie jedes Element von S mehrfach enthalten kann.

Definition 6.2 ***Multiset*** *(gemäß [Jen92])*
Ein Multiset m über eine nicht-leere Menge S ist eine Funktion

$$m \in [S \to \mathbb{N}].$$

$[S \to \mathbb{N}]$ ist die Menge aller Funktionen[1] *von S in $\mathbb{N}$. Die Anzahl des Vorkommens von Element s in Multiset m ist $m(s) \in \mathbb{N}$. In der Regel wird m als Summe geschrieben:*

$$m = \sum_{s \in S} m(s)`s,$$

dabei ist $m(s)`s$ als „$m(s)$-mal s“ zu lesen. $\{m(s)|s \in S\}$ sind die Koeffizienten des Multisets m, $m(s)$ ist dabei der Koeffizient von s. Ein Element $s \in S$ gehört genau dann zum Multiset m, wenn $m(s) \neq 0$, man schreibt folglich $s \in m$.

- *Addition und Subtraktion von Multisets m_1, m_2:*

$$m_1 \pm m_2 = \sum_{s \in S} (m_1(s)) \pm m_2(s))`s$$

  Subtraktion ist nur dann möglich, wenn $m_2 \leq m_1$.

- *Skalarmultiplikation, $n \in \mathbb{N}$, m Multiset: $n * m = \sum_{s \in S} (n * m(s))`s$*
- *Größe des Multisets: $|m| = \sum_{s \in S} m(s)$. Bei $|m| = \infty$ heißt m unendlich.*
- *Vergleichsoperatoren: $\leq, \geq, =$ und $\neq$*
  Gleichheit: Zwei Multisets sind gleich, wenn alle ihre Koeffizienten gleich sind.
  Ungleichheit: $\forall s \in S : m_1 \leq m_2$ und $\forall s \in S : m_1 \geq m_2$

Der Operator ++ definiert die Vereinigung (Union) von Multisets. Folglich ergibt der Ausdruck $2`token_1 + +3`token_2$ mit $token_1, token_2 \in S$ ein Multiset mit drei $token_2$- und zwei $token_1$-Elementen.
Abbildung 6.1 zeigt beispielhaft die Elemente eines gefärbten Petrinetzes. Kommentare sind in `(* *)` gefasst.

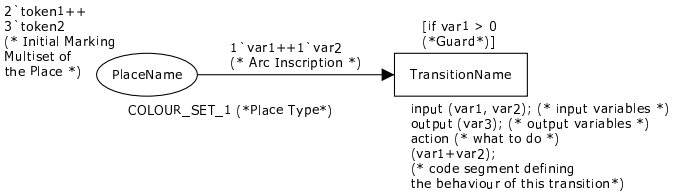


Abbildung 6.1: Elemente eines gefärbten Petrinetzes

Folgende Notation wird in der formalen Definition gefärbter Petrinetze verwendet [CHP93]:

- Die Menge von Multisets über eine Menge T wird als T_{MS} geschrieben.

[1] Die Notation $m \in [S \to \mathbb{N}]$ wird im aktuellen Kapitel gemäß [Jen92] äquivalent zur Notation $m : S \to \mathbb{N}$ verwendet.

- Der Typ einer Variable v wird als $Type(v)$ bezeichnet. Analog wird der Typ eines Ausdruckes $expr$ als $Type(expr)$ geschrieben.
- Die Menge aller in einem Ausdruck vorkommenden Variablen ist $Var(expr)$.
- Ein Ausdruck $expr$ mit $Var(expr) = 0$ heißt geschlossener Ausdruck.
- Der Datentyp Boolean wird als $\mathbb{B} = \{true, false\}$ geschrieben.

Definition 6.3 ***Gefärbtes Petrinetz (nach [Jen92])***
Ein gefärbtes Petrinetz nach ist ein Tupel

$$CPN = (\Sigma, P, T, A, N, C, G, E, I)$$

das die folgenden Anforderungen erfüllt:

1. Σ *ist eine endliche Menge von Colour Sets.*
2. P *ist eine endliche Menge von Stellen.*
3. T *ist eine endliche Menge von Transitionen mit* $P \cap T = \emptyset$.
4. A *ist eine endliche Menge von Bögen mit* $P \cap A = T \cap A = \emptyset$.
5. N *ist eine Knotenfunktion* $N : A \rightarrow (P \times T) \cup (T \times P)$.
6. C *ist eine Farbfunktion* $C : P \rightarrow \Sigma$.
7. G *ist eine Wächterfunktion. Sie ist definiert von* T *in alle Ausdrücke, so dass:*

$$\forall t \in T : [(Type(G(t)) = \mathbb{B}) \wedge (Type(Var(G(t))) \subseteq \Sigma)]$$

8. E *ist eine Funktion von Bogenausdrücken. Sie ist definiert von* A *in alle Ausdrücke, so dass mit der zu* $N(a)$ *gehörigen Stelle* $p(a)$*:*

$$\forall a \in A : [(Type(E(a)) = C(p(a))_{MS}) \wedge (Type(Var(E(a))) \subseteq \Sigma)]$$

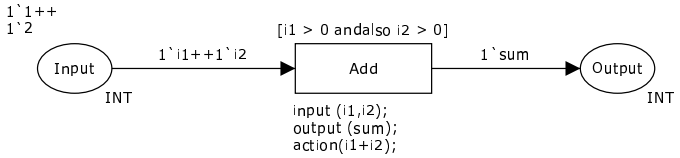
9. I *bezeichnet die Initialisierungsfunktion. Sie ist definiert von* P *in alle geschlossenen Ausdrücke, so dass:*

$$\forall p \in P : [Type(I(p)) = C(p)_{MS}]$$

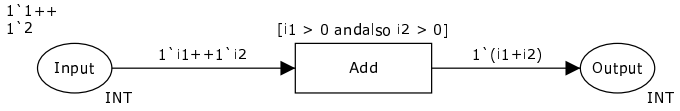
In den Transitionen werden die in den Marken enthaltenen Daten verarbeitet und neue Ausgabemarken generiert. Die Farbtypen der Ausgabemarken sind dabei unabhängig von den Typen der Eingabemarken. Jede Transition besitzt einen Bezeichner und eine Wächterfunktion (Guard Function, Guard), in welcher unter Verwendung aller Eingabemarken der Transition zusätzliche Bedingungen für ihr Schalten definiert werden.

Abbildung 6.2(a) zeigt ein sehr einfaches gefärbtes Petrinetz. Die Stelle *Input* kann ein Multiset von `INT`-Marken (Ganzzahlen) beinhalten und ist konkret mit zwei Marken belegt. Die Transition *Add* ist aktiviert, wenn die Eingabestelle mindestens zwei Token enthält (siehe Arc), deren Farben (`INT`-Werte) beide > 0 sind (siehe Guard). Die

Transition realisiert die Summenbildung aus `i1` und `i2`. Diese Summe wird als einzelnes Ausgabetoken in die Stelle *Output* übertragen.
Bemerkung: Die Beschreibung der Datenverarbeitung in Transitionen kann sowohl im Code-Segment als auch in Form von Bogenausdrücken auf den ausgehenden Arcs erfolgen. Das Petrinetz aus Abb. 6.2(a) ist also bezüglich der Funktionalität identisch mit dem aus Abb. 6.2(b).



(a) Mit Code-Segment



(b) Ohne Code-Segment

Abbildung 6.2: Beispiel: Addierer zweier positiver Zahlen

6.1.2 Hierarchische gefärbte Petrinetze

Die Modellierungsmethode der gefärbten Petrinetze wird zur Modellierung des vorliegenden Sachverhalts mit einer anderen Methode aus dem Bereich der High-Level-Petrinetze kombiniert: den hierarchischen Petrinetzen [Feh93].
Hierarchische gefärbte Petrinetze (HCPN) erlauben die Darstellung von Sachverhalten und Verarbeitungskomplexen auf unterschiedlichen Abstraktionsebenen (Pages). Eine Page kann über eine sogenannte Substitutionstransition, mit der eine Page niedrigerer Hierarchieebene verknüpft wird, komplexe Sachverhalte abstrahieren.
So kann beispielsweise ein komplexer Datenverarbeitungsprozess in Form einer Substitutionstransition *VerarbeiteDaten* abstrahiert werden, während die Details der Datenverarbeitung auf der zugehörigen Subpage, also der einer Substitutionstransition zugeordneten Page modelliert werden. Die Hierarchieebenen sind dabei schachtelbar, d.h. eine Subpage kann wiederum Substitutionstransitionen enthalten, die zu weiteren Subpages führen.
Zu jedem hierarchischen gefärbten Petrinetz existiert ein äquivalentes nicht-hierarchisches Petrinetz, welches sich durch Ersetzung aller Substitutionstransitionen durch ihre Subpages ergibt.

In der folgenden formalen Definition wird die Menge der Elemente X eines HCPN verwendet. Dabei gilt: $X = P \cup T$, d.h. X beinhaltet alle Stellen und Transitionen des Netzes.

$X \in [X \rightarrow X_S]$ bildet jedes Element $x \in X$ auf die Menge seiner Umgebungselemente ab, also an alle Elemente, die mit x über einen eingehenden oder ausgehenden Bogen

verbunden sind:

$$X(x) = \{x' \in X | \exists a \in A : [N(a) = (x, x') \vee N(a) = (x', x)]\}$$

$In \in [X \to X_S]$ bildet jedes Element $x \in X$ auf die Menge seiner Eingabeelemente ab, also die Knoten, die einen Bogen zu x besitzen:

$$In(x) = \{x' \in X | \exists a \in A : N(a) = (x', x)\}$$

$Out \in [X \to X_S]$ bildet jedes Element $x \in X$ auf die Menge seiner Ausgabeelemente ab, also die Knoten, zu denen von x aus ein Bogen hinführt:

$$Out(x) = \{x' \in X | \exists a \in A : N(a) = (x, x')\}$$

Weiterhin ist die Socket-Typ-Funktion $ST(x_1, x_2)$, $x_1, x_2 \in X$ eine Funktion, die Paare von Socket-Elementen und Substitutions-Elementen auf $\{in, out, i/o\}$ abbildet:

$$ST(x_1, x_2) = \begin{cases} in & wenn\ x_1 \in (In(x_2) \setminus Out(x_2)) \\ out & wenn\ x_1 \in (Out(x_2) \setminus In(x_2)) \\ i/o & wenn\ x_2 \in (In(x_2) \cap Out(x_2)) \end{cases}$$

Der Socket-Typ für eine Stelle p und eine Transition t ist also in, wenn p eine Eingabe-, aber keine Ausgabestelle von t ist. Analog out, wenn p nur Ausgabestelle, und i/o, wenn p Ein- und Ausgabestelle ist.

Definition 6.4 ***Hierarchisches gefärbtes Petrinetz*** *(gemäß [Jen92], [NeH04])*
Ein HCPN ist ein Tupel

$$HCPN = (S, SN, SA, PN, PT, PA, FS, FT, PP),$$

welches die folgenden Anforderungen erfüllt:

1. *S ist eine endliche Menge von Seiten (Pages), so dass:*

   - *Jede Seite $s \in S$ ist ein gefärbtes nicht-hierarchisches[2] Petrinetz :*

     $$(\Sigma_s, P_s, T_s, A_s, N_s, C_s, G_s, E_s, I_s)$$

     Für die Elemente des gesamten HCPN wird die folgende Notation verwendet:

     $$\Sigma = \bigcup_{s \in S} \Sigma_s,\ P = \bigcup_{s \in S} P_s,\ T = \bigcup_{s \in S} T_s,\ A = \bigcup_{s \in S} A_s.$$

     Dabei überlappen die Mengen der Colour Sets üblicherweise, während die Mengen $(P_{s_i}, T_{s_i}, A_{s_i}), (P_{s_j}, T_{s_j}, A_{s_j})$ für verschiedene Seiten s_i, s_j in der Regel disjunkt sind.

[2] Beachte hierzu die Anmerkung auf Seite 128.

- *Die Mengen von Netzelementen sind paarweise disjunkt:*

$$\forall s_1, s_2 \in S : [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1} \cup A_{s_1}) \cap (P_{s_2} \cup T_{s_2} \cup A_{s_2}) = \emptyset]$$

2. *$SN \subseteq T$ ist eine Menge von Substitutions-Elementen aus der Menge aller Transitionen.*

3. *SA ist eine Seiten-Zuordnungsfunktion. Sie ist definiert als $SA : SN \rightarrow S$, so dass keine Seite eine Subpage von sich selbst ist:*

$$\{s_0 s_1 ... s_n \in S^* | n \in \mathbb{N} \wedge s_0 = s_n \wedge \forall k \in 1..n : s_k \in SA(SN_{s_{k-1}})\} = \emptyset$$

4. *$PN \subseteq P$ ist eine Menge von Port-Elementen aus der Menge aller Stellen.*

5. *PT ist eine Port-Typfunktion. Sie ist als $PN \rightarrow \{in, out, i/o\}$ definiert.*

6. *PA ist eine Port-Zuordnungsfunktion. Sie ist definiert von SN in binäre Relationen (also in Verbindungen zwischen zwei Elementen des Netzes: $PA : SN \rightarrow P \times T \cup T \times P$). Dabei gilt:*

   - *Socket-Elemente sind mit Port-Elementen verbunden:*

$$\forall t \in SN : PA(t) \subseteq X(t) \times PN_{SA(t)}$$

   - *Socket-Elemente sind vom richtigen Typ:*

$$\forall t \in SN \forall (p_1, p_2) \in PA(t) : [PT(p_2) \in \{in, out, i/o\} \Rightarrow ST(p_1, t) = PT(p_2)]$$

   - *Verbundene Elemente haben identische Colour Sets, äquivalente[3] Initialisierungsausdrücke, Wächterfunktionen und Bogenausdrücke:*

$$(a) \forall t \in SN \cup T : \forall (p_1, p_2) \in PA(t) : [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$$

$$(b) \forall p \in SN \cup P : \forall (t_1, t_2) \in PA(p) : [G(t_1) = G(t_2) \wedge \begin{cases} A(t_1, p) = A(t_2, p), falls\ PT(t_2) = in, \\ A(p, t_1) = A(p, t_2), falls\ PT(t_2) = out, \\ PT(t_2) = in \wedge PT(t_2) = out, falls\ PT(t_2) = i/o \end{cases}]$$

   $(c) \forall (p, t) \in SN$ *und* $\forall (t, p) \in SN :$ *für t muss (a), für p (b) erfüllt sein.*

7. *$FS \subseteq P_s$ ist eine endliche Menge von Fusion Sets, so dass Elemente eines Fusion Sets identische Colour Sets und äquivalente Initialisierungsausdrücke besitzen:*

$$\forall fs \in FS : \forall p_1, p_2 \in fs : [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$$

8. *FT ist eine Fusions-Typ-Funktion. Sie ist definiert $FT : FS \rightarrow \{global, page, instance\}$, so dass Seiten- und Instanz-Fusion-Sets zur gleichen Page gehören:*

$$\forall fs \in FS : [FT(fs) \neq global \Rightarrow \exists s \in S : fs \subseteq P_s]$$

[3] Äquivalent bedeutet hier, dass die Ausdrücke nicht identisch sein, aber in der Auswertung für das leere Bindungselement den gleichen Wert ergeben müssen.

9. *$PP \in S_{MS}$ ist ein Multiset von Prime Pages.*

Um Verwechslungen zwischen den Begriffen zu vermeiden, folgt nun eine informelle Definition von Subpages und Superpages.

Definition 6.5 ***Subpages und Superpages***
Eine Superpage ist eine Seite eines HCPN, welche Substitutionstransitionen enthält, deren genaue Funktionsweise in Seiten der nächstniedrigeren Hierarchieebene (Subpages) beschrieben ist. Die Seiten der höchsten Hierarchieebene werden als Main Pages bezeichnet.
Eine Subpage ist also eine Seite, welche einer Superpage zugeordnet ist und die wiederum selbst Substitutionstransitionen und damit Verweise auf eine niedrigere Hierarchieebene beinhalten kann.

Einige Punkte der formalen Definition werden nun ausführlicher erläutert:

1. Jede Seite ist ein nicht-leeres, nicht-hierarchisches Petrinetz, wobei gemäß [Jen06]: „[The formal definition of CPN] certainly allows for *any* number of hierarchical levels and it also allows for ordinary transitions. It tells that there is a set of [non-hierarchical] pages - nothing about how they are related to each other. This is specified by the page assignment function [...]."
   Laut Jensen ist also die Verschachtelung mehrerer Hierarchieebenen ohne Widerspruch zu seiner Definition möglich. Dies wird auch im Computerprogramm *CPN Tools* [CPNTO], welches in seiner Arbeitsgruppe an der Universität Aarhus entwickelt wurde, unterstützt.

2. Jedes Substitutions-Element ist eine Transition. Im Folgenden wird ausschließlich die Bezeichnung *Substitutionstransition* verwendet.

3. Die Seiten-Zuordnungsfunktion stellt eine Verbindung zwischen einer Substitutionstransition und der ihr zugehörigen Subpage her.

4. Jedes Port-Element ist eine Stelle. Im Folgenden wird daher die Bezeichnung *Port-Stelle* oder vereinfacht *Port* verwendet.

5. Der Port-Typ teilt die Port-Stellen in *in*-, *out*- und *i/o*-Port-Stellen.

6. Die Port-Zuordnung verbindet Socket-Stellen mit Port-Stellen. Unter Socket-Stellen versteht man dabei die Eingabe- und Ausgabestellen der Substitutionstransition auf der Superpage. Jeder Socket-Stelle auf der Superpage muss eine Port-Stelle auf der Subpage entsprechen. Den Eingabestellen auf Superpage-Ebene entsprechen auf Subpage-Ebene die *in*-Ports, Ausgabestellen die *out*-Ports, bidirektional verbundenen Stellen die *i/o*-Ports.
   Ein Port hat mit seinem Socket den Farbtyp und die (ausgewertete) Anfangsbelegung gemeinsam. Insbesondere gilt hier, dass die Anfangsbelegung für den Port nicht explizit geschrieben werden muss, da er sie automatisch vom Socket übernimmt.

7. Fusion Sets sind Mengen von Stellen mit gleichem Farbtyp. Alles, was mit einer einzelnen Stelle des Fusion Sets geschieht (Hinzufügen bzw. Entfernen von Token),

geschieht auch mit allen anderen Stellen des Fusion Sets. Das Verhalten eines Fusion Sets ist also analog dazu, eine einzige Stelle zu verwenden, welche mit Bögen zu allen Transitionen, die ansonsten eine Verbindung zu einem Element des Fusion Sets hätten, verbunden ist.

8. Der Fusions-Typ unterteilt die Menge der Fusion Sets in *global*, *page* und *instance* Fusion Sets.

9. Die Prime Pages sind ein Multiset über die Menge aller Seiten. Sie legen fest, wie viele Instanzen jeder einzelnen Page vorliegen. Es ist also möglich, dass eine Page in Form mehrerer konkreter Instanzen vorkommt.

Im Rahmen dieser Arbeit werden weder Fusion Sets noch Prime Pages eingesetzt.

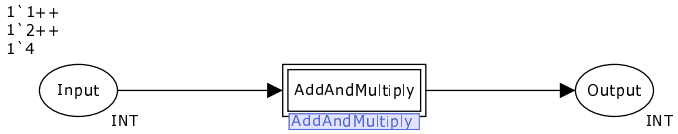


Abbildung 6.3: Beispiel: Main Page

Abbildung 6.3 zeigt die einzige Main Page für ein Beispielnetz, welches `INT`-Werte verarbeitet. Die doppelt umrandete Transition ist dabei eine Substitutionstransition mit der Subpage *AddAndMultiply* wie in Abb. 6.4 dargestellt.

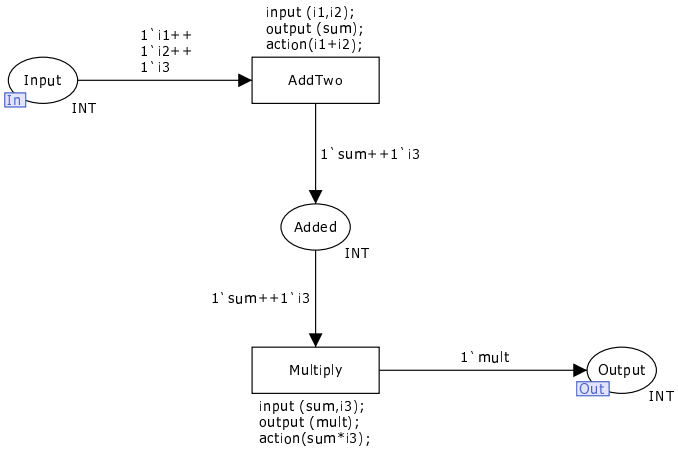


Abbildung 6.4: Beispiel: Subpage für die Substitutionstransition *AddAndMultiply*

Die Initialbelegung der Stellen wird hier auf die oberste Abstraktionsebene beschränkt. Die Marken stehen dann auch auf der Subpage zur Verfügung. Den Ein- und Ausgabestellen der Substitutionstransition auf oberer Ebene werden äquivalente Stellen (Ports) auf der Subpage zugeordnet. Diese sind dort mit *in*, *out* oder *i/o* markiert. Im vorliegenden Beispiel ist die Stelle *Input* ein *in*-Port, *Output* ein *out*-Port. Die Verarbeitung der Daten erfolgt in beliebig vielen Transitionen auf der Subpage.

6.1.3 Zustandsräume

6.1.3.1 Einführung

Zum Verständnis der folgenden Definition des Zustandsraums (State Space) eines gefärbten Petrinetzes (unabhängig von der Implementierung hierarchischer Strukturen) ist die Definition des Bindungselements notwendig:

Definition 6.6 ***Bindung und Bindungselement***
Die Bindung einer Transition t ist die Belegung aller Variablen von t mit Werten. Ein Bindungselement (Binding Element) ist ein Paar (t, b) bestehend aus einer Transition $t \in T$ und einer Bindung b für t. Es wird als aktiv bezeichnet, wenn die Wächterfunktion von t zu true ausgewertet wird und die Eingabestellen von t genug Token enthalten, um die Transition zu feuern.

Die Bindung b ist also die Belegung der Variablen auf den eingehenden Bögen von t mit konkreten Marken aus den Eingabestellen.

Der Zustandsraum eines gefärbten Petrinetzes ist ein nicht notwendigerweise endlicher gerichteter Graph, der alle erreichbaren Zustände des Netzes beinhaltet (vollständige Enumeration). Jeder Knoten des Graphs repräsentiert einen Zustand, also eine eindeutige Belegung aller Stellen im Netz mit Marken.
Jeder gerichteten Kante zwischen zwei Knoten entspricht die Verarbeitung eines speziellen Bindungselements, also das Feuern einer Transition mit einer spezifischen Variablenbelegung, wodurch das Netz vom Ausgangszustand der Kante in den Zielzustand überführt wird.
Der erste Knoten des Zustandsraums wird auch als Initialknoten bezeichnet. Er repräsentiert die Anfangsmarkierung (Initial Marking) des Netzes. Jeder Knoten ist ausgehend vom Initialknoten über endlich viele Kanten erreichbar.
Die Knoten, die von einem betrachteten Knoten aus unmittelbar über eine Kante erreichbar sind, werden auch als Nachfolgeknoten (Successor Nodes) bezeichnet.

Abbildung 6.5 zeigt ein gefärbtes Petrinetzmodell mit nicht endlichem Zustandsraum.

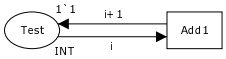


Abbildung 6.5: Beispiel für ein CPN mit unendlichem State Space

Definition 6.7 ***Zustandsraum*** *(nach [Val98])*
Der Zustandsraum eines Petrinetzes ist ein Tupel

$$(S, T, \Delta, S_I)$$

mit

- *S ist eine Menge von (globalen System-)Zuständen.*

- *T ist eine Menge struktureller Transitionen, d.h. Entitäten, deren Ausführung eine Zustandsänderung bewirkt.*
- *$\Delta \subseteq S \times T \times S$ ist eine Menge von semantischen Transitionen (Kanten). Dabei ist jede semantische Transition die Durchführung (Occurrence) einer strukturellen Transition, wobei $\delta \in \Delta = (s, t, s')$ mit Startzustand $s \in S$, durchgeführter Transition $t \in T$ und Zielzustand $s' \in S$.*
- *S_I ist eine Menge von Initialzuständen. Dabei gilt $S_I \subseteq S$, $S_I \neq \emptyset$*

Ein State Space ist endlich, wenn S und T endlich sind.

6.1.3.2 Zustandsraumexplosion und Gegenmaßnahmen

Ein häufig auftretendes Problem bei der Zustandsraumberechnung für ein gefärbtes Petrinetz ist die Zustandsraumexplosion (State Space Explosion). Dieser Begriff bezeichnet den Umstand, dass der Zustandsraum mit Zunahme der Anzahl an Stellen, Transitionen und Marken zum exponentiellen Wachstum tendiert.

In Abbildung 6.6 ist ein simples Beispielnetz dargestellt. Es werden einzeln Ganzzahlen aus der Stelle *Input* entnommen und auf jede dieser Zahlen 1 addiert. Daraufhin wird jeder Wert erst mit 2, dann mit 3 multipliziert. Dieses Beispielnetz hat mit gezeigtem

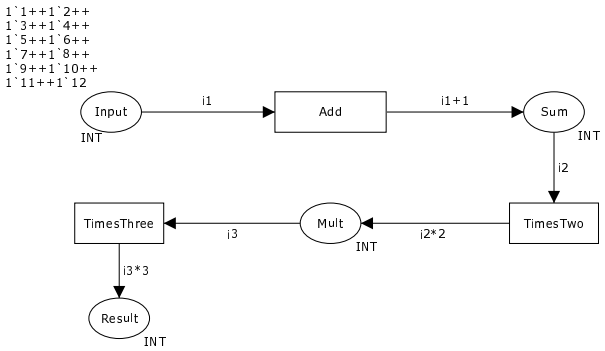


Abbildung 6.6: Beispielnetz für Zustandsraumexplosion

Initial Marking mit 6 Marken in der *Input*-Stelle bereits einen State Space der Größe 4096, bei 12 Token von 14726.
Abbildung 6.7 zeigt einen kleinen Teil des Zustandsraums bei Anfangsbelegung von *Input* mit 6 Marken. Hier wird deutlich, dass dieser enorm große State Space in einem Netz mit nur sehr wenigen Stellen und Transitionen vor allem durch die Variationsmöglichkeiten der Reihenfolge von Transaktionsausführungen zustande kommt.

Ausgehend vom Initialzustand 1 können 6 States erreicht werden. Jeder steht dabei für die Auslösung der Transition *Add* mit einer der 6 möglichen Belegungen von Variable `i1`. Zustand 2 wird beispielsweise erreicht, indem *Add* mit `i1=6` ausgeführt wird. Ausgehend hiervon kann entweder gleich die Transition *TimesTwo* für das Ergebnis des ersten Schritts ausgeführt (Zustand 8) oder ein weiteres Mal *Add* mit einer der fünf weiteren

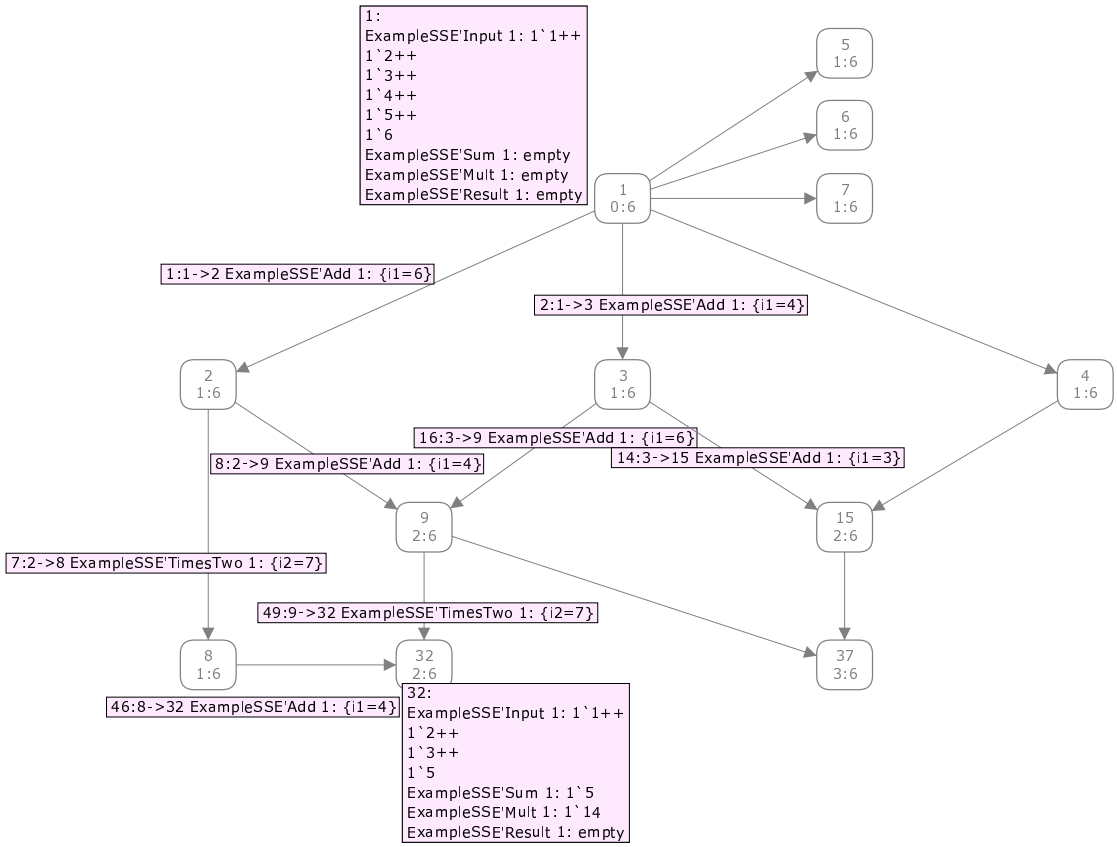


Abbildung 6.7: Auszug aus dem State Space des Beispielnetzes, Anfangsbelegung 6 Token

möglichen Belegungen für `i1` gefeuert werden. Ist nun beispielsweise `i1=4`, so gelangt man zu Zustand 9. Dieser hätte aber auch erreicht werden können, wenn zuerst `i1=4` und im zweiten Schritt `i1=6` die Belegung gewesen wäre (man kommt dann von State 3). In dem gezeigten Beispiel ist die Reihenfolge, in welcher die Transitionen durchlaufen werden, prinzipiell nicht von Belang, da jedes Token einzeln verarbeitet wird und das Gesamtergebnis der Berechnung, also der Endzustand des Zustandsraums, für alle möglichen Reihenfolgen der Ausführung identisch ist. In der letzten Transition laufen also alle vorherigen Zustände in einem Endzustand zusammen (siehe Abbildung 6.8), in welchem für jedes Token des Initial Marking das Ergebnis $(\texttt{i1}+1) \cdot 2 \cdot 3$ in *Result* gespeichert ist.

Hier bietet sich ein Anknüpfungspunkt zur Optimierung: Dort, wo Daten unabhängig voneinander auf gleiche Art und Weise verarbeitet werden, wird nun eine feste Ausführungsreihenfolge im Petrinetz festgelegt. Somit wird eine große Anzahl an Zuständen gespart, welche auf der unterschiedlichen Reihenfolge von Transitionsausführung basieren, aber am Ende keine unterschiedlichen Ergebnisse liefern. Hierfür werden Multisets in Listen von Elementen desselben Farbtyps umgewandelt und unterstützend Listen und Stellen vom Farbtyp `E` verwendet. Marken vom Typ `E` können nur eine Farbe, nämlich `e` annehmen. Sie tragen keine Information und sind somit das CPN-Äquivalent zu gewöhn-

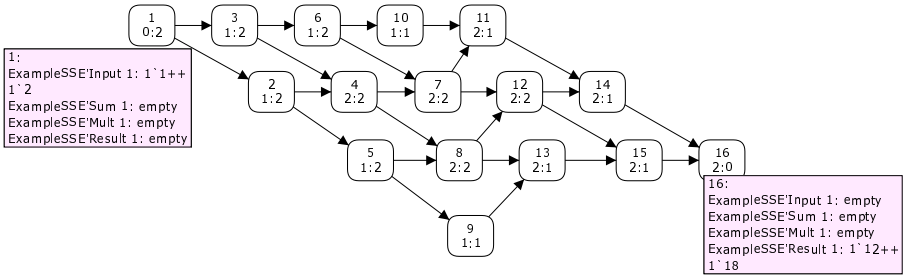


Abbildung 6.8: State Space zum Beispielnetz, Anfangsbelegung 2 Token

lichen Petrinetz-Token.
Im Beispiel heißt dies, dass die *Input*-Stelle nun nicht mehr mehrere `INT`-Token speichert, sondern immer genau eine Liste von `INT`-Werten (Format einer Liste in CPN ist `[a,b,c]`). Dafür muss selbstverständlich das Colour Set von *Input* entsprechend umgewandelt werden. Dann werden der Reihe nach alle Werte dieser Liste an die *Add*-Transition übergeben. Dabei wird immer das erste Element der Liste (`hd int_list`) verarbeitet und an die Liste, welche die Ergebnisse beinhaltet, angehangen. Der unverarbeitete Rest (`tl int_list`) wird zurück in *Input* übertragen.
Sind alle Elemente verarbeitet, greift die Bedingung `if List.length(tl int_list)=[]`, d.h. es existieren keine Listenelemente mehr, welche auf Verarbeitung durch *Add* warten. Erst dann wird ein `e`-Token in *AddFinished* übertragen; bevor dies nicht geschehen ist, kann die Transition *TimesTwo* nicht feuern. Ebenso werden nun zuerst alle Elemente durch *TimesTwo* verarbeitet und erst dann *TimesThree* aktiviert.

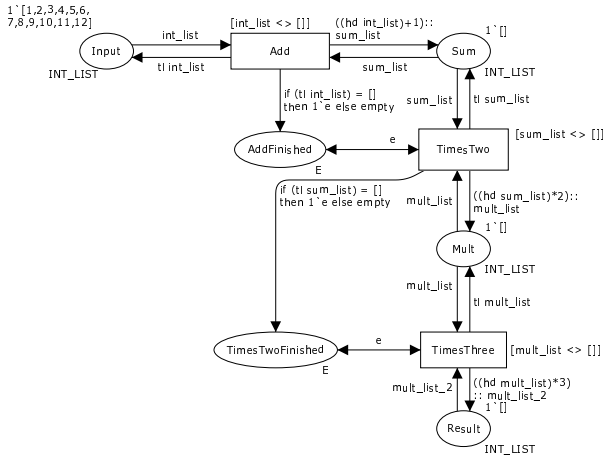


Abbildung 6.9: Beispielnetz ohne State Space Explosion

Durch dieses Vorgehen wird der Zustandsraum enorm verkleinert: wird auch hier ein Initial Marking mit 12 Ganzzahlen verwendet (diesmal in Listenform), so hat der State Space nur 37 Zustände - gegenüber den 14726 Zuständen im Original-Modell ein immenser Gewinn bei identischem Ergebnis.

Diese Listentechnik wurde bei der Erstellung des Modells einer AAI mit verteiltem Verzeichnis an allen Stellen, an denen dies möglich war, eingesetzt. Dadurch konnte die Zustandsraumexplosion vermieden und der State Space für die spätere Sicherheitsanalyse (siehe Kapitel 7) hinreichend klein gehalten werden. Weiterhin wurde zu diesem Ziel Determinismus in den Datenflüssen angestrebt, d.h. das Feuern einer beliebigen Transition aktiviert stets maximal eine weitere Transition.

6.2 Rollenbasiertes Modell einer AAI mit *P2P-ZuSI*

Das Modell wurde mit der Software *CPN Tools* [CPNTO] der CPN Group, Universität Aarhus, erstellt. Neben der reinen Erzeugung von (hierarchischen) gefärbten Petrinetzen ermöglicht diese Software auch Simulation und Zustandsraumanalyse. Für die Formulierung von Bogenausdrücken, Guards usw. wird die Programmiersprache CPN/ML [CPNML] verwendet, ein CPN-spezifischer Dialekt der funktionalen Programmiersprache SML [MTH97][4].

6.2.1 Top-Level-Modell

Die AAI mit *P2P-ZuSI* als Verzeichnisdienst wird angelehnt an das Schichtenmodell aus 5.5 in ein gefärbtes Petrinetz überführt. Folgende Entwurfskriterien liegen diesem Prozess zu Grunde:

1. *Ebenenbasierte Sicht:* Die drei Schichten AAI, Peer-to-Peer und *P2P-ZuSI*-Regelwerk werden gemeinsam auf einer einzigen Main Page abgebildet. So wird insbesondere die Übersichtlichkeit der Datenflüsse und des Initial Marking gewährleistet.

2. *Rollenbasierte Sicht:* Die handelnden Entitäten der AAI- und P2P-Schicht werden jeweils einfach in Form von Rollen dargestellt. Die AAI-Rollen nutzen dabei das *P2P-ZuSI*-Regelwerk, welches selbst nur unterschiedliche Methoden (implementiert als Transitionen) besitzt, aber keine Rollen.

3. *Initial Marking auf oberster Ebene:* Alle für das Initial Marking relevanten Stellen sind bereits auf der Main Page (Top-Level-Modell) enthalten. Dadurch wird sowohl die Anpassung des Initial Marking an verschiedene Szenarien als auch der Überblick über die aktuelle Belegung vereinfacht.

4. *Abstraktion der Methoden durch Hierarchie:* Die Interaktion zwischen Rollen gleicher und unterschiedlicher Ebenen wird in Form von Substitutionstransitionen realisiert. Diese werden im Detail in den Subpages ausgearbeitet.

5. *Beschränkung auf Schlüssel- und Privilegzertifikate:* Authentifizierung und Autorisierung werden in einfacher Form abgebildet, indem für die Authentifizierung eine Kette von Schlüsselzertifikaten, für die Autorisierung eine Kette von Privilegzertifikaten hergeleitet wird. Sonstige Deskriptive Attributzertifikate werden nicht betrachtet.

[4] Ausdrücke, Methoden usw. in CPN/ML- *oder* SML-Notation werden im Folgenden zusammenfassend als ML-Ausdrücke, -Methoden usw. bezeichnet.

6. *Implizites Vertrauensmodell:* Privilegzertifikate, welche die Berechtigung zur Ausstellung von Schlüsselzertifikaten beinhalten, werden aus Gründen der Modellkomplexität nicht eingesetzt.

Insbesondere die beiden letzten, die Modellkomplexität mindernden Entwurfskriterien sind bereits im Vorgriff auf die modellbasierte Sicherheitsanalyse (Kapitel 7) gewählt. Es genügt dort die Untersuchung der Sicherheit von Schlüsselzertifikaten, da diese Repräsentanten für alle Arten von Deskriptiven Attributzertifikaten darstellen. Existiert kein erfolgreicher Angriff auf Schlüsselzertifikate, so auch nicht auf andere Deskriptive Attributzertifikate. Würde hingegen ein Angriff auf Schlüsselzertifikate gefunden, wäre dieser auch auf andere Deskriptive Attributzertifikate übertragbar. Analog verhält es sich mit Empfehlungen: diese stellen eine bestimmte Form von Privilegzertifikaten dar. Bei expliziter Modellierung von Vertrauen müssten für jedes Schlüsselzertifikat auch Ketten von Privilegzertifikaten des Typs *ci(pk)* generiert werden.
Die Erzeugung einer solchen Kette wird genauso wie die im Zuge des *Autorisierungs*verfahrens durchgeführte Herleitung einer Kette von Privilegzertifikaten (siehe 6.2.4.4.4) realisiert. Findet sich also dort ein erfolgreicher Angriff, wäre der gleiche Angriff auch bei expliziter Vertrauensmodellierung bei der Herleitung einer Kette von Schlüsselzertifikaten möglich. Erweist sich die Herleitung von Privileg-Zertifizierungsketten bei der Autorisierung hingegen als sicher, gilt dies auch für Privileg-Zertifizierungsketten von Empfehlungen.
Die Möglichkeiten der später erfolgenden Sicherheitsanalyse werden also durch den Verzicht auf die Modellierung von explizitem Vertrauen und Deskriptiven Attributzertifikaten, die nicht vom Typ *pk* sind, *nicht* beschränkt.
Abbildung 6.10 zeigt das Top-Level-Modell der AAI mit *P2P-ZuSI*-Verzeichnis. Die Rollen der AAI sind blau umrahmt. In der Mitte befinden sich türkis umrandet die Methoden von *P2P-ZuSI*. Auf der rechten Seite sind in rot die Rollen im Peer-to-Peer-Netzwerk dargestellt.
Im Folgenden werden die drei wesentlichen Teile des Top-Level-Modells (Main Page) erläutert. Danach wird zu den Subpages übergegangen. Alle erwähnten ML-Methoden finden sich in Appendix B. Abbildung 6.11 zeigt zur besseren Übersicht die Substitutionstransitionen (und damit die direkten Subpages) des Top-Level-Modells.

Bemerkung: Die zwischen dem *P2P-ZuSI*-Regelwerk und der P2P-Schicht liegenden, nicht umrahmten Stellen und Transitionen dienen lediglich der Umleitung und Ordnung von Datenflüssen innerhalb des Petrinetzes und sind inhaltlich nicht von Belang.

6.2.2 Peer-to-Peer-Schicht: *PNode, SNode, RNode*

Ein Knoten auf P2P-Schicht kann prinzipiell vier Rollen einnehmen:

1. *Publishing Node (PNode):* bringt einen neuen Datensatz in das Verzeichnis ein

2. *Requesting Node (RNode):* stellt eine Anfrage nach einem durch eine FileID identifizierten Datensatz

3. *Lookup Node (LNode):* erhält eine Lookup-Nachricht und liefert eine Menge von Daten besser passender Knoten

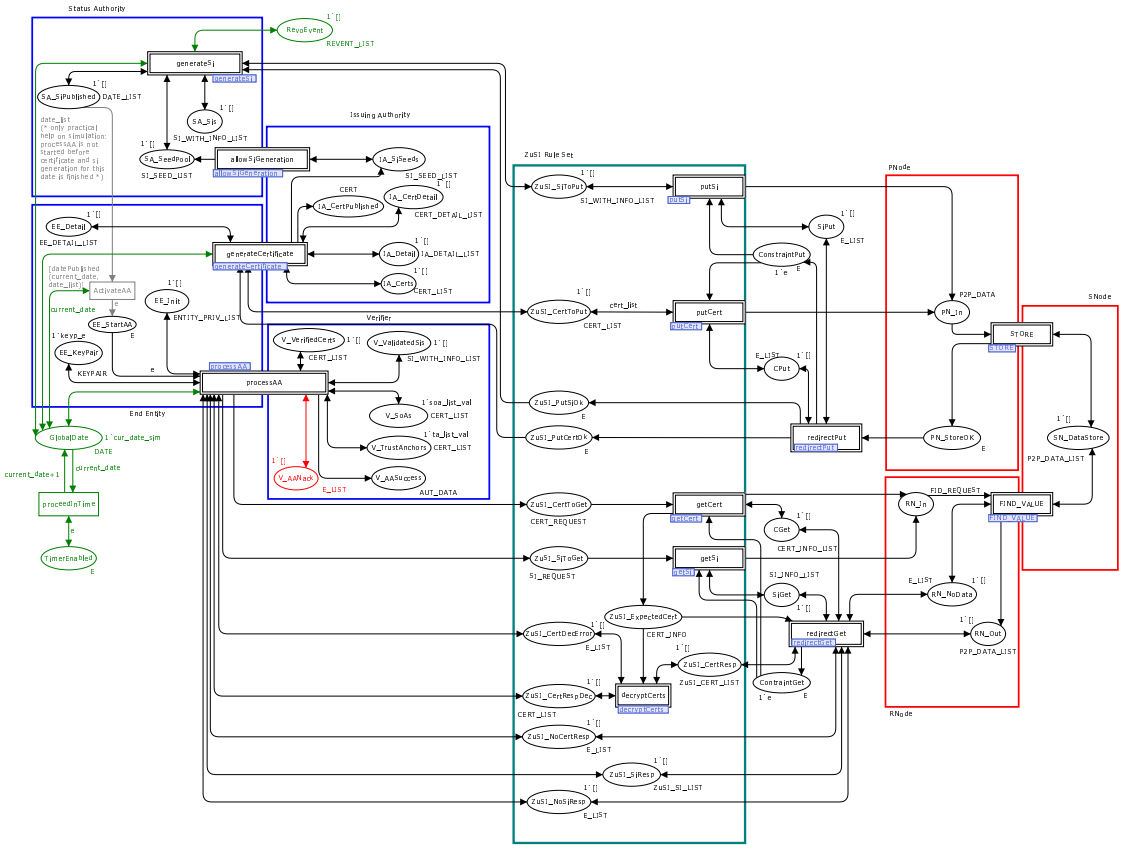


Abbildung 6.10: Top-Level-Modell

4. *Storage Node (SNode):* speichert einen Datensatz und gibt ihn auf eine entsprechende Anfrage hin zurück

Ein Knoten, welcher einen Request des Typs `FIND_VALUE` erhält und den Datensatz zurückliefert, handelt nicht in der Rolle *Lookup Node*, sondern in der Rolle *Storage Node*. Es entscheidet sich also anhand seiner Datenbasis, in welcher der beiden Rollen ein kontaktierter Knoten handelt.

In der gewählten rollenbasierten Sicht spielt die Wahl eines *konkreten* Knotens[5] für die Speicherung und Anfrage von Datensätzen keine Rolle, so dass das Routing-Verfahren zur Auffindung von zuständigen Knoten im Rahmen von Kademlia nicht relevant für das Modell ist. Der Rolle *LNode* kommt dementsprechend keine Bedeutung zu.
Auch hinsichtlich der Sicherheitsaspekte ist die Rolle *LNode* nicht notwendig, denn die Fehlverhaltensmöglichkeiten sind stark beschränkt:

- *Keine Antwort:* Ein Knoten liefert keine Antwort auf eine Anfrage. Vom Lookup-Initiator wird dann ein anderer konkreter Knoten kontaktiert. In der rollenbasierten Sicht ist die Wahl der konkreten Instanz einer Rolle aber wie erwähnt unerheblich.

[5] also einer spezifischen Knoten-Entität mit einer speziellen NodeID und IP-Adresse

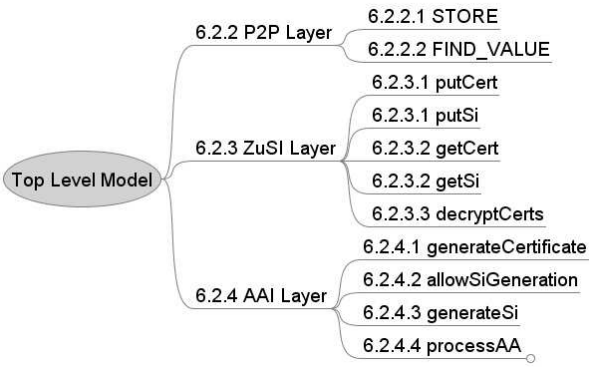


Abbildung 6.11: Subpages des Top-Level-Modells

- *Falsche Antwort:* Ein Knoten liefert falsche oder unpassende Tupel (Knoteninformationen) als Antwort auf eine Anfrage. In diesem Fall stellt der Lookup-Initiator eine Anfrage an einen anderen Knoten in der Rolle *LNode*, was wie oben keine Entsprechung in der rollenbasierten Sicht hat.

- *Falscher Datensatz:* Ein Knoten liefert einen falschen Datensatz auf eine Anfrage zurück, obwohl er nicht der zuständige Knoten für diese FileID ist. In diesem Fall agiert der Lookup Node aber in der Rolle eines Speicherknotens, d.h. diese Form des Fehlverhaltens wird automatisch abgedeckt, wenn die Rückgabe falscher Antworten in der Rolle *SNode* untersucht wird.

Diese Rolle wird folglich nicht modelliert. Damit entfällt auch die Darstellung der Methode `FIND_NODE`.
Abbildung 6.12 zeigt die drei zentralen Rollen der P2P-Schicht. Die obere Box stellt die Rolle *PNode* dar. In der Stelle PN_In werden Kombinationen von FileIDs und Daten abgelegt, die mittels der Transition *STORE* in der Rolle *SNode* gespeichert werden. Anfragen aus der Stelle *RN_In* der Rolle *RNode* werden verwendet, um in der Transition $FIND\_VALUE$ die gesuchten Datensätze zu beziehen und über *RN_Out* bzw. *RN_NoData* zurück an die entsprechende Methode des *P2P-ZuSI*-Regelwerks zu übergeben.
Die Stelle *SN_DataStore* repräsentiert den globalen Zustand des P2P-Verzeichnisses, also sämtliche auf der Gesamtheit aller Peers gespeicherten Datensätze.

6.2.2.1 Subpage *STORE*

Die P2P-Methode `STORE` wird zwischen *PNode* und dem Speicherknoten *SNode* (rechte Box) durchgeführt. Abbildung 6.13 zeigt die zugehörige Subpage *STORE*. Beim Schalten der Transition wird der Datensatz (vom Farbtyp `P2P_DATA`) der Datenbasis hinzugefügt, indem er in die Liste der gespeicherten Datensätze (Stelle *SN_DataStore* der Rolle *SNode*) aufgenommen wird. Weiterhin wird mittels einer `e`-Marke in *PN_StoreOk* quittiert, dass die Speicheraufforderung an *SNode* erfolgreich versandt wurde.

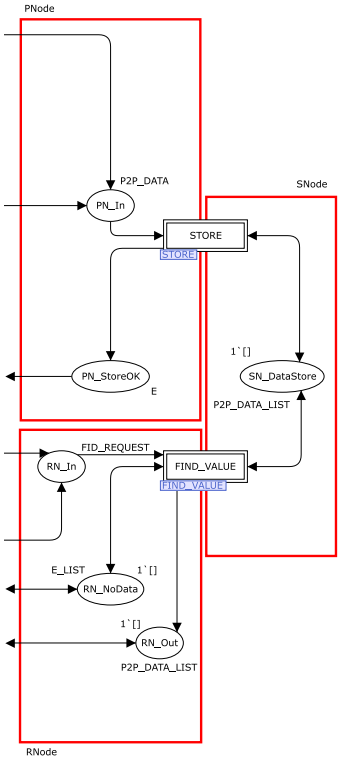


Abbildung 6.12: Peer-to-Peer-Rollen auf Top Level

6.2.2.2 Subpage *FIND_VALUE*

Abbildung 6.14 zeigt die Subpage für die Methode `FIND_VALUE`, welche zwischen *SNode* und *RNode* ausgeführt wird. Der hier erstmals verwendete Operator `#` ermöglicht die Referenzierung von Bestandteilen zusammengesetzter Farbtypen: `#1 token` liefert beispielsweise das 1. Element aus `token` zurück, `#1(#2 token)` das 1. Element des 2. Elements aus `token`. So ist die Referenzierung von Elementen in beliebiger Schachtelungstiefe möglich.

Aus *RN_In* wird die FileID für den gesuchten Datensatz geholt und gegen die gespeicherten IDs der Datensätze aus *SN_DataStore* abgeglichen (Methode `containsFid`)[6].
Je nach Anfragemethode (`getFirst` oder `getAll`) wird dabei entweder der erste gespeicherte Datensatz, auf den die FileID passt (ML-Methode `getFirstDataWithFid`), oder alle Datensätze mit passender FileID (Methode `getAllDataWithFid`) geliefert.
Es ist zu beachten, dass dies eine Vereinfachung darstellt: Die Methode `getFirst` kann in der Realität mehrere Datensätze auf eine Anfrage zurückgeben, sofern der erste Knoten, der mehrere Datensätze mit der gesuchten FileID besitzt, diese auf einen Lookup mit Methode `getFirst` hin bereitstellt. Im Modell wird aber stets maximal ein Datensatz zurückgegeben.

[6]Die konkreten ML-Methoden der P2P-Schicht finden sich in Appendix B.4.

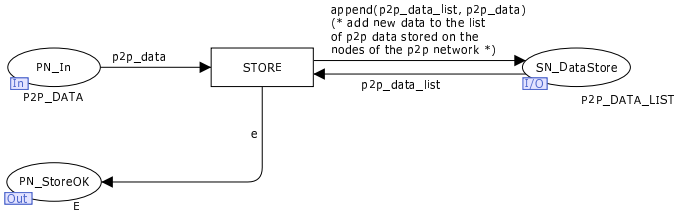


Abbildung 6.13: Subpage *STORE*

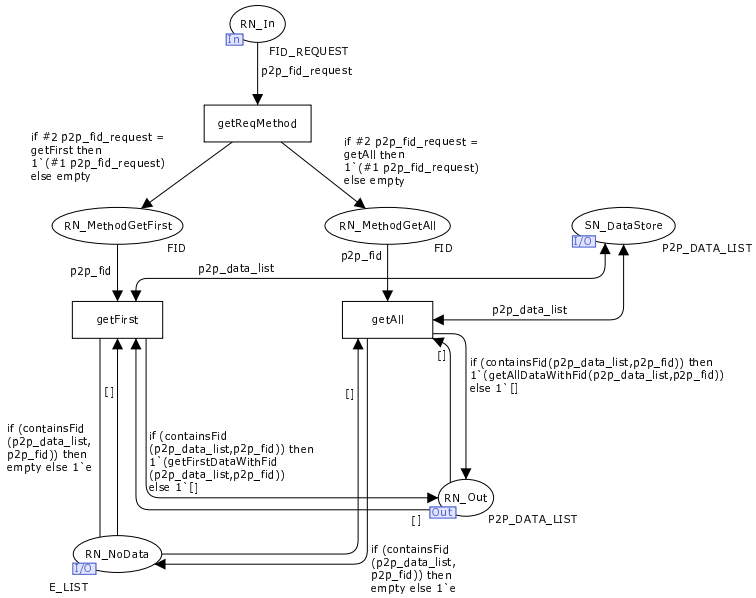


Abbildung 6.14: Subpage *FIND_VALUE*

Da der Modellfokus rollenbasiert mit globaler Sicht auf die verteilt gespeicherte Datenbasis ist, kann nicht bekannt sein, wie viele Datensätze mit gleicher FileID der erste auf ein `getFirst` antwortende Knoten gespeichert hat.
Vor dem Hintergrund der folgenden Sicherheitsanalyse (Kapitel 7) ist es für einen böswilligen Speicherknoten sogar sinnvoll, nur eine Antwort zurückzuliefern, nämlich den für seine Zwecke am besten geeigneten (z.B. modifizierten) Datensatz. Die Methode `getFirst` stellt also gegenüber der Realität keine Einschränkung der Angreifermöglichkeiten dar und ist somit eine zulässige Vereinfachung.
Die aus *SN_DataStore* extrahierte Antwort auf die Anfrage wird in *RN_Out* abgelegt. Existiert kein Datensatz mit der gesuchten FileID, so wird als Indikator für einen nicht vorhandenen Datensatz ein `e`-Token in *RN_NoData* übertragen.

6.2.3 *P2P-ZuSI*-Regelwerk-Schicht

Die *P2P-ZuSI*-Regeln setzen sich wie bereits in Abschnitt 5.5 erläutert aus den Methoden *putSi* und *putCert* für die Publikation von Zertifikaten und Statusinformationen und den Methoden *getSi* und *getCert* zur Datenabfrage zusammen. Diese werden als Substitutionstransitionen wie in Abbildung 6.15 dargestellt modelliert.

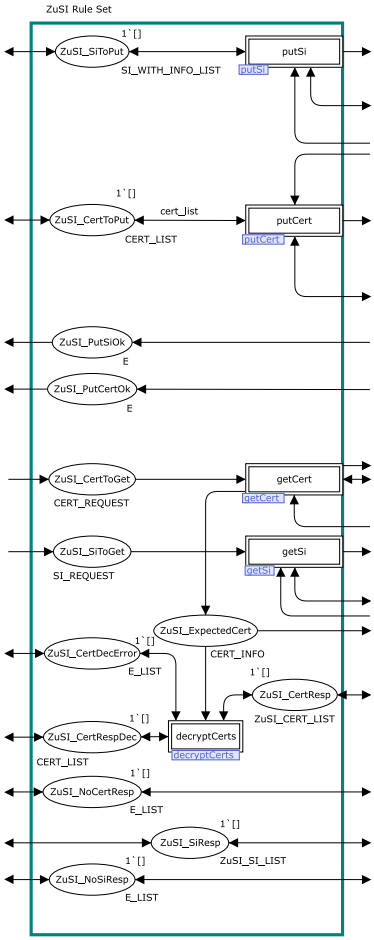


Abbildung 6.15: *P2P-ZuSI*-Regelwerk auf Top Level

6.2.3.1 Subpages *putCert* und *putSi*

Die Abbildungen 6.16 und 6.17 zeigen die Subpages zu *putCert* und *putSi*. Die an die P2P-Schicht zur Publikation übergebenen Datensätze sind vom Farbtyp `P2P_DATA`, d.h. ein 2-Tupel aus FileID und Inhalt (Statusinformation oder Zertifikat).
In *encryptCert* auf Subpage *putCert* wird das Zertifikat mit einem symmetrischen Verfahren wie in 5.3.2.2 beschrieben verschlüsselt (ML-Methode `encryptCert`).
In *putSi* wird die FileID aus der Periode, für welche die Statusinformation gilt, und dem

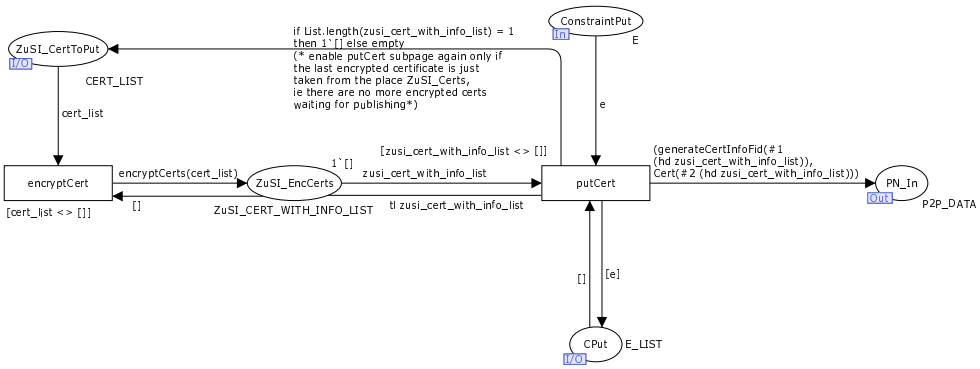


Abbildung 6.16: Subpage *putCert*

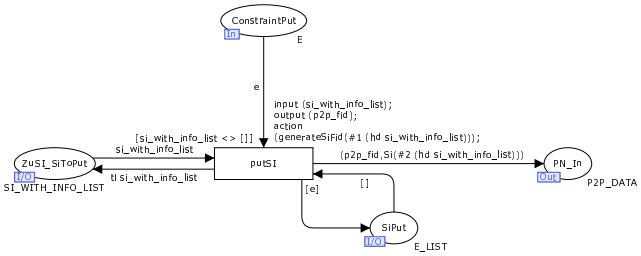


Abbildung 6.17: Subpage *putSi*

Hashwert des Zertifikats gebildet (Methode `createSiFid`[7]). In *putCert* werden analog in der Methode `createCertFid` Inhaber und Attributtyp verwendet.

6.2.3.2 Subpages *getCert* und *getSi*

Die Subpages zu den Anfragemethoden *getCert* und *getSi* sind in den Abbildungen 6.18 und 6.19 abgebildet. Hier werden die gesuchten FileIDs gebildet (`generateCertInfoFid` und `generateSiFid`) und an die Stelle *RN_In* (P2P-Rolle *RNode*) weitergeleitet.

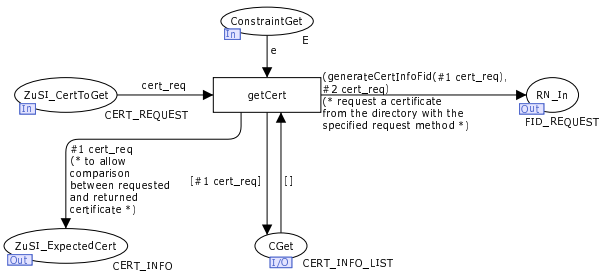


Abbildung 6.18: Subpage *getCert*

[7] siehe Appendix B.3

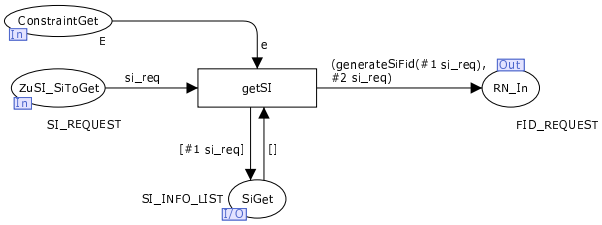


Abbildung 6.19: Subpage *getSi*

6.2.3.3 Subpage *decryptCerts*

Wird auf eine Anfrage über *getCert* eine Antwort geliefert (Liste von Zertifikaten), muss diese noch entschlüsselt werden. Auch dies geschieht innerhalb der *P2P-ZuSI*-Regelwerk-Schicht in der Transition *decryptCerts* (Methode `decryptCertsFromEntity`, die Inhaber und Attribut-/Privilegtyp der Anfrage als Parameter verwendet) wie in Abbildung 6.20 dargestellt.

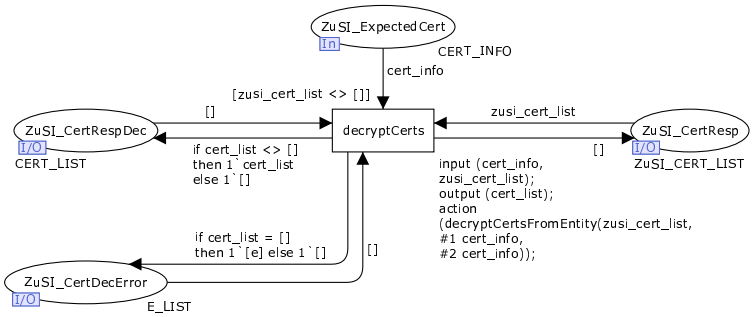


Abbildung 6.20: Subpage *decryptCerts*

6.2.4 AAI-Schicht: *Issuing Authority, Status Authority, End Entity, Verifier*

Die Rollen der AAI-Schicht interagieren miteinander und unter Verwendung des *P2P-ZuSI*-Regelwerks mit dem verteilten Peer-to-Peer-Verzeichnis. Abbildung 6.21 zeigt diese zentralen Methoden und Datenflüsse.

Die grün gefärbten Transitionen und Stellen außerhalb der blau umrahmten AAI-Rollen symbolisieren externe, globale Ereignisse und Gegebenheiten: in der Stelle *GlobalDate* wird das aktuelle Datum vorgehalten. Es wird innerhalb der AAI benötigt, um beispielsweise den Ausstellungszeitpunkt eines neu generierten Zertifikats zu berechnen oder den Wert i für eine Statusinformation zu determinieren. Die Transition *proceedInTime* ermöglicht die Inkrementierung des globalen Datums.

Weiterhin existiert die Stelle *RevoEvent*, welche eine Liste von Rückrufereignissen spei-

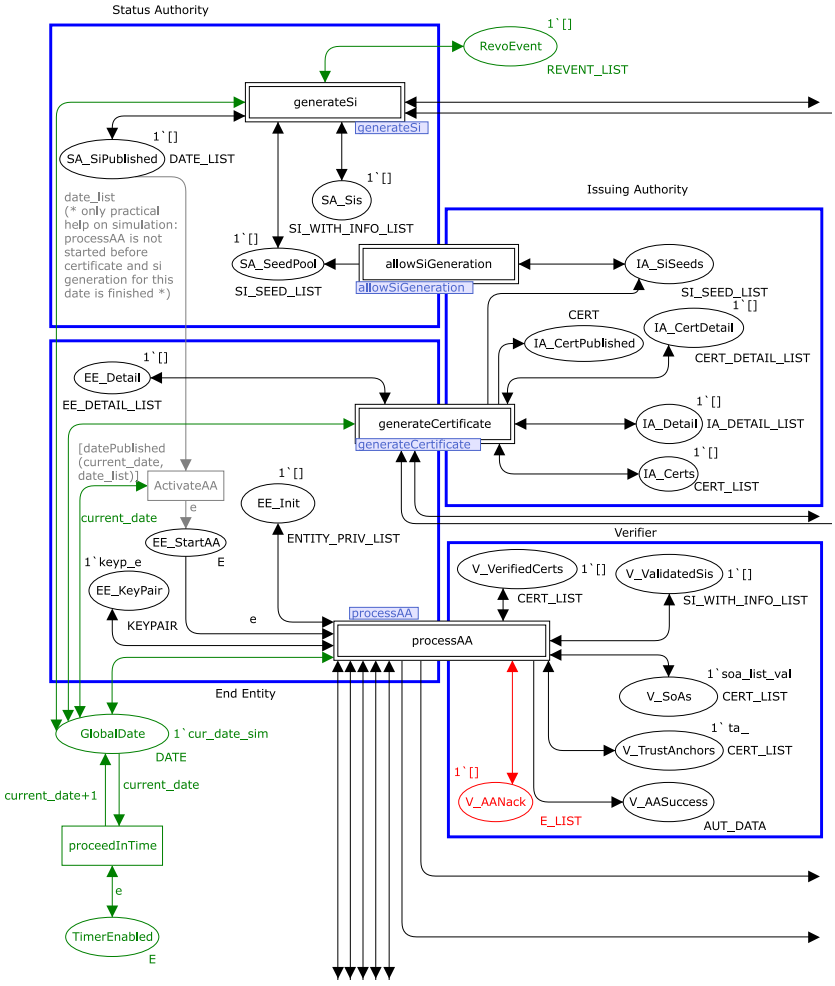


Abbildung 6.21: AAI auf Top Level

chert. Ein Rückrufereignis beinhaltet in der gewählten Form ein Datum und den Hashwert des Zertifikats, auf das es sich bezieht. Die Rückrufereignisse werden im Modell nicht von einer AAI-Rolle ausgelöst, sondern im Rahmen des Initial Marking vorgegeben. Es ist also als außerhalb der konkreten Rollen entspringendes Ereignis modelliert und abstrahiert alle möglichen Umstände, wie der tatsächliche Rückruf (z.B. die Information über ein kompromittiertes Schlüsselpaar) an eine Status Authority herangetragen werden kann. In der Transition *generateSi* wird dann für jedes Zertifikat, für welches eine Statusinformation generiert werden soll, geprüft, ob ein Rückrufereignis stattgefunden hat und damit die Publikation von Statusinformationen für dieses Zertifikat einzustellen ist.

Die Aktivitäten der AAI-Rollen lassen sich in zwei Abschnitte unterteilen:

1. *Datenerzeugung:* Erzeugung von Zertifikaten und Statusinformationen modelliert durch die Subpages *generateCertificate* und *generateSi*. Eine Entität der Rolle SA darf nur dann überhaupt Statusinformationen erzeugen, wenn sie dazu von einer

Entität in Rolle IA beauftragt wurde und die dazu notwendigen Daten (Zufallszahlen, Zertifikatinformationen) erhalten hat (Subpage *allowSiGeneration*).

2. *Datenverwendung:* Authentifizierungs- und Autorisierungsprozess dargestellt mittels der Subpage *processAA*.

Es wird angenommen, dass zu einem globalen Datum immer zuerst Abschnitt 1 und danach Abschnitt 2 durchlaufen wird. Nur so sind alle aktuellen Daten für die Durchführung der Authentifizierung und Autorisierung vorhanden. Den zweiten Abschnitt zuerst zu durchlaufen wäre sinnlos, da dies immer zur erfolglosen Beendigung des A&A-Verfahrens führen würde, da dann keine aktuellen Statusinformationen vorhanden wären.
Die Wahrung der Reihenfolge wird praktisch durchgeführt, indem die Stelle *EE_StartAA* erst dann mit einer Marke belegt wird, wenn in der Transition *generateSi* alle Statusinformationen des aktuellen Datums erzeugt und publiziert wurden. Dadurch wird der innerhalb der einzelnen Prozesse der AAI gewahrte Determinismus auf die gesamte AAI erweitert, d.h. es wird eine deterministische Reihenfolge für die Ausführung dieser Einzelprozesse erzeugt.

6.2.4.1 Subpage *generateCertificate*

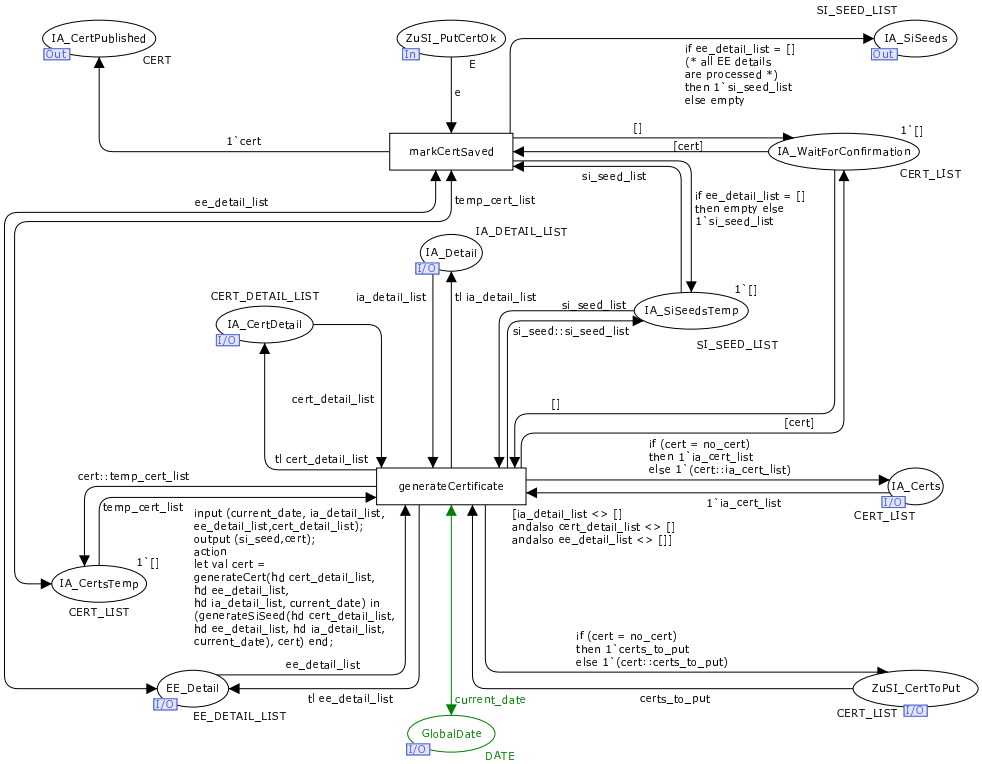


Abbildung 6.22: Subpage *generateCertificate*

Abbildung 6.22 zeigt die Subpage für die Erzeugung von Zertifikaten. Dabei wird zunächst in Transition *generateCertificate* aus von der End-Entität (in Stelle *EE_Detail*) bereit-

gestellten und der IA selbst bekannten Daten (Stellen *IA_Detail*, *IA_CertDetail*) unter Verwendung der Methode `generateCert`[8] ein Zertifikat erzeugt. Dann wird es an die *P2P-ZuSI*-Methode *putCert* übergeben (Ausgabestelle *ZuSI_CertToPut*) und auf die Speicherbestätigung gewartet (Eingabestelle *ZuSI_PutCertOk*).
Sind alle auf diese Art generierten Zertifikate publiziert, wird dies intern vermerkt (Ausgabestelle *IA_CertPublished*) und die Daten zur Erzeugung von Statusinformationen für die Übertragung an SA vorbereitet (Ausgabestelle *IA_SiSeeds*). Der für letzteres verwendete Datentyp `SI_SEED` beinhaltet Hashwert, Gültigkeitsbeginn und Gültigkeitsende des Zertifikats sowie die Zufallszahl, welche als Basis für die Berechnung des Validierungsziels verwendet wurde.

6.2.4.2 Subpage *allowSiGeneration*

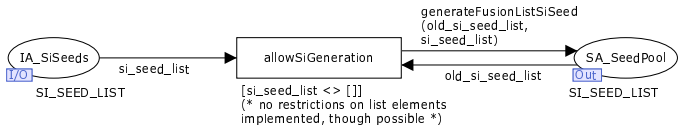


Abbildung 6.23: Subpage *allowSiGeneration*

In Abbildung 6.23 ist die Übertragung der Daten zur SI-Erzeugung in der Subpage *allowSiGeneration* dargestellt. Diese ist absichtlich sehr einfach gehalten: die neuen Informationen werden lediglich den in *SA_SeedPool* (also der Stelle, in welcher die Gesamtheit der `SI_SEED`s aller Entitäten in der Rolle SA erfasst sind) bereits gespeicherten Daten hinzugefügt.

6.2.4.3 Subpage *generateSi*

Die Erzeugung von Statusinformationen für die aktuelle Periode wird in der Subpage *generateSi* wie in Abbildung 6.24 durchgeführt. Es wird angenommen, dass die Ausstellungsperioden synchronisiert sind: die Erzeugung von Statusinformationen für alle Zertifikate findet täglich statt.
Zunächst werden unter Verwendung des globalen Datums aus *GlobalDate* die SI-Seeds aussortiert, welche zu regulär abgelaufenen Zertifikaten gehören oder per Rückrufereignis (aus Stelle *RevoEvent*) zurückgerufen wurden (Methode `cleanSiSeedList`).
Dann werden aus allen übrigen, gültigen `SI_SEED`s aktuelle Statusinformationen erzeugt (Methode `generateCurrentSis`) und mittels der *P2P-ZuSI*-Methode *putSi* im Verzeichnis publiziert (Ausgabestelle *ZuSI_SiToPut*).
Ist die Publikation für diese Statusinformationen erfolgreich abgeschlossen, wird die Transition *cleanSeedList* für die nächste Periode freigegeben, indem eine Marke vom Typ `DATE_LIST`, welche alle Publikationsperioden der Vergangenheit beinhaltet, in die Stelle *SA_CleanActive* übertragen wird.

[8] siehe Appendix B.2.4

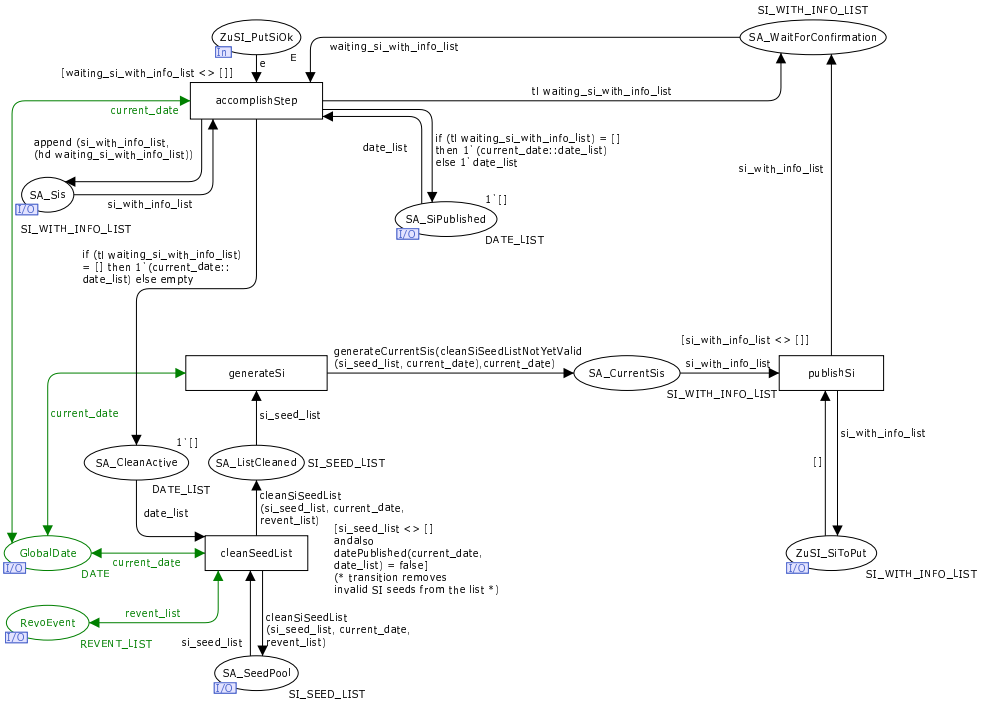


Abbildung 6.24: Subpage *generateSi*

6.2.4.4 Authentifizierung und Autorisierung auf Subpage *processAA*

Abbildung 6.25 visualisiert den hierarchischen Aufbau der Pages auf AAI-Level. Anhand der hier dargestellten Hierarchie werden nun sukzessive die verschachtelten Subpages vorgestellt.

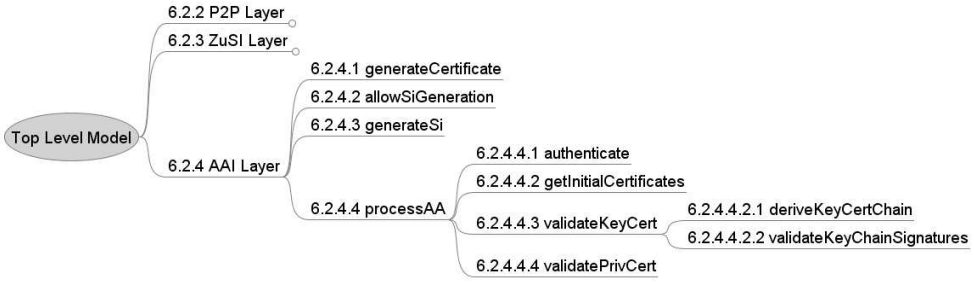


Abbildung 6.25: Seitenhierarchie : AAI-Schicht

Abbildung 6.26 zeigt die Page *processAA*. Die erste Transition *extractConcreteEEToProceed* wählt aus allen in *EE_Init* vorhandenen Daten vom Typ `ENTITY_PRIV` (Paare aus Entitätsbezeichner und beanspruchtem Privilegtyp von konkreten End-Entitäten) das erste Paar aus. So wird von einer allgemeinen rollenbasierten Sicht auf die AAI als Gan-

zes auf den Authentifizierungs- und Autorisierungsvorgang einer *konkreten* End-Entität E gegenüber einem Prüfer übergegangen.

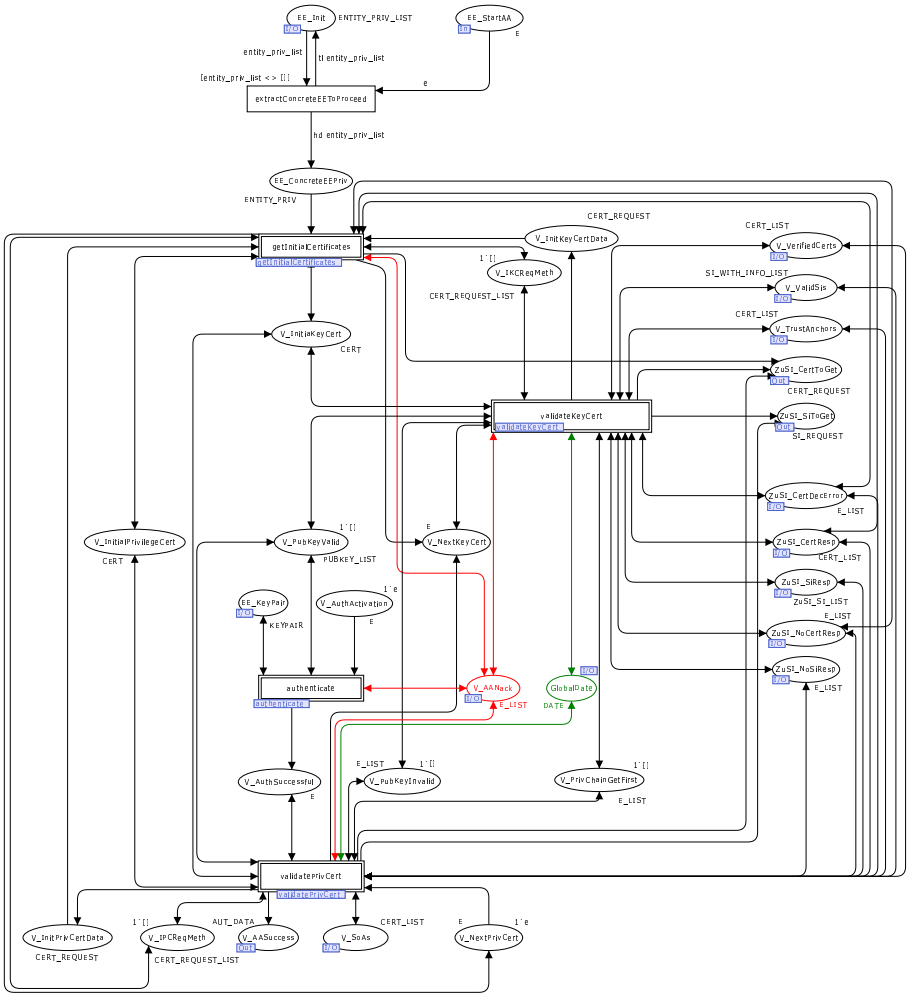


Abbildung 6.26: Subpage *processAA*

Zunächst versucht der Prüfer, zu den gegebenen Informationen über die End-Entität und über den von ihr beanspruchten Privilegtyp Initialzertifikate aus dem Verzeichnis zu beziehen, also Zertifikate, die dann als Ausgangspunkte der Kettenerzeugungsprozesse verwendet werden. Dies wird in der Substitutionstransition *getInitialCertificates* durchgeführt, in der pro Durchgang je ein Initialzertifikat pro Typ (Schlüssel/Privileg) gefunden werden kann (siehe Abschnitt 6.2.4.4.2 im Detail).
Ist dies erfolgreich abgeschlossen, wird das ermittelte initiale Schlüsselzertifikat in der Substitutionstransition *validateKeyCert* auf seine Gültigkeit geprüft und damit die Authentizität des enthaltenen öffentlichen Schlüssels der End-Entität verifiziert oder falsifiziert (siehe 6.2.4.4.3). Ist ein Schlüssel als authentisch bestätigt, führt der Prüfer V im Rahmen von *authenticate* ein Authentifizierungsverfahren mit E durch.

Dann wird das Privilegzertifikat validiert und somit geprüft, ob die authentifizierte End-Entität das im Zertifikat enthaltene - und somit für sich beanspruchte - Privileg tatsächlich besitzt. Wenn ja, wird dies in der Ausgabestelle *V_AASuccess* vermerkt. Eine erfolglose Authentifizierung und Autorisierung wird in der rot hervorgehobenen Stelle *V_AANack* in Form der Liste `[e]` vermerkt. Wird in einer beliebigen Subpage eine solche Liste in *V_AANack* transferiert, so wird dadurch mit sofortiger Wirkung das Feuern beliebiger weiterer Transitionen verhindert.

6.2.4.4.1 Subpage *authenticate* (Superpage *processAA*)

Die Subpage *authenticate* (Abbildung 6.27) implementiert ein sehr einfaches Challenge-Response-Verfahren. Dafür muss dem Prüfer der authentische öffentliche Schlüssel der End-Entität vorliegen (aus *validateKeyCert*), während diese ihr Schlüsselpaar aus der Stelle *EE_KeyPair* verwendet. Es ist zu beachten, dass das Verfahren so einfach wie möglich modelliert wurde und in der dargestellten Form keinen Anspruch auf Sicherheit erhebt. Es ist als reine Abstraktion eines beliebigen sicheren Authentifizierungsverfahrens auf Basis von Public-Key-Kryptographie zu verstehen.

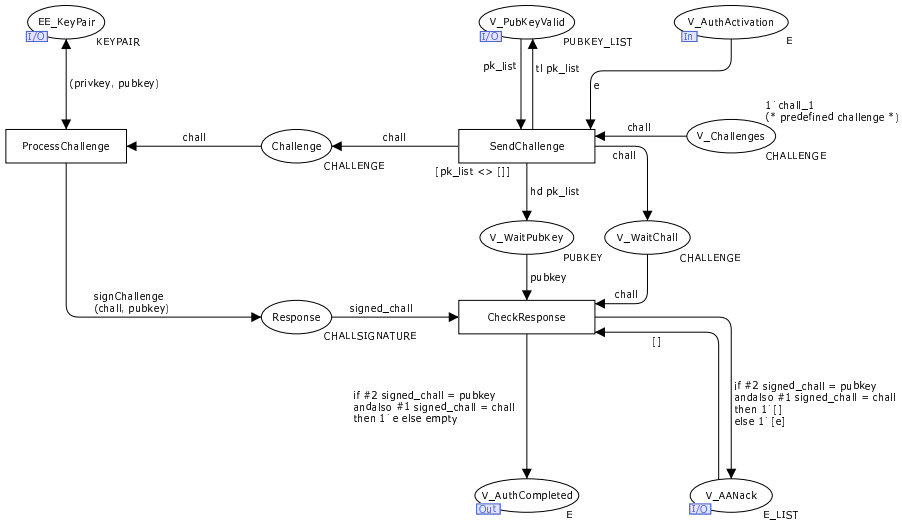


Abbildung 6.27: Subpage *authenticate*

6.2.4.4.2 Subpage *getInitialCertificates* (Superpage *processAA*)

Auf der Subpage *getInitialCertificates* (Abbildung 6.28) ermittelt der Prüfer die Initialzertifikate für die Herleitung von Schlüssel- und Privilegzertifizierungsketten. Er erhält von der End-Entität deren Entitätsbezeichner E (Inhaber der Initialzertifikate) und den für die Autorisierung benötigten Privilegtyp.

Zuerst stellt V nun über das *P2P-ZuSI*-Regelwerk eine Anfrage (Methode `getFirst`) nach einem Schlüsselzertifikat für die angegebene End-Entität (Transition *getInitKeyCert*, Stelle *ZuSI_CertToGet*). Erhält er ein solches zurück (Transition *receiveInitKeyCert*), speichert er es in der Stelle *V_InitialKeyCert*.

Ist die Antwort eine `[e]`-Liste in der Stelle *V_CertDecError*, so weiß der Prüfer, dass es

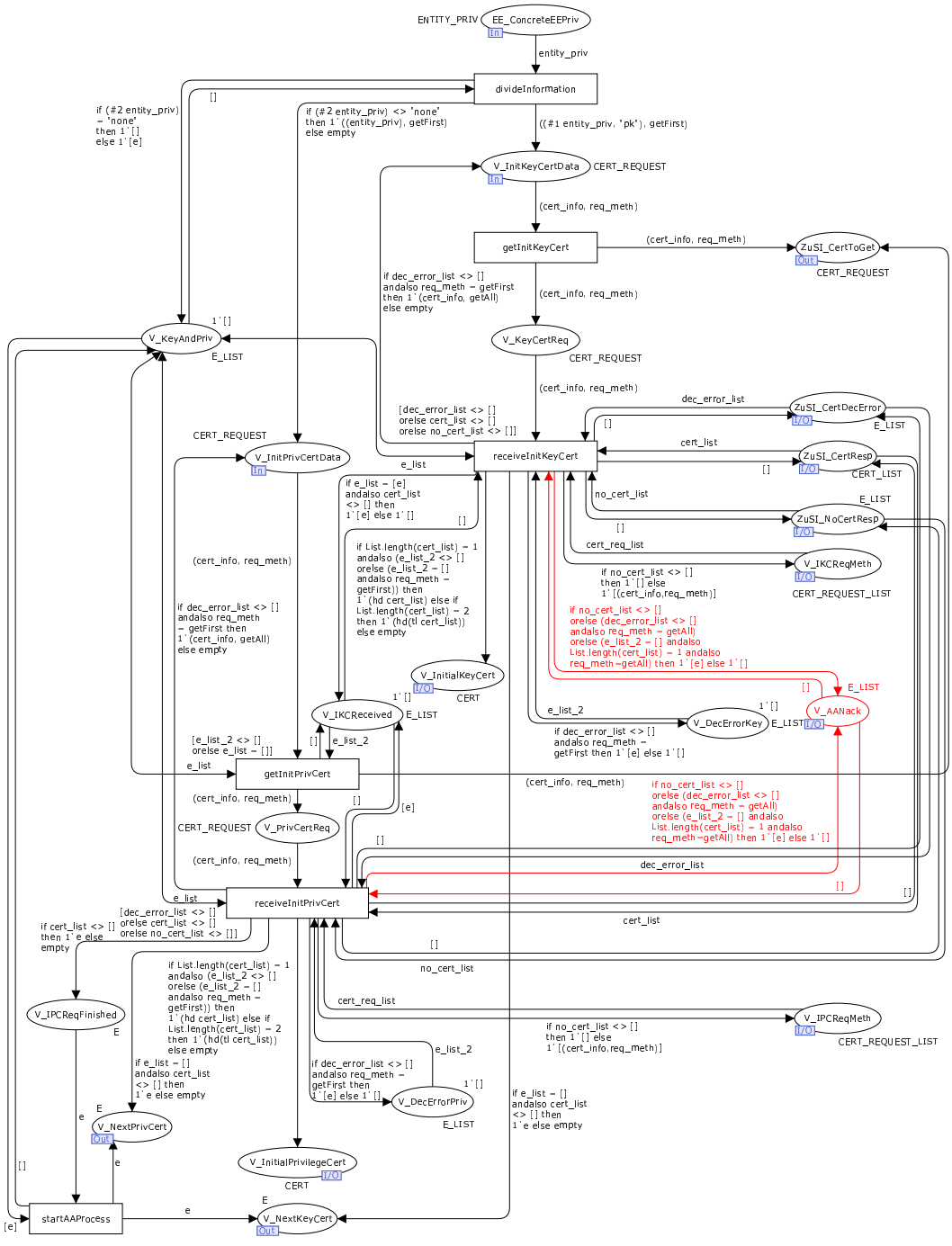


Abbildung 6.28: Subpage *getInitialCertificates*

zwar eine Antwort gab, diese aber nicht entschlüsselt werden konnte. Er vermerkt dies dann (Stelle *V_DecErrorKey*) und stellt eine neue Anfrage (*getInitKeyCert*) mit Methode `getAll`. Erhält er immer noch kein lesbares Zertifikat oder generell keine Antwort (Liste

`[e]` in Stelle *ZuSI_NoCertResp*), so bricht er den A&A-Prozess ab (Transfer von `[e]` in Stelle *V_AANack*).

In der Realität können mehrere Datensätze mit gleicher FileID existieren. Zur Vereinfachung wird aber für das Modell die Annahme getroffen, dass global maximal eine - eindeutige - Zertifizierungskette je Typ existiert. Das heißt: Es existiert nur genau ein initiales Schlüssel- und ein initiales Privilegzertifikat im Verzeichnis, dazu jeweils genau ein Ausstellerzertifikat usw. Für die spätere Sicherheitsanalyse - neben der Systembeschreibung die Hauptmotivation für das HCPN-Modell - ist nur wichtig, ob diese eine Zertifizierungskette (von einem Trust-Anchor-Zertifikat zum Initialzertifikat) erzeugt werden kann oder nicht. Möglichkeiten zur Erzeugung verschiedener Ketten erhöhen nur die Komplexität des Modells, sind zur Beantwortung dieser Frage aber nicht von Belang.
Da später in der Sicherheitsanalyse allerdings die Effekte von Zertifikatmodifikationen geprüft werden, muss man davon ausgehen, dass mehrere Antworten auf eine Anfrage geliefert werden, nämlich das korrekte Zertifikat und ein oder mehrere modifizierte Versionen. Im Vorgriff auf Kapitel 7 sei hier erwähnt, dass in den verschiedenen in der Sicherheitsanalyse betrachteten Angriffsfällen jeweils nur *eine* Art der Modifikation untersucht wird. Damit genügt es, von einem einzigen modifizierten Zertifikat auszugehen, und es können maximal zwei Antworten auf eine Anfrage existieren.
Stellt der Prüfer nun in *getInitialKeyCert* eine Anfrage mit `getFirst`, so erhält er wegen der Konzeption dieser Methode maximal eine Antwort. Fragt er später nochmals mit `getAll`, so kann er aufgrund der obigen Annahme höchstens zwei Antworten erhalten. Die erste hat er bereits erfolglos verarbeitet (sonst wäre eine Anfrage mit `getAll` ja nicht nötig). Liefert eine Zertifikatanfrage mit `getAll` also zwei Datensätze zurück, so muss der erste nicht mehr berücksichtigt werden. Ein Prüfer wird diesen also ignorieren und nur das zweite - noch unbekannte - Zertifikat verwenden.
Dieses Prinzip führt dazu, dass bei allen Zertifikatanfragen an das Verzeichnis auf der aktuellen Subpage ebenso wie auf den Subpages *deriveKeyCertChain* und *validatePrivCert* stets maximal ein einziges Zertifikat weiter verarbeitet wird.

Konnte der Prüfer schließlich ein initiales Schlüsselzertifikat beziehen, aktiviert er die Transition *getInitPrivCert*, indem er die Stelle *V_IKCReceived* mit einer `[e]`-Liste belegt. In *getInitPrivCert* wird dann analog zu *getInitKeyCert* verfahren, um ein initiales Privilegzertifikat für E und den angegebenen Privilegtyp zu ermitteln. Gelingt dies, werden in der Transition *startAAProcess* die Subpages für die Kettengenerierung aktiviert, indem die Stellen *V_NextKeyCert* und *V_NextPrivCert* mit `e`-Token belegt werden.

Stellt sich später im Verlauf der Durchführung der Kettenerzeugung für Schlüssel- oder Privilegzertifikate heraus, dass aus dem initialen Zertifikat keine korrekte Kette hergeleitet werden kann, wird auf Subpage *getInitialCertificate* entweder *getInitKeyCert* oder *getInitPrivCert* erneut mit Anfragemethode `getAll` durchlaufen und dann die Subpage wieder aktiviert, in welcher die erfolglose Kettengenerierung stattfand (Subpage *validateKeyCert* oder *validatePrivCert*).
Der Vermerk, mit welcher Methode das aktuell verarbeitete Initialzertifikat angefragt wurde, ist daher von essentieller Bedeutung; dies speichern die Stellen *V_IKCReqMeth* und *V_IPCReqMeth*.

6.2.4.4.3 Subpage *validateKeyCert* (Superpage *processAA*)

Die Page *validateKeyCert* besitzt zwei Subpages (siehe Abb. 6.29): *deriveKeyCertChain* zur Erzeugung einer Kette von Schlüsselzertifikaten und *validateKeyChainSignatures*, in welcher die Signaturen auf den Zertifikaten einer Kette geprüft werden.

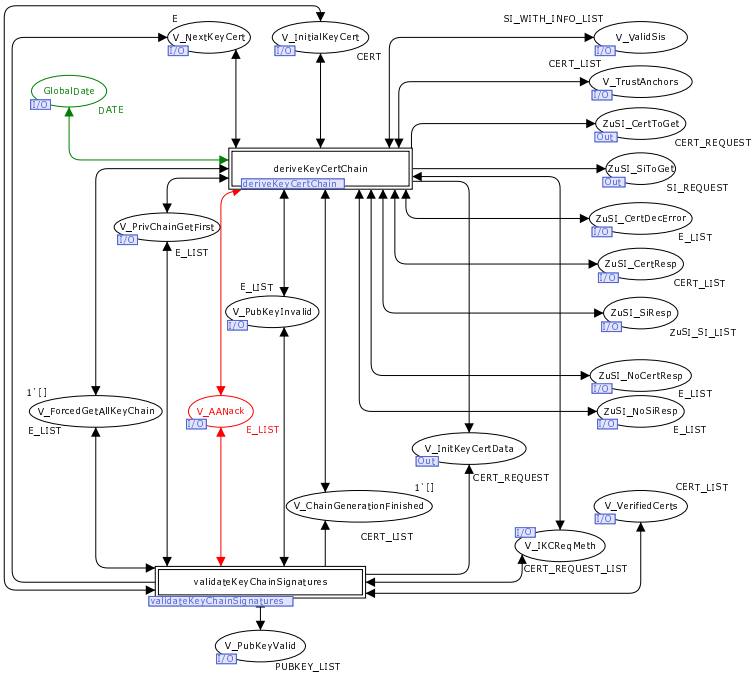


Abbildung 6.29: Subpage *validateKeyCert*

6.2.4.4.3.1 Subpage *deriveKeyCertChain* (Superpage *validateKeyCert*)

Wegen der Komplexität der Subpage *deriveKeyCertChain* (Abbildung 6.31 auf Seite 153) wird der wesentliche Aufbau ausführlich erläutert. Es ist zu beachten, dass dabei einige Details des Modells aus Vereinfachungsgründen ausgelassen werden.

Die Seite implementiert einen iterativen Prozess:

1. hole und prüfe die Statusinformation des aktuellen Zertifikats
2. füge das Zertifikat zur Zertifizierungskette hinzu, wenn kein Zyklus vorliegt
3. hole ein Ausstellerzertifikat zum aktuellen Zertifikat und setze dieses als neues aktuelles Zertifikat

Als erstes wird das Initialzertifikat als aktuelles Zertifikat verarbeitet, als letztes das vom Prüfer ausgestellte, lokal gespeicherte Trust-Anchor-Zertifikat an die Kette angehangen (sofern sie vervollständigt werden kann). Es ist Zyklenerkennung implementiert: Ist ein Zertifikat bereits in der Kette erfasst und soll nun ein zweites Mal hinzugefügt werden, liegt ein solcher Zyklus vor und der Prozess wird abgebrochen.

Abbildung 6.30 veranschaulicht dieses Vorgehen und illustriert damit die wesentlichen Datenflüsse auf der Subpage.

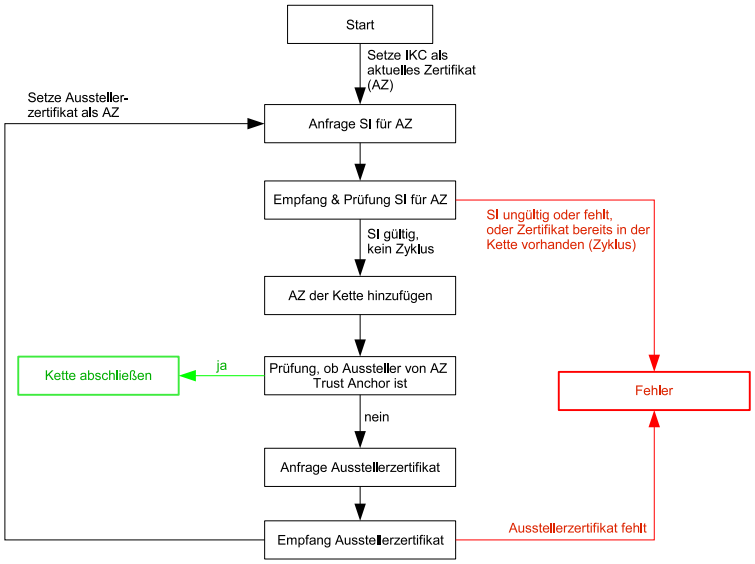


Abbildung 6.30: Skizze der Datenflüsse auf Subpage *deriveKeyCertChain*

Die Verarbeitung beginnt mit der Transition *checkIfKeyCert*, in der von V geprüft wird, ob das gegebene Initialzertifikat ein Schlüsselzertifikat ist. Wenn ja, wird es in die Stelle *V_StartingPointSiRequest* übertragen, welche auch den Startpunkt jeder Iteration darstellt, d.h. jedes empfangene Ausstellerzertifikat wird später von *receiveIssuerCertificate* in genau diese Stelle übertragen.
Dann wird über *P2P-ZuSI* eine Anfrage (mit `getFirst`) nach der aktuellen Statusinformation für das Zertifikat gestellt. Die Antwort wird in *checkRequestedSis* geprüft. Ist eine gültige Statusinformation vorhanden, wird der Prozess fortgesetzt (Ausgabestelle *V_SiValid*). Außerdem wird das Zertifikat in der Zertifizierungskette gespeichert (Ausgabestelle *V_ToAdd*, Transition *addCertToChain* inkl. Zyklusprüfung durch Methode `containsCert`).
Wird hingegen auf die SI-Anfrage hin keine gültige SI als Antwort geliefert, wiederholt V die Anfrage an das Verzeichnis mit der Methode `getAll`. Wird überhaupt keine passende Statusinformation gefunden oder ist auch nach einer Anfrage mit `getAll` keine gültige SI in der Antwortliste `si_with_info_list` vorhanden, wird die Fehlerbehandlung angestoßen (Ausgabestelle *V_Failure*), welche dann in *handleMissingDataOrFailure* durchgeführt wird. Dies geschieht aber nur dann, wenn es keine Möglichkeit gibt, das Zertifikat, dessen Statusinformation es hier zu bestimmen gilt, nochmals neu mit `getAll` anzufragen (Bogen von *checkRequestedSis* zur Stelle *V_IssuerCertNeeded*). Hierfür ist der aktuelle Inhalt der Stelle *V_LastKeyCertReqMethod* von Belang, welche speichert, mit welcher Methode das aktuelle Zertifikat angefragt wurde (zur Erläuterung im Detail siehe Transition *receiveIssuerCertificate*).
Der nächste Schritt nach erfolgreicher SI-Prüfung ist der Abgleich des Zertifikatausstellers

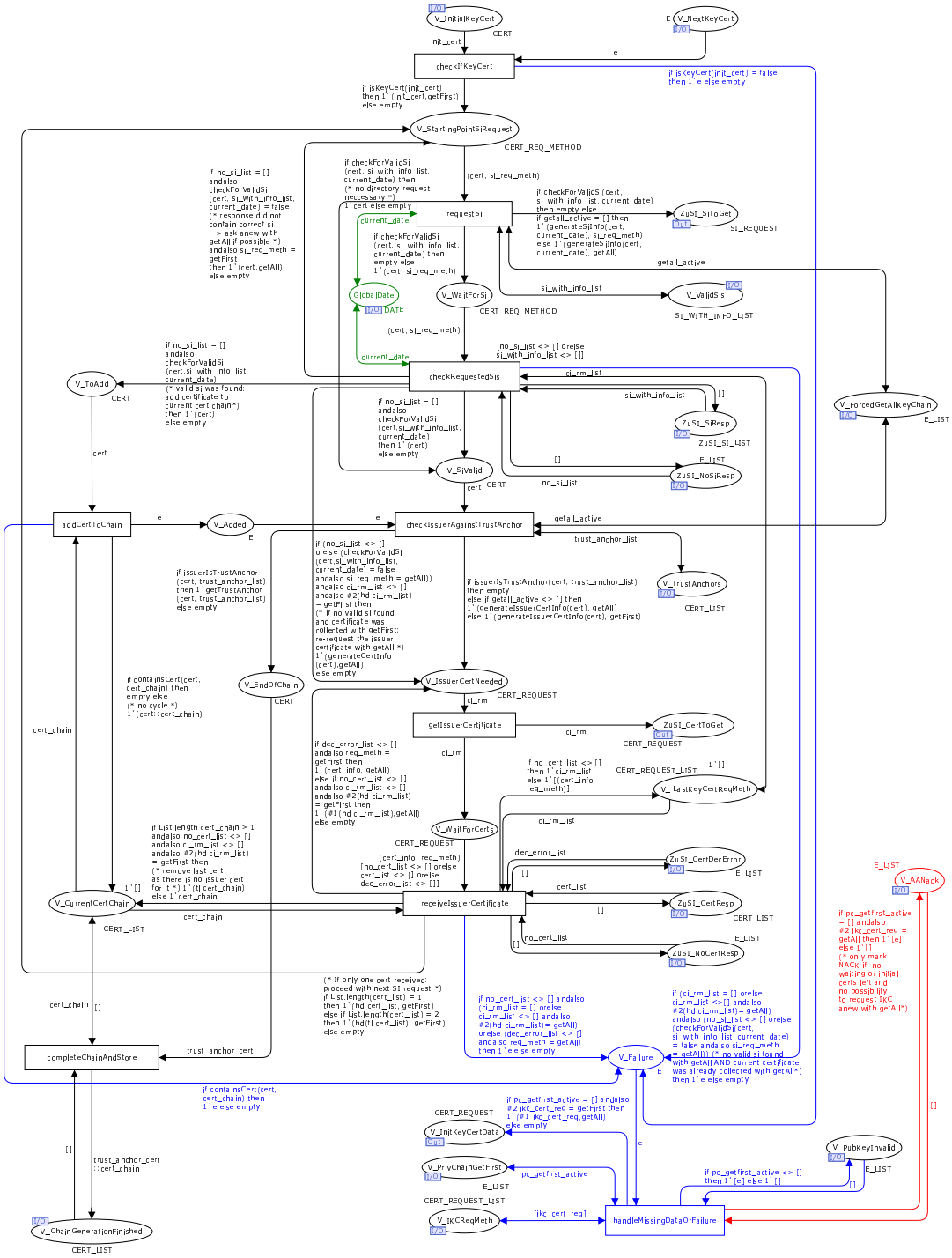


Abbildung 6.31: Subpage *deriveKeyCertChain*

mit der Liste bekannter Trust Anchors. Ist der Aussteller ein Trust Anchor, so die Kette mit dem zugehörigen TA-Zertifikat abgeschlossen (Ausgabestelle *V_EndOfChain*

und Transition *completeChainAndStore*). Ansonsten ist es notwendig, ein Schlüsselzertifikat anzufragen, welches als Inhaber den Aussteller des zuletzt geprüften Zertifikats hat. Dies wird in der Transition *getIssuerCertificate* umgesetzt und das Ergebnis in *receiveIssuerCertificate* ausgewertet.
Für *receiveIssuerCertificate* spielt die Stelle *V_LastKeyCertReqMeth* eine tragende Rolle. Hier wird - in einer maximal ein-elementigen Liste des Farbtyps `CERT_REQUEST` (zusammengesetzt aus `CERT_INFO` und `REQ_METHOD`) - gespeichert, mit welcher Methode (`getFirst`/`getAll`) das *zuletzt aus dem P2P-Verzeichnis erhaltene* Schlüsselzertifikat angefragt wurde. Wird von der *P2P-ZuSI*-Methode *getCert* irgendein Datensatz oder aber ein Entschlüsselungsfehler (Indikator für einen erhaltenen Datensatz, der bloß nicht dechiffriert werden konnte) zurückgeliefert, so wird die verwendete Methode zusammen mit der Information über das angefragte Zertifikat in *V_LastKeyCertReqMeth* vermerkt. Wird hingegen kein Datensatz zurückgegeben, so bleibt der letzte Vermerk bestehen.
Dieser Vermerk ist nun relevant für die weitere Verarbeitung der Antwort auf die Zertifikatanfrage.

- *Fehler bei der Entschlüsselung:* Konnte eine Zertifikatantwort mit dem *P2P-ZuSI*-Regelwerk nicht entschlüsselt werden (`dec_error_list <> []`), so wird das Zertifikat nochmals mit `getAll` angefragt, sofern der aktuelle Versuch (Variable `req_meth`) mit `getFirst` durchgeführt wurde. Es wird `getFirst` und der Inhalt der Variable `cert_info` in *V_LastKeyCertReqMeth* gespeichert.

- *Keine Antwort:* Ist kein passendes Zertifikat im Verzeichnis vorhanden (Variable `no_cert_list <> []`), wird die Anfragemethode aus der vorigen Iteration analysiert (Liste `ci_rm_list` aus Eingabestelle *V_LastKeyCertReqMeth*). Es wird also geprüft, ob das Zertifikat c_{prev}, zu welchem in *getIssuerCertificate* das Ausstellerzertifikat geholt werden sollte, mit `getFirst` oder `getAll` angefragt wurde.
  Ist ersteres der Fall, wird *getIssuerCertificate* nochmals für die Inhaber-Typ-Kombination von c_{prev} (`#1(hd ci_rm_list)` im Bogen zurück zu *V_IssuerCertNeeded*) ausgeführt, diesmal aber mit `getAll`. Es handelt sich hier also um einen Rücksprung in den vorherigen Schritt der Kettenerzeugung. Dabei wird auch c_{prev} aus der aktuellen Kette entfernt, da hierzu ja kein Ausstellerzertifikat gefunden wurde und die Kette damit nicht vollendet werden kann.
  Weiterhin existiert die Möglichkeit, dass nicht in den vorigen Schritt zurückgesprungen werden kann, da kein solcher existiert (laufende Verarbeitung des Initialzertifikats) oder im Vorgängerschritt bereits mit `getAll` angefragt wurde. Dann wird eine Fehlermeldung in *V_Failure* übertragen.

- *Zwei Antworten:* Auf eine Anfrage werden mehrere erfolgreich entschlüsselte Zertifikate zurückgeliefert. Da dies aber nur bei Anfrage mit `getAll` vorkommen kann und in der - erfolglosen - Verarbeitung der vorhergegangenen `getFirst`-Anfrage bereits das erste dieser beiden Zertifikate verwendet wurde, ist das erste Listenelement nicht erneut zu berücksichtigen. Es wird in diesem Fall also einfach ignoriert und mit dem zweiten Element (`hd(tl cert_list)`) wieder oben im Netz mit der Anfrage nach einer aktuellen Statusinformation begonnen (Rücksprung zur Stelle *V_StartingPointSiRequest*).

Die Fehlerbehandlung ist auf dieser Subpage in blau dargestellt. Tritt ein Fehler auf,

wird in *handleMissingDataOrFailure* vermerkt, dass die Authentifizierung gescheitert ist: Wenn die Kettenerzeugung im Rahmen einer Privilegkettenvalidierung wie in Abschnitt 6.2.4.4.4 beschrieben angestoßen wurde, wird eine Marke in die Stelle *V_PubKeyInvalid* übertragen, welche dann in der Subpage *validatePrivCert* weiter verarbeitet wird. Ansonsten wird, wenn möglich, in die Page *getInitialCertificates* zurückgesprungen, um nochmals mit `getAll` nach einem weiteren initialen Schlüsselzertifikat zu suchen. Sonst übermittelt *handleMissingDataOrFailure* eine `[e]`-Marke in die Stelle *V_AANack* und terminiert damit den gesamten A&A-Vorgang.

Der iterative Prozess wird also so lange durchgeführt, bis er entweder durch einen Fehler abgebrochen oder die aktuelle Zertifizierungskette erfolgreich mit einem Trust-Anchor-Zertifikat (ausgestellt vom Prüfer selbst) abgeschlossen wird. Im letzteren Fall wird die Transition *completeChainAndStore* gefeuert. Der Prüfer speichert die vollständige Kette in der Stelle *V_ChainGenerationFinished*, der Kettenerzeugungsprozess ist damit erfolgreich beendet.

6.2.4.4.3.2 Subpage *validateKeyChainSignatures* (Superp. *validateKeyCert*)

Die Seite *validateKeyChainSignatures* (Abbildung 6.32) wird aktiviert, sobald die Kettenerzeugung in *deriveKeyCertChain* erfolgreich abgeschlossen und damit eine Marke in der i/o-Port-Stelle *V_ChainGenerationFinished* vorhanden ist.
Die Transition *beginValidation* extrahiert diese Zertifizierungskette und startet damit den Validierungsprozess. In *validateSignature* wird dann das jeweils erste Zertifikat der Liste (beginnend mit dem Trust-Anchor-Zertifikat) abgetrennt und mit dessen öffentlichem Schlüssel die Signatur des neuen Anfangszertifikat der Liste geprüft. Man beginnt also damit, das zweite Zertifikat der Kette mit dem öffentlichen Schlüssel des Trust Anchors zu validieren, danach das dritte mit dem Schlüssel des (nun validierten) bisher zweiten usw.
Jedes erfolgreich geprüfte Zertifikat wird in *V_VerifiedCerts* gespeichert. Ist das letzte Zertifikat einer Kette (also das ursprüngliche Initialzertifikat) erfolgreich geprüft, wird der dort enthaltene öffentliche Schlüssel als gültig angesehen (Ausgabestelle *V_PubKeyValid*).

Ist die Validierung der Kette *nicht* erfolgreich, so können verschiedene Pfade eingeschlagen werden:

1. *Erneute Kettenerstellung mit erzwungenem* `getAll` (Stellen *V_InitialKeyCert* und *V_ForcedGetAllKeyChain*): Die Subpage *V_deriveKeyCertChain* wird nochmals von Anfang an mit dem im letzten Versuch verwendeten Initialzertifikat durchlaufen. Diesmal werden *alle* Anfragen an das Verzeichnis direkt mit `getAll` durchgeführt, um definitiv alle Daten zu berücksichtigen.

2. *Erneute Anfrage nach initialem Schlüsselzertifikat:* Ist Möglichkeit 1 bereits erfolglos versucht worden, wird nochmals die Subpage *getInitialCertificates* durchlaufen. Dort wird mit `getAll` nach einem passenden Initialzertifikat gesucht. Wird dabei eines gefunden, das bisher noch nicht verarbeitet wurde, wird damit die Page *deriveKeyCertChain* erneut ausgeführt.

3. *Erkläre Ungültigkeit des Schlüssels:* Wenn der Schlüsselvalidierungsprozess auf der

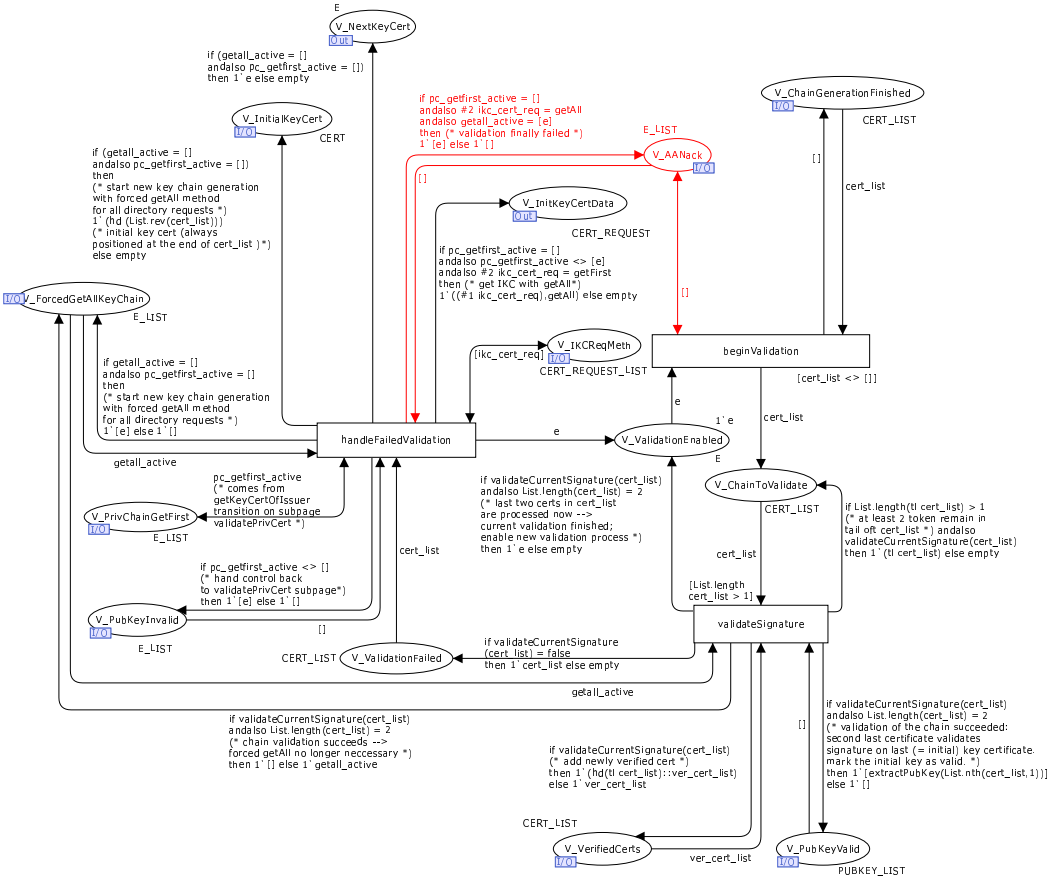


Abbildung 6.32: Subpage *validateKeyChainSignatures*

Subpage *validatePrivCert* initiiert wurde (siehe 6.2.4.4.4) und das verwendete Initialzertifikat dort mit `getFirst` geholt wurde (vermerkt in *V_PrivChainGetFirst*), wird der Schlüssel als ungültig erklärt (Stelle *V_PubKeyInvalid*) und der Datenfluss über diesen *i/o*-Port wieder in die Subpage *validatePrivCert* geleitet.

4. *Abbruch:* Wenn keine gültige Zertifizierungskette erzeugt werden kann, obwohl bereits alle zur Verfügung stehenden Möglichkeiten ausgeschöpft wurden, wird der Authentifizierungs- und Autorisierungsprozess endgültig abgebrochen (*V_AANack*).

6.2.4.4.4 Subpage *validatePrivCert* (Superpage *processAA*)

In der Subpage *validatePrivCert* wird eine Kette hergeleitet, um ein Privilegzertifikat zu validieren. Ist es gültig, wird der A&A-Prozess erfolgreich abgeschlossen und die Entität gilt als autorisiert, das entsprechende Privileg auszuüben.
Die Subpage ist großteils analog zu Subpage *deriveKeyCertChain* aufgebaut, wie in Abbildung 6.33 skizziert. Abb. 6.34 zeigt die vollständige Subpage *validatePrivCert*.
Es wird mit der Prüfung der Statusinformation zum aktuellen Privilegzertifikat begonnen (Transitionen *requestSi*, *checkRequestedSis*), wobei das erste aktuelle Zertifikat das

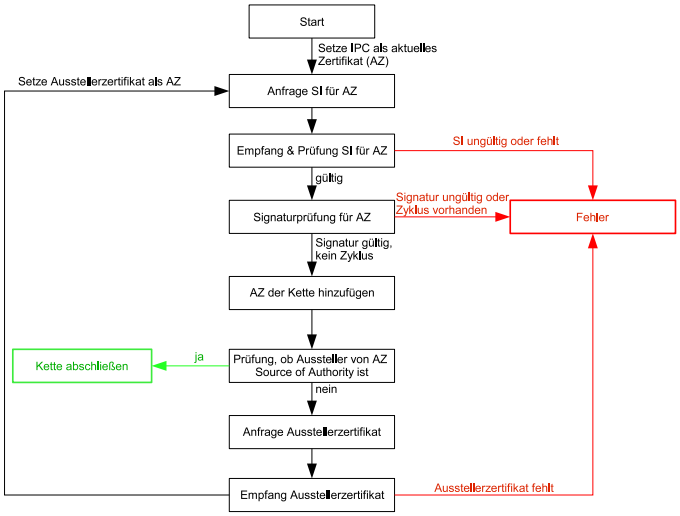


Abbildung 6.33: Skizze der Datenflüsse auf Subpage *validatePrivCert*

Initialzertifikat ist. Danach wird zunächst die Signatur geprüft. Ist noch kein verifiziertes Zertifikat (d.h. eines, dessen Korrektheit bereits geprüft und dessen zertifizierter öffentlicher Schlüssel daher als authentisch anerkannt ist (vgl. 6.2.4.4.3.1)) bekannt, muss ein Schlüsselzertifikat über *P2P-ZuSI* angefordert werden (Transition *getKeyCertOfIssuer*). Ist aus dem Verzeichnis ein solches geliefert worden (ggf. nach wiederholter Anfrage mit `getAll`), wird die Erzeugung und Validierung einer Zertifizierungskette für dieses Schlüsselzertifikat initiiert (Transition *initKeyChainGeneration* mit Ausgabestelle *V_InitialKeyCert*, welche einen neuen Durchlauf der Page *validateKeyCert* anstößt).
In der Transition *checkSignature* wird dann geprüft, ob ein authentischer öffentlicher Schlüssel ermittelt werden konnte (entweder mittels eines verifizierten Zertifikats oder durch Ausführung von *validateKeyCert*). Wenn ja, wird weiterhin die Signatur auf dem Zertifikat validiert. Ist auch diese Prüfung erfolgreich und liegt kein Zyklus vor (Methode `containsCert` prüft dies), fügt V das Zertifikat der Zertifizierungskette hinzu (*V_ToAdd*, *addCertToChain*). Ist die Signatur *nicht* korrekt, wird entweder zur Anfrage nach dem Schlüsselzertifikat zurückgesprungen (wenn dies zuvor mit `getFirst` angefragt) oder eine `[e]`-Liste an *V_Failure* übertragen.
War hingegen die Herleitung eines authentischen öffentlichen Schlüssels nicht möglich, wird entweder zurück zu *getKeyCertOfIssuer* gesprungen, um diesmal mit `getAll` nach einem weiteren initialen Schlüsselzertifikat für den Aussteller zu suchen (Bogen zur Ausgabestelle *V_SiValid-StartSignValidation*), oder aber zur Anfrage nach dem Ausstellerzertifikat der vorigen Iteration (Bogen zu *V_IssuerCertNeeded*). Stehen diese beiden Optionen nicht mehr zur Verfügung, wird auch in diesem Fall ein Fehler gemeldet.
Ist die Signatur erfolgreich validiert, erfolgt die Prüfung, ob der Aussteller des Zertifikats eine Source of Authority ist. Wenn ja, wird die Kette mit dem SoA-Zertifikat (ausgestellt von V) abgeschlossen (*V_ChainEnds*, *completeChainAndStore*). Wenn nein, muss ein Privilegzertifikat für den Aussteller beschafft werden. Dies ist analog zu *deriveKeyCertChain* in den Transitionen *getIssuerCertificate* und *receiveIssuerCertificate* modelliert.

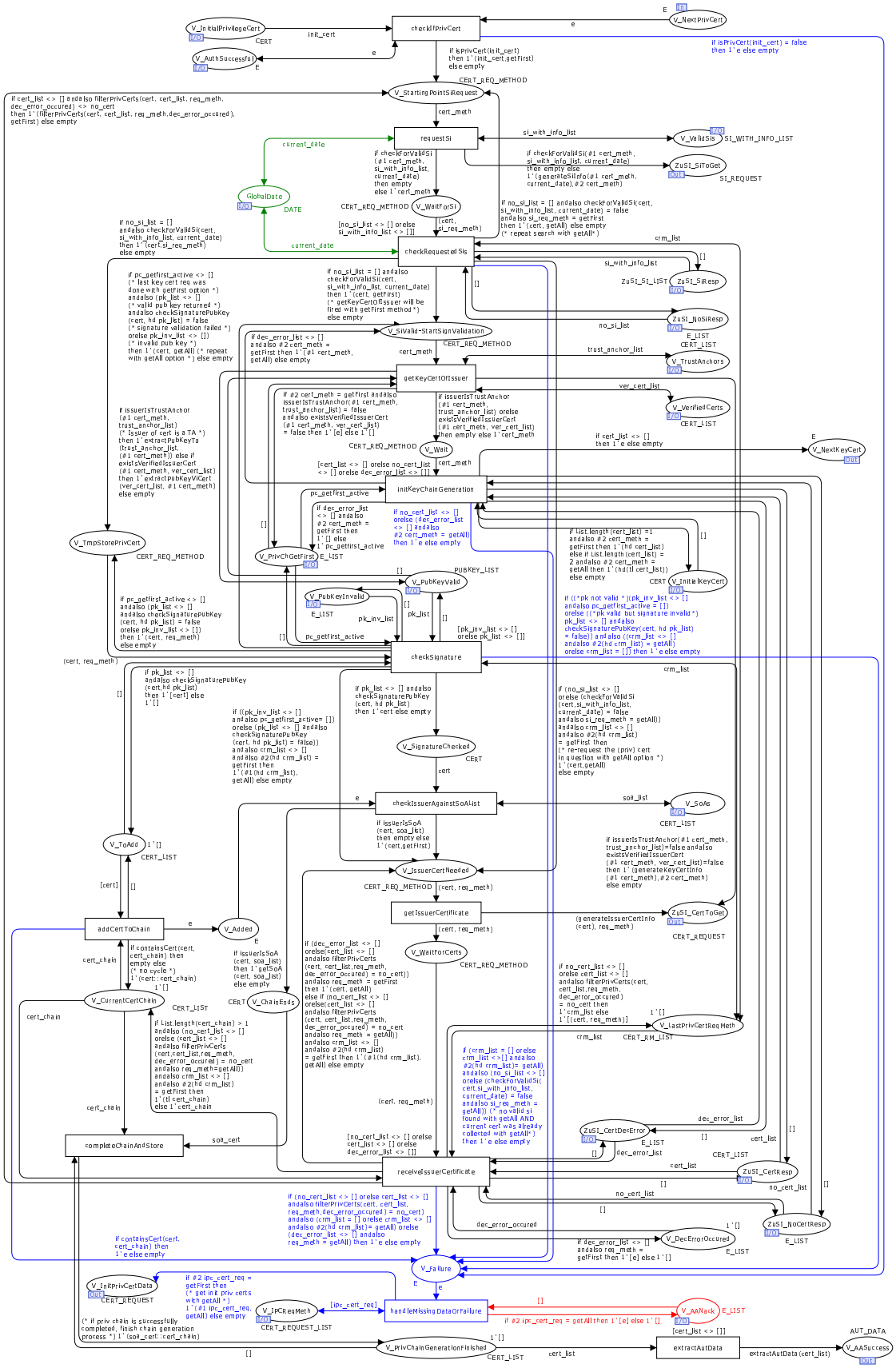


Abbildung 6.34: Subpage *validatePrivCert*

Wird in *receiveIssuerCertificate* ein Zertifikat empfangen (bzw. zwei Zertifikate bei Anfrage mit `getAll`, wovon nur noch das zweite verarbeitet wird), fährt der Prüfer nur dann oben mit *requestSi* für das Ergebniszertifikat fort, wenn dessen Delegationsstufe größer als die des zuletzt der Kette hinzugefügten Zertifikats ist.
Nach dem erfolgreichem Aufbau einer Kette wird zuletzt die Transition *extractAutData* gefeuert. Diese vermerkt in der Stelle *V_AASuccess* den verifizierten Privilegtyp und den Inhaber dieses Privilegs.

6.3 Fazit

Das in diesem Kapitel vorgestellte, vollständig unter Verwendung von *CPN Tools* simulierbare Grundmodell wurde bereits im Hinblick auf die Eignung für eine zustandsraumbasierte Sicherheitsanalyse der AAI mit verteiltem Verzeichnis entwickelt. Dabei kam insbesondere eine Vielzahl an Methoden zur Einschränkung des Zustandsraums und der Verringerung der Modellkomplexität allgemein zum Einsatz. Das *P2P-ZuSI*-Regelwerk wie in Kapitel 5 definiert ist dabei vollständig abgebildet.

Den umfangreichsten Teil des Modells stellen die AAI-Subpages *getInitialCertificates*, *deriveKeyCertChain*, *validateKeyChainSignatures* und *validatePrivCert* dar. Der Grund dafür ist der Modellierungsfokus: Das Modell soll nicht nur das entwickelte System in einer simulierbaren Form abbilden, sondern auch die Analyse von Angriffsszenarien ermöglichen (siehe Kapitel 7).
Angriffe, bei welchen Daten im P2P-Verzeichnis modifiziert werden, wirken sich nur auf die Erzeugung und Validierung von Zertifizierungsketten aus, da nur hier die Anfrage und Verarbeitung von Daten aus dem Verzeichnis erfolgt. Ein mit böswilliger Absicht modifizierter Datensatz kann an vielen Stellen auf *P2P-ZuSI*-Anfragen zurückgeliefert werden.
In der Sicherheitsanalyse muss geprüft werden können, ob ein solcher Datensatz an irgendeiner Stelle im A&A-Prozess vom Prüfer akzeptiert wird, woraus sich die Kompromittierung eines Sicherheitsziels ergeben wird. Daher ist die sehr detaillierte Ausarbeitung der Subpages von *processAA*, in denen Daten aus dem Verzeichnis verwendet werden, von essentieller Bedeutung für die Sicherheitsuntersuchung.

Das präsentierte HCPN beschreibt formal die Kombination einer abstrakten AAI mit *P2P-ZuSI* als Verzeichnisdienst und bildet die Basis für eine modellbasierte Sicherheitsanalyse. Weiterhin kann das Modell mit seinen detailliert ausgearbeiteten Rollen und Methoden als Grundlage für eine spätere Softwareimplementierung dienen (siehe Abschnitt 9.2.2).

Kapitel 7

Sicherheitsanalyse des Systems

Das in Kapitel 6 entwickelte Grundmodell für eine *P2P-ZuSI*-basierte AAI wird nun als Grundlage für die Sicherheitsuntersuchung verwendet. Ziel dieser Analyse ist der Nachweis der beanspruchten Sicherheitseigenschaften von *P2P-ZuSI*.
Dabei werden die konkreten Sicherheitsziele den beispielsweise in den IT-Grundschutzkatalogen des Bundesamts für Sicherheit in der Informationstechnik (BSI) definierten Standard-Kategorien Vertraulichkeit, Integrität und Verfügbarkeit zugeordnet und untersucht.
Dabei ist zu beachten, dass der Betrachtungsfokus auf den Effekten der verteilten Speicherung liegt. Zentrale Frage ist also, ob es Angriffe auf die Sicherheitsziele gibt, welche dadurch entstehen, dass die Daten nicht zentral auf einem vertrauenswürdigen System, sondern dezentral auf nicht notwendigerweise vertrauenswürdigen Peer-to-Peer-Knoten gespeichert sind. Dabei werden die kryptographischen Primitive (Verschlüsselungs- und Signaturmethoden, Hashfunktionen usw.) als sicher vorausgesetzt.

Anhand einer Simulation wird der Brute-Force-Angriff auf die Vertraulichkeit von Zertifikaten und Statusinformationen wie bereits in 5.3.3 beschrieben gezeigt. Die Untersuchung von Angriffen auf die Integrität wird im wesentlichen auf Basis von Zustandsraumbetrachtungen (siehe Abschnitt 6.1.3) des entwickelten HCPN durchgeführt.

7.1 Sicherheitsziele in einer AAI mit verteiltem Peer-to-Peer-Verzeichnis

In der Informationssicherheit werden Sicherheitsziele in drei zentrale Zielkategorien unterschieden (z.B. [BSI06]):

1. *Verfügbarkeit:* Gewährleistung, dass Daten und Dienste jederzeit zugreifbar sind
2. *Vertraulichkeit:* Gewährleistung, dass eine nicht öffentliche Information nur zum Zugriff autorisierten Entitäten zur Verfügung steht
3. *Integrität:* Gewährleistung, dass Daten und Nachrichten unverändert und vollständig zu jedem beliebigen Zeitpunkt eines festgelegten Betrachtungszeitraums vorliegen. In abgeschwächter Form auch die Gewährleistung, dass die Manipulation oder Entfernung von Daten und Nachrichten erfolgreich erkannt werden kann.

Die Sicherheitsanalyse arbeitet daher auf Basis dieser Kategorien, um zu umfassenden Aussagen über die Gesamtsicherheit des Systems zu gelangen.

7.1.1 Verfügbarkeit

Die Verfügbarkeit von Daten wird bereits in Abschnitt 4.3 ausführlich stochastisch analysiert. Die Ergebnisse stellen eine untere Schranke, also die *minimale* Verfügbarkeitswahrscheinlichkeit dar, da für die Untersuchung vorausgesetzt wird, dass Angreifer mit Angriffsziel Verfügbarkeit im Rahmen ihrer Möglichkeiten die maximale Schadenserzeugung anstreben. Alle gezielten Verfügbarkeitsangriffe auf bestimmte Datensätze werden damit von dieser Analyse abgedeckt, die Verfügbarkeitswahrscheinlichkeit für diese einzelnen Datensätze kann nicht unter die in 4.3 berechnete fallen.
Die Verfügbarkeit von Inhalten ist also in ausreichendem Maße gewährleistet und bedarf keiner weiteren Untersuchung.

Abbildung 7.1: Sicherheitsziel Verfügbarkeit

Weiterhin ist die Verfügbarkeit des gesamten Dienstes ein Kriterium für die Sicherheit. Dies bedeutet, dass das Peer-to-Peer-Verzeichnis zu jedem beliebigen Zeitpunkt zugreifbar sein muss. Durch die inhärenten Eigenschaften von strukturierten P2P-Systemen (siehe Abschnitt 2.3) ist eine hohe Ausfallsicherheit gewährleistet: auch wenn Knoten das Netz verlassen, führen die übrigen Teilnehmer den Dienst weiterhin aus. Auch hinsichtlich des Dienstangebots ist also die Verfügbarkeit gesichert.

Für die Analyse der weiteren Sicherheitsziele wird daher die Verfügbarkeit aller im Verzeichnis abgelegten Datensätze zum Betrachtungszeitpunkt als gegeben vorausgesetzt.

7.1.2 Vertraulichkeit

Sicherheitsziele hinsichtlich der Vertraulichkeit werden in zwei Kategorien unterteilt: Unbeobachtbarkeit von Kommunikationsbeziehungen und Datenvertraulichkeit.
Unter Unbeobachtbarkeit gegenüber Dritten ist zu verstehen, dass Kommunikationsbeziehungen zwischen AAI-Entitäten vertraulich bleiben[1]. Es soll also insbesondere nicht erkennbar sein, welche End-Entität sich gegenüber welchem Prüfer autorisiert.
Kommunikationsverbindungen zwischen End-Entität und Issuing Authority dienen immer der Erzeugung eines Zertifikats. Wird ein solches von einem Angreifer dechiffriert,

[1]Kommunikationsbeziehungen zwischen Peer-to-Peer-Knoten sind hingegen nicht als vertraulich zu schützen.

Abbildung 7.2: Sicherheitsziel Vertraulichkeit

kann er die Verbindung zwischen EE und IA aufdecken (d.h. er erfährt, welche IA für welche EE Zertifikate signiert), was somit vollständig auf den Fall der Vertraulichkeitsverletzung von Inhalten zurückführt. Die Verbindungen zwischen Issuing Authority und Status Authority sind nicht vertraulich zu behandeln, da die Autorisierung der SA zur Ausstellung von Statusinformationen kein Geheimnis darstellt.
Der einzige interessante Fall ist also, dass ein Angreifer anhand von Verzeichnisanfragen eines Prüfers versucht zu rekonstruieren, welche End-Entität mit V in Kontakt steht. Daraus kann er versuchen rückzuschließen, für welche Ressource(n) die EE autorisiert werden will.
Ein Knoten (oder eine Gruppe von Knoten) im P2P-System, welcher *nicht* im Rahmen einer Verschwörung mit dem betreffenden Prüfer kollaboriert, kann aber höchstens einen Teil der Lookup-Nachrichten und Speicheraufforderungen untersuchen, die vom Knoten (in Rolle *RNode*) des Prüfers versandt werden. Selbst wenn er daraus ermitteln könnte, welche Daten hier angefragt werden, kann er nicht mit Sicherheit herausfinden, mit welcher End-Entität der Prüfer in Kontakt steht, da er niemals sicher sein kann, ob ein angefragter Datensatz sich auf ein Initialzertifikat der End-Entität oder auf ein Zertifikat an beliebiger anderer Position innerhalb einer Zertifizierungskette bezieht.
Da Gleichverteilung der FileIDs angenommen wird, variieren die in verschiedenen Lookups kontaktierten Knoten stark. Daher kann mit hoher Wahrscheinlichkeit selbst eine große Verschwörung von Knoten niemals alle Anfragen eines Peers in der Rolle *RNode* (Knoten eines AAI-Prüfers) abfangen, so dass seitens der Angreifer keine abschließende Sicherheit über die Kommunikationsverbindungen zwischen End-Entitäten und Prüfern erreicht werden kann. Somit ist das Sicherheitsziel Unbeobachtbarkeit gewahrt.

Unter Datenvertraulichkeit ist die Geheimhaltung der im Verzeichnis gespeicherten Datensätze zu verstehen. Hier wird in Statusinformationen und Zertifikate unterschieden.
Statusinformationen als Hashwerte von Zufallszahlen tragen keine per se vertraulichen Informationen in sich. Ist die Zugehörigkeit einer Statusinformation zu einem Zertifikat und einer Periode i von Interesse für einen Angreifer, so wird er versuchen, die FileID mittels eines Brute-Force-Angriffs zu rekonstruieren, indem er die Hashfunktion zur FileID-Berechnung auf alle möglichen Kombinationen von Zertifikat(-Hash) und Periode anwendet und mit der gegebenen FileID des Datensatzes vergleicht.
Es ist aber keine vertrauliche Information, dass ein bestimmtes Zertifikat in einer spezifischen Periode noch gültig ist, im Gegenteil. Die Zuordnung einer Statusinformation zu Zertifikat und Periode durch einen Speicherknoten kann also nur dann negative Aus-

wirkungen haben, wenn daraufhin ein gezielter Angriff hinsichtlich eines anderen Sicherheitsziels aus dem Bereich der Verfügbarkeit (bereits in 7.1.1 und 4.3 abgehandelt) oder Integrität (siehe Abschnitt 7.1.3 und 7.2.2) durchgeführt werden soll.
Zertifikate hingegen können vertrauliche Informationen enthalten, die unabhängig von der Ermöglichung weiterer Angriffe für einen Angreifer von Interesse sein können. Auch hier ist nur ein Brute-Force-Angriff möglich (siehe 5.3.3), um die Kombination von Inhaber und Attributtyp des verschlüsselten Zertifikats zu ermitteln und mit diesem Wissen die Inhalte dechiffrieren zu können.
Beide Angriffe auf die Vertraulichkeit der Inhalte werden in Abschnitt 7.2.1 anhand des Modells illustriert und diskutiert.

7.1.3 Integrität

Ein Teilziel hinsichtlich der Integrität ist die Zurechenbarkeit von transferierten Nachrichten zu einem Sender. Gängigste Methode hierfür ist die digitale Signatur der kommunizierten Daten durch den Absender. Die Zurechenbarkeit ergibt sich aus der Integrität des signierten Datensatzes und der Integrität und Authentizität der Absenderidentität.
Zurechenbarkeit auf P2P-Ebene ist hier kein angestrebtes Sicherheitsziel. Dies wäre nur dann sinnvoll, wenn man böswilliges von protokollgemäßem Verhalten von Knoten unterscheiden und Angriffe einem Sender zuordnen könnte. Dies ist aber nicht möglich (siehe Abschnitt 5.4).
Zurechenbarkeit auf AAI-Ebene hingegen beschränkt sich auf die Kommunikationsprotokolle zwischen AAI-Entitäten und ist daher völlig unabhängig von der Art der Datenverwaltung. Da Sicherheitsfragen der reinen AAI-Ebene nicht Gegenstand der Sicherheitsanalyse sind, ist auch diese Art der Zurechenbarkeit von der Untersuchung auszuschließen. Zurechenbarkeit ist also in der Sicherheitsanalyse nicht zu behandeln und daher in Abbildung 7.3 grau markiert.

Abbildung 7.3: Sicherheitsziel Integrität

Die Integritätssicherung der Daten spielt eine zentrale Rolle in der AAI mit *P2P-ZuSI* als Verzeichnisdienst, da die Speicherknoten und die Publikation von Datensätzen nicht kontrolliert werden können. So ist es nicht möglich zu garantieren, dass einmal publizierte Daten dauerhaft unverändert und vollständig bleiben. Vielmehr ist damit zu rechnen, dass böswillige Knoten durch Manipulation gespeicherter Datensätze oder Publikation falscher Datensätze versuchen, Vorteile zu erlangen oder den Dienst zu stören.

Unter Integritätssicherung der gespeicherten Daten ist daher zu verstehen, dass Integritätsverletzungen sicher aufgedeckt werden und keinen Schaden anrichten können.
Es ist hier wiederum nach Datentyp zu unterscheiden, da für Zertifikate und Statusinformationen unterschiedliche Techniken zur Integritätssicherung zum Einsatz kommen.
In Abschnitt 7.2.2 werden konkrete Szenarien für Integritätsangriffe entwickelt, in das hierarchische gefärbte Petrinetz aus Kapitel 6 integriert und anhand einer Zustandsraumanalyse ermittelt, ob ein Angreifer aus den Eigenschaften des verteilten Peer-to-Peer-Verzeichnisses eine Möglichkeit herleiten kann, Daten zu modifizieren und zu seinen Gunsten zu nutzen.

7.1.4 Allgemeines Angreifermodell für *P2P-ZuSI*

Ein Angreifer ist Teilnehmer des Gesamtsystems. Er kennt damit die Methoden und Regelwerke auf allen drei Schichten des Systems und kann in allen Rollen agieren.
Es wird starke Kryptographie vorausgesetzt, d.h. ein Angreifer kann kryptographische Methoden nicht brechen. Insbesondere kann er verschlüsselte Daten nicht ohne den korrekten Schlüssel dechiffrieren. Auch die verwendeten Hashfunktionen werden als sichere, nicht kompromittierbare Einwegfunktionen angenommen. Es gilt also auch für diesen Angreifer die Annahme *A2* aus Abschnitt 4.2.2.1.

7.1.5 Gegenüberstellung der verwendeten Analysemethoden

Es sind also lediglich noch die Sicherheitsziele Vertraulichkeit und Integrität hinsichtlich der im verteilten Verzeichnis gespeicherten Daten zu untersuchen. Tabelle 7.1.5 stellt kurz die verwendeten Methoden gegenüber.

Sicherheitsziel	**Methode**
Datenverfügbarkeit	Quantitative stochastische Analyse in Abschnitt 4.3
Datenvertraulichkeit	Beispielsimulation im HCPN mit expliziter Angreifermodellierung, da Angriff bereits bekannt
Datenintegrität	Qualitative Zustandsraumanalyse des HCPN mit impliziter Angreifermodellierung

Tabelle 7.1: Eingesetzte Analysemethoden

Die Definition impliziter und expliziter Angreifermodellierung im HCPN-Modell sind Gegenstand der Abschnitte 7.2.1 und 7.2.2.

7.2 Analyse anhand des Modells

7.2.1 Szenario 1: Angriffe auf die Vertraulichkeit

Im Folgenden wird ein konkreter Angreifer in das HCPN integriert, welcher in der Rolle *SNode* agiert. Dies bedeutet, dass er Datensätze, welche er speichert, hinsichtlich der Vertraulichkeit zu kompromittieren sucht. Während in der später folgenden Integritätsanalyse (Abschnitt 7.2.2) ermittelt wird, *ob* bestimmte Angriffe erfolgreich sind, wird hier gezeigt, *dass* ein bestimmter Angriff zu einer Kompromittierung der Vertraulichkeit führt. Darin begründet sich die unterschiedliche Vorgehensweise: hier wird eine Beispielsimulation des bereits in 7.1.2 und 5.3.3 skizzierten Angriffs erzeugt, in 7.2.2 anhand von Zustandsraumuntersuchungen ermittelt, ob erfolgreiche Angriffe existieren.

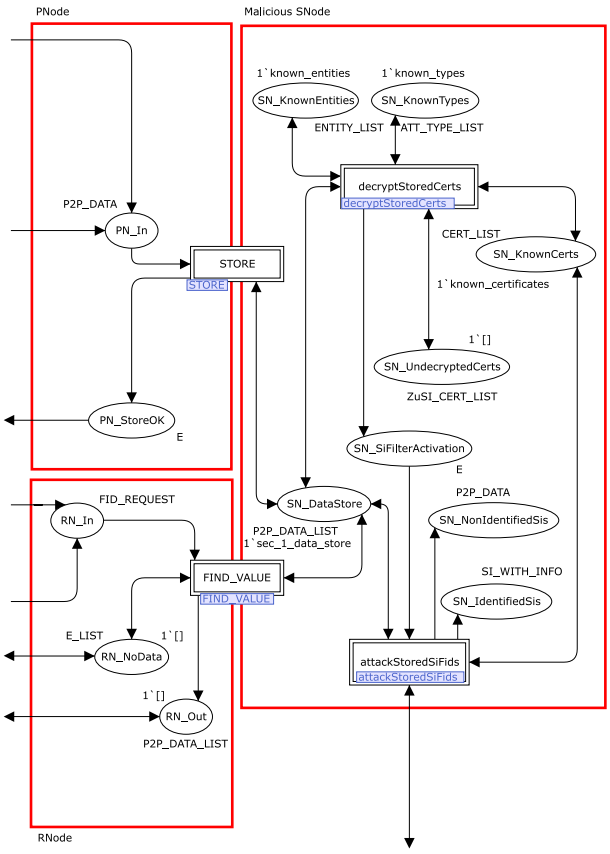


Abbildung 7.4: Peer-to-Peer-Schicht im Top-Level-Modell bei Vertraulichkeitsangriff

Der Angreifer mit Angriffsziel Vertraulichkeit verfügt über Kenntnisse hinsichtlich der AAI, d.h. er kennt eine Teilmenge der existierenden Entitäten, Zertifikate und Attributtypen. Er hat nicht zeitgebundenen Zugriff auf die Zieldaten seines Angriffs, d.h. sie stehen ihm dauerhaft auf seinem eigenen Computersystem zur Verfügung. Er kann zudem zwischen Statusinformationen und Zertifikaten aufgrund der verschiedenen Größenordnungen ihrer Dateilänge unterscheiden.

Die Entität *SNode* wird auf der Main Page um die beiden Substitutionstransitionen *decryptStoredCerts*, *attackStoredSiFids* und sieben Stellen erweitert, um den Angreifer in Form eines böswilligen Speicherknotens abzubilden.
In den Stellen *SN_KnownEntities* und *SN_KnownTypes* werden die dem Angreifer bekannten Entitäten und Typen gespeichert, mit denen er in Subpage *decryptStoredCerts* den Angriff auf die Vertraulichkeit der Zertifikate durchführt. Dabei sind die Typen nach absteigender Wahrscheinlichkeit des Auftretens sortiert, d.h. der erste Typ in der Liste vom Colour Set `ATT_TYPE_LIST` ist derjenige Attributtyp, welcher erwartungsgemäß unter den Zertifikaten am häufigsten vorkommt.
In der Ausgabestelle *SN_KnownCerts* legt der Angreifer die erfolgreich entschlüsselten Zertifikate ab, *SN_UndecryptedCerts* nimmt diejenigen auf, die nicht dechiffriert werden konnten.
Die Stelle *SN_KnownCerts* ist weiterhin Eingabestelle von *attackStoredSiFids*. Daher enthält sie schon zu Beginn eine Liste a priori bekannter Zertifikate im Klartext. Damit der Angreifer bei seinem Angriff auf die FileIDs der Statusinformationen auf eine maximale Menge an Klartext-Zertifikaten zurückgreifen kann, wird immer zuerst die Verarbeitung in *decryptStoredCerts* abgeschlossen, bevor *attackStoredSiFids* aktiviert wird (über Transfer eines *e*-Tokens in und von *SN_SiFilterActivation*).
In *SN_IdentifiedSis* speichert der Angreifer die Statusinformationen, deren zugehöriges Zertifikat und die passende Periode er ermitteln konnte, gemeinsam mit diesen Daten. Nicht zuordenbare SI-Datensätze legt er in *SN_NonIdentifiedSis* ab.

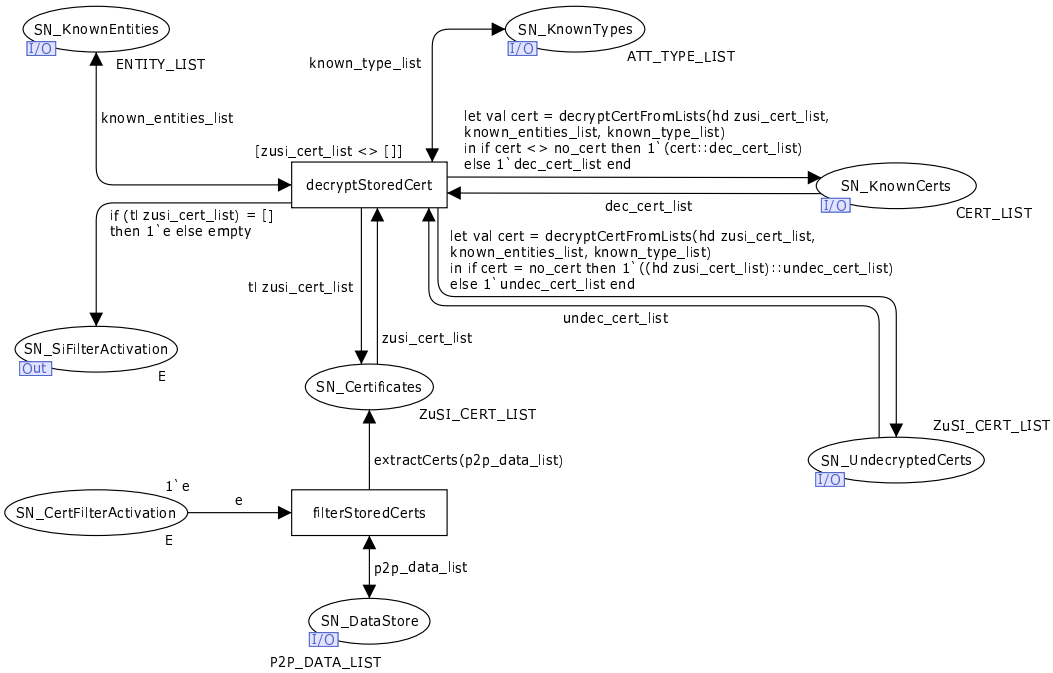


Abbildung 7.5: Subpage *decryptStoredCerts*

Abbildung 7.5 zeigt die Subpage *decryptStoredCerts*. Die folgende Betrachtung fokussiert ausschließlich Angreifer in der Rolle *SNode* und blendet folglich *SNode*-Knoten, die sich regelkonform verhalten, aus. Als Konsequenz beschränkt sich die Belegung der Stelle *SN_DataStore* auf die Datensätze, welche böswilligen Knoten vorliegen. Auf anderen Knoten gespeicherte Daten werden nicht berücksichtigt.
Zunächst werden aus den Daten in *SN_DataStore* die Zertifikate herausgefiltert (Methode `extractCerts`). Anschließend durchläuft jedes chiffrierte Zertifikat die Transition *decryptStoredCert*. Hier kommt die Methode `decryptCertFromLists`[2] zum Einsatz, welche die bekannten Entitäten und Typen miteinander kombiniert, um einen passenden Schlüssel zu erzeugen.

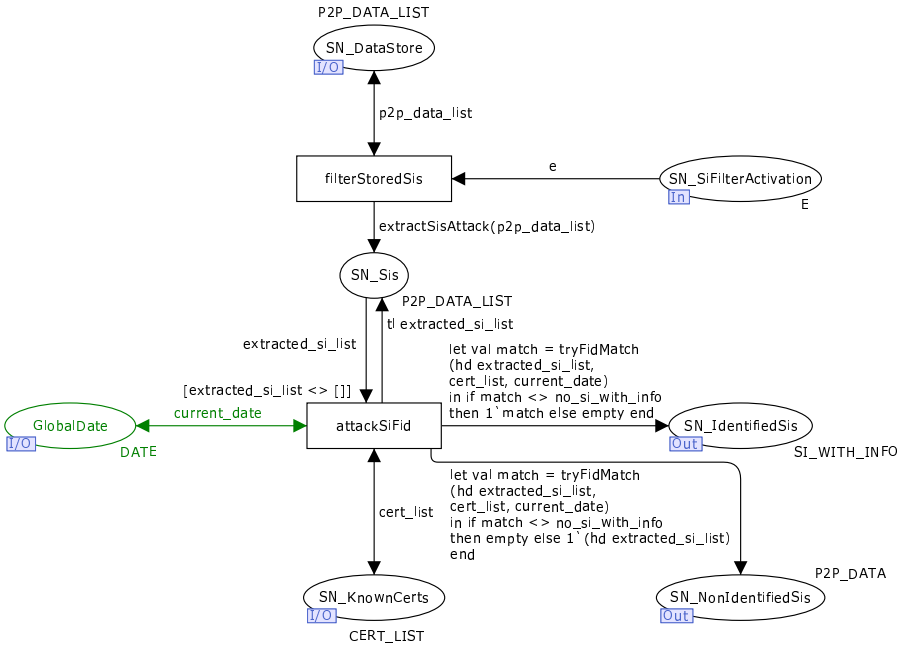


Abbildung 7.6: Subpage *attackStoredSiFids*

Auch in der Subpage *attackStoredSiFids* (Abbildung 7.6) wird zuerst eine Filtermethode angewandt. Hierbei werden aus den gespeicherten Datensätzen in *SN_DataStore* nur die Statusinformationen ausgewählt (Methode `extractSisAttack`).
Dann wird jede einzelne davon in *attackSiFid* verarbeitet. Dabei verwendet der Angreifer der Reihe nach jedes Zertifikat aus der Liste in *SN_KnownCerts*: er berechnet jeweils zunächst den aktuellen Wert i und prüft, ob die FileID fid der Statusinformation $fid = H(H(cert) \dotplus i)$ ist. Ist dies nicht der Fall, werden alle anderen möglichen (kleineren) Werte i für dieses Zertifikat ausprobiert. Ergibt auch dies keinen Erfolg, d.h. kann kein zur tatsächlichen FileID identischer Bezeichner erzeugt werden, wird das nächste Zertifikat der Liste genommen und damit analog verfahren. Dieses Vorgehen wird über die Methode `tryFidMatch` umgesetzt.
Kann zu einer SI die richtige Kombination von Zertifikat und Periode ermittelt werden,

[2] siehe Appendix C.2.1

wird sie gemeinsam mit diesen Informationen in der Ausgabestelle *SN_IdentifiedSis* gespeichert, sonst in *SN_NonIdentifiedSis*.

Das um den Angreifer erweiterte Modell ermöglicht die Darstellung eines erfolgreichen Angriffs auf die Vertraulichkeit. Belegt man die entsprechenden Stellen mit exemplarischen Listen von bekannten Entitäten, Typen und Zertifikaten, so ergibt sich eine Beispielsimulation[3]. Das Initial Marking wird dabei so gewählt, dass ausschließlich die Transitionen der Angreifer-Rolle aktiviert sind, da andere Rollen nicht zum Erfolg des Angriffs beitragen oder diesen verhindern können. Modell und Beispielbelegung sind auf der beigefügten CD als simulierbare *CPN-Tools*-Datei enthalten.

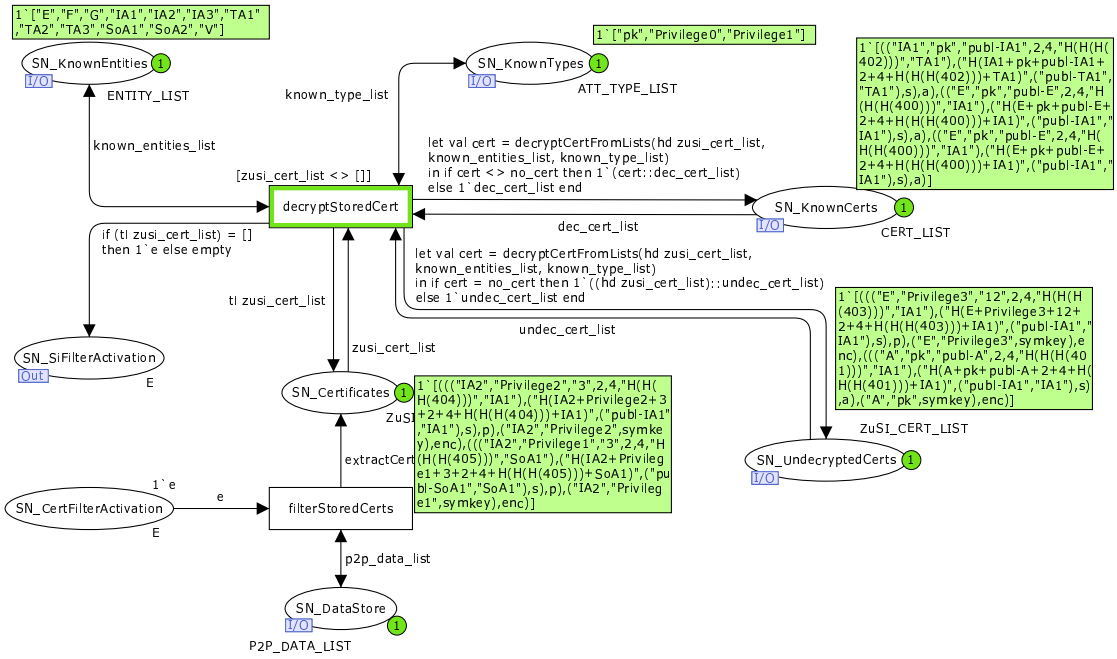


Abbildung 7.7: Subpage *decryptStoredCerts* im Simulationsschritt 5

Abbildung 7.7 zeigt den Zustand auf der Subpage *decryptStoredCerts* im 5. Simulationsschritt[4]. Es ist zu sehen, dass nicht alle Zertifikate erfolgreich entschlüsselt werden können, da der Angreifer nicht über alle notwendigen Vorkenntnisse verfügt.
Analog zeigt Abbildung 7.8 den Zustand der Subpage *attackStoredSiFids* im 10. Simulationsschritt. Es wurden bereits alle Statusinformationen abgearbeitet: zu einer davon konnte die Zuordnung zu Zertifikat und Periode ermittelt werden.

Die gezeigte Beispielsimulation illustriert das genaue Vorgehen des Angreifers bei der Kompromittierung der Vertraulichkeit seiner gespeicherten Datensätze. Da jeder Knoten die gespeicherten Daten zu seiner unbeschränkten Verfügung hat, besteht keine Möglich-

[3]Initial Marking siehe Appendix C.2.2
[4]Der grüne Rahmen um *decryptStoredCert* gibt an, dass die Transition noch aktiviert ist.

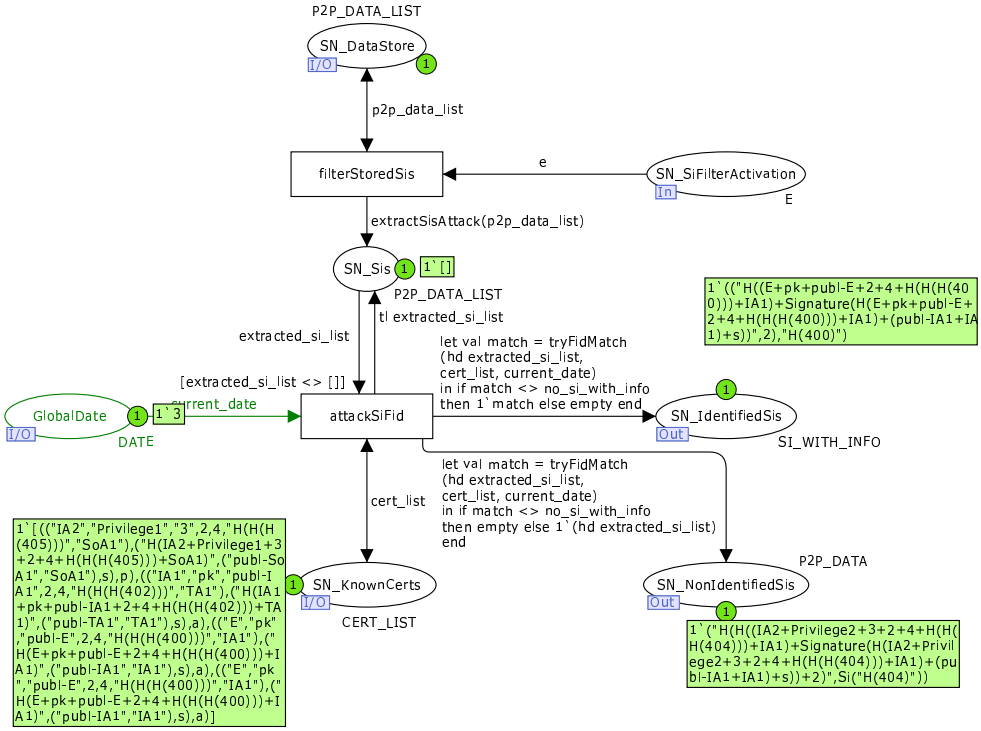


Abbildung 7.8: Subpage *attackStoredSiFids* im Simulationsschritt 10

keit, den Angriff zu verhindern. Die Verschlüsselung von Zertifikaten erhöht lediglich den für den Erfolg notwendigen Aufwand. Diese Tatsache hat Auswirkungen auf die empfohlenen Einsatzgebiete für *P2P-ZuSI* (siehe Kapitel 8).

7.2.2 Szenario 2: Angriffe auf die Integrität

Da ein böswilliger Speicherknoten also mit dem entsprechenden Aufwand in der Lage ist, Datensätze zu entschlüsseln und die Klartextdaten zu modifizieren und generell jeder Teilnehmer des P2P-Netzwerks beliebige - also auch modifizierte Daten - publizieren kann, ist die Untersuchung der Auswirkungen von Integritätsangriffen ein zentraler Punkt der Sicherheitsanalyse.
Könnte ein Angreifer Datensätze der AAI so modifizieren, dass ein Prüfer sie im Zuge eines A&A-Verfahrens als korrekt annimmt, so könnte er damit auf AAI-Schicht betrügen. Er könnte beispielsweise ein zurückgerufenes Schlüsselzertifikat mittels einer gefälschten SI als gültig erscheinen lassen und in einem A&A-Verfahren das kompromittierte Schlüsselpaar einer anderen End-Entität nutzen, um sich gegenüber einem Prüfer fälschlich als diese auszugeben.

Um zu zeigen, dass solche Angriffsmöglichkeiten aufgrund der inhärenten Eigenschaften der verwendeten Datentypen nicht auftreten können, werden nun Szenarien vorgestellt, welche alle Möglichkeiten für Integritätsangriffe abdecken. Diese werden in das Modell aus Kapitel 6 eingebunden und mittels Zustandsraumanalysen ausgewertet.

Abbildung 7.9: Angriffsziele gegen das Sicherheitsziel Integrität

Die Angriffe werden danach kategorisiert, welcher Datentyp in erster Linie kompromittiert wird, d.h. danach, ob sie primär auf die Manipulation von Statusinformationen oder von Zertifikaten ausgerichtet sind, und nach dem Ziel, das mit diesem Angriff erreicht werden soll. Dabei sind die verschiedenen Angreifer disjunkt, d.h. je Szenario wird nur eine konkrete Angriffsmöglichkeit ausgeschöpft.
Angriffe auf Statusinformationen werden isoliert untersucht, d.h. in diesen Szenarien werden *ausschließlich* Manipulationen auf diese Art des Datentyps betrachtet. Bei Angriffen auf Zertifikate hingegen kann der Angreifer auch manipulierte Statusinformationen nutzen. Ein solches Szenario kann also neben der Manipulation eines Zertifikats auch die Manipulation einer zugehörigen SI beinhalten.
Es ist einsichtig, dass Angriffe auf die konkreten Datensätze die Grundlage für komplexere Angriffe sein können. Kann ein Angreifer erfolgreich Manipulationen an Datensätzen vornehmen, so erwirbt er damit auch die Fähigkeit, die Protokolle und Interaktionen von AAI-Entitäten zu untergraben und erlangt damit eine Vielzahl an Betrugsmöglichkeiten. Kann er hingegen die grundlegenden Datensätze der AAI, die im verteilten Verzeichnis vorgehalten werden, *nicht* erfolgreich angreifen, so bestehen auch keine weiter reichenden Möglichkeiten für den Erfolg komplexerer Angriffe. Es gilt außerdem: hat er in keinem Fall Erfolg bei der Manipulation eines einzelnen Datensatzes eines Typs, so auch nicht bei Manipulation einer Zusammenstellung mehrerer solcher Datensätze.

Rahmenbedingungen der Angreifermodellierung
Zur Modellierung eines Angreifers in einem HCPN existieren zwei grundlegende Herangehensweisen:

1. *Explizite Modellierung des Angreifers als Rolle:* Wie in 7.2.1 wird der Angreifer sichtbar in das Netz eingebaut. Seine möglichen Aktivitäten und zur Verfügung stehenden Daten werden in Form von Transitionen und Stellen abgebildet. Durch Feuern seiner aktiven Transitionen in beliebiger oder festgelegter Reihenfolge erzeugt er Daten, welche er dann an reguläre Entitäten überträgt. Es existieren dazu zwei Subvarianten:

   (a) *Modellierung aller denkbaren Angreifer in einem einzigen HCPN:* Ein universeller Angreifer wird in das Modell eingefügt und mit umfangreichen Daten

belegt. Im Zustandsraum sind dann alle möglichen Angriffe und ihre Ergebnisse enthalten, d.h. es kann durch Untersuchung des Zustandsraums ermittelt werden, ob der Angreifer mit seinen Fähigkeiten und Daten ein oder mehrere Sicherheitsziele auf irgendeine Weise verletzen kann.
Diese Form der Modellierung wird beispielsweise für die CPN-gestützte Analyse kryptographischer Protokolle angewandt [Dre05] [DTM95]. Sie führt aber zu komplexen Modellen und ist generell nur dann einsetzbar, wenn keine gegenläufigen Sicherheitsziele existieren.
Für den hier vorliegenden Anwendungsfall ist sie nicht geeignet, da unterschiedliche Szenarien vorliegen, bei denen die in der Zustandsraumanalyse gesuchten unzulässigen Zustände widersprüchlich sind: so ist in manchen Einsatzszenarien jeder Zustand unzulässig, in welchem das A&A-Verfahren erfolgreich durchlaufen wurde, in anderen jeder Zustand mit abgebrochenem A&A-Verfahren.

(b) *Modellierung separater Angriffsszenarien in verschiedenen HCPN:* Für verschiedene Angrifsszenarien werden explizite Angreifer in das Grundmodell integriert und mit entsprechenden Daten in Form von Marken ausgestattet.
Im vorliegenden Anwendungsfall gibt es aber diverse Möglichkeiten für die Modifikation und Bereitstellung bestimmter Daten durch einen böswilligen Systemteilnehmer. So kann er beispielsweise eine Statusinformation verändern, wenn er in der Rolle eines Speicherknotens agiert und diese SI speichert, oder indem er unter Verwendung der *P2P-ZuSI*-Regeln die gesuchte Statusinformation anfragt, modifiziert und dann mit der Methode *putSi* neu publiziert.
In beiden Fällen ist der Effekt der gleiche: es existieren parallel zwei Versionen einer Statusinformation im Verzeichnis, d.h. in der Stelle *SN_DataStore* des Modells. Selbst für dieses *eine* Angriffsszenario „Modifikation einer Statusinformation“ wären so also mehrere explizite Angreifer möglich. Somit ergibt sich eine hohe Modellkomplexität zur Abdeckung aller dieser Varianten.

2. *Implizite Modellierung des erfolgten Angriffs als Initial Marking:* Der Angreifer findet keine sichtbare Entsprechung im Netz. Stattdessen wird ermittelt, welche Daten ein Angreifer im entsprechenden Szenario erzeugen kann. Diese sind identisch mit den Marken, welche ein explizit modellierter Angreifer durch Ausführung seiner Transitionen erzeugen könnte.
Die so ermittelten Marken werden dann in das Initial Marking des Modells aufgenommen. Sie repräsentieren einen bereits erfolgten Angriff. Der Schwerpunkt der Analyse liegt dann auf der Frage, ob dieser Angriff zum Erfolg führen kann.
Für die modellbasierte Analyse hinsichtlich des Sicherheitsziels *Integrität* wird dieser Modellierungsansatz gewählt. Da *alle* möglichen Angreifer die von ihnen modifizierten Datensätze der Stelle *SN_DataStore* hinzufügen (durch Bearbeitung ihrer gespeicherten Daten oder durch Publikation veränderter Datensätze), ist das Initial Marking, welches die Situation des verteilten Verzeichnisses nach einem erfolgten Angriff repräsentiert, leicht zu erzeugen: es beschränkt sich auf die Initialbelegung von *SN_DataStore*.
In der Analyse ist nur von Bedeutung, ob diese vom Angreifer modifizierten Datensätze zu einem erfolgreichen Angriff führen können. Dies wird ermittelt, in-

dem die Subpage *processAA* durchlaufen wird, d.h. die im Verzeichnis vorhandenen gefälschten Daten werden in einem konkreten Authentifizierungs- und Autorisierungsprozess verwendet.
Verschiedene Szenarien repräsentieren verschiedene Manipulationsmöglichkeiten der Datensätze. Es genügt (mittels vollständiger Enumeration der möglichen Zustände) zu zeigen, dass ein im Verzeichnis vorhandener modifizierter Datensatz *nie* zu einer Kompromittierung des Authentifizierungs- und Autorisierungsverfahrens führt. Wie der Angreifer konkret zu dieser Manipulation gelangt, also mit welchen Methoden und mit Hilfe welcher Daten er diese - unter Einhaltung seiner im Angreifermodell zugewiesenen Fähigkeiten und Kenntnisse - durchführt, ist hier nicht von Belang, sondern würde erst bei Ermittlung eines erfolgreichen Angriffs interessant.

Hinsichtlich der Analyseergebnisse unterscheiden sich die beiden Modellierungsvarianten nicht: die in der zweiten Variante als gegeben vorausgesetzten Daten sind eben die, welche ein explizit modellierter Angreifer mit äquivalenten Daten und Fähigkeiten produzieren würde. Der Aufwand für die Zustandsraumberechnung ist daher im zweiten Fall geringer.

Es werden im Folgenden Teilszenarien für Angriffe erzeugt, die gemeinsam die gesamten Fähigkeiten und Ziele eines universellen Angreifers auf das Sicherheitsziel Integrität abdecken. Diese Aufteilung muss lückenlos sein, um keine Angriffsmöglichkeiten auszulassen. Dies ist durch die Wahl der Kategorien, nach denen die Aufteilung erfolgt, gewährleistet: zunächst wird nach betroffenem Datentyp (Zertifikat oder Statusinformation) unterschieden, danach werden alle potentiellen Ziele bzw. Motivationen des Angreifers hinsichtlich der Modifikation dieses Datentyps argumentativ hergeleitet. Die Zustandsraumberechnung und -analyse ist dann effizient möglich, da jedes Szenario nur die Effekte *genau eines* Angriffs auf das A&A-Verfahren abbildet.

Allgemeines Initial Marking
Für jedes ermittelte Angriffsszenario wird dann wie bereits erwähnt eine Initialbelegung mit entsprechend modifizierten Daten in *SN_DataStore* generiert. Ihre Zusammensetzung ist jeweils in den betreffenden Abschnitten erläutert. Die Stellen *EE_Init* und *EE_KeyPair* werden mit je einer Marke belegt wie in Appendix C.3.2 definiert.
Die Stellen *V_SoAs* und *V_TrustAnchors* sind mit allgemein vorgegebenen, für alle Szenarien identischen Listen von SoA- und Trust-Anchor-Zertifikaten des Prüfers belegt. V wird dabei stets als gut angesehen, d.h. er ist selbst kein Angreifer, da sich Integritätsangriffe explizit gegen den Prüfer einer AAI richten: sie haben zum Ziel, durch die Modifikation von Daten Authentifizierung und/oder Autorisierung so zu manipulieren, dass die Ergebnisse verfälscht sind und damit der Prüfer betrogen wird[5].
Alle Stellen der nicht beteiligen AAI-Entitäten SA und IA sind im Initial Marking auf Ebene der Main Page mit leeren Listen initialisiert, also deaktiviert, da die Publikation von Daten (sowohl durch berechtigte Entitäten als auch durch den Angreifer) in der betrachteten Periode als bereits abgeschlossen angenommen wird (`e`-Token in *E_StartAA*). Alle weiteren Stellen sind mit den Standardwerten aus dem in Kapitel 6 vorgestellten

[5]Ein böswilliger Prüfer selbst kann im übrigen auf einfache Art eine End-Entität betrügen, indem er ihre Authentifizierung und Autorisierung prinzipiell ablehnt. Dafür ist aber keine Modifikation der Daten nötig - es handelt sich um einen reinen AAI-Angriff bzw. absichtliches Fehlverhalten einer AAI-Entität.

Grundmodell belegt, d.h. ohne Marke, mit *e*-Marke oder einer leeren Liste.

Alle detaillierten Initial Markings der Angriffsszenarien und spezifischen Methoden finden sich in ML-Notation in Appendix C.

7.2.2.1 Szenario 2.1: Manipulation einer Statusinformation

Ein Angreifer mit Angriffsziel Statusinformation kann nur zwei verschiedene Ziele verfolgen (siehe Abb. 7.10):

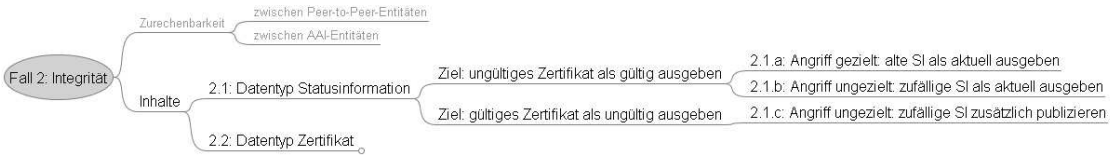


Abbildung 7.10: Sicherheitsziel Integrität: Angriffe auf Statusinformationen

1. *Ungültig gewordenes Zertifikat gültig erscheinen lassen (Betrug):* Ein bestimmtes Zertifikat wurde zurückgerufen. Der Angreifer will es aber für einen A&A-Prozess aktuell erscheinen lassen. Dabei kann er beispielsweise bezwecken, ein nicht mehr gültiges Privilegzertifikat zu verwenden, um sich unberechtigterweise Zugriff auf eine Ressource zu beschaffen.

2. *Gültiges Zertifikat ungültig erscheinen lassen (Denial of Service, DoS):* Der Angreifer bezweckt mit seinem Angriff, gegenüber einem Prüfer ein Zertifikat als zurückgerufen erscheinen zu lassen, und damit zu verhindern, dass ein Authentifizierungs- und Autorisierungsverfahren erfolgreich abgeschlossen werden kann.

Weitere Ziele sind nicht denkbar: eine Statusinformation belegt nur, dass ein Zertifikat gültig ist. Die Gültigkeit oder Ungültigkeit eines Zertifikats zu suggerieren ist also das einzige, was ein Angreifer versuchen kann, wenn er seine Angriffe auf diesen Datentyp aufbaut. Es ist dabei zu beachten, dass er aufgrund der sichergestellten Verfügbarkeit eine existierende Statusinformation nicht aus dem Netz entfernen kann, so dass er zur Vorspiegelung der Ungültigkeit lediglich den Ansatz wählen kann, eine nicht validierbare Statusinformation bereitzustellen.

Im Folgenden werden diese beiden Ziele in konkrete Angriffsszenarien überführt. Anhand der Zustandsraumberechnung wird ermittelt, ob *unzulässige Zustände* erreicht werden können, d.h. solche, in welchen Sicherheitsziele verletzt sind. Diese ergeben sich wie folgt:

1. Will der Angreifer ein ungültiges Zertifikat als gültig erscheinen lassen, sind alle Zustände unzulässig, in denen Authentifizierung und/oder Autorisierung erfolgreich durchgeführt wurde.

2. Will der Angreifer ein gültiges Zertifikat als gültig erscheinen lassen, sind alle Zustände unzulässig, in denen Authentifizierung und/oder Autorisierung erfolglos abgebrochen wurde.

Für das erste Angriffsziel existieren zwei Varianten: der Angreifer kann gezielt oder ungezielt vorgehen. Im ersten Fall nimmt er eine bekannte, aus einer früheren Periode stammende Statusinformation für das Zertifikat und versucht, sie als neu auszugeben, im zweiten Fall wählt er eine zufällige Statusinformation. Dabei wird die Analyse zeigen, dass weder das eine noch das andere Vorgehen erfolgreich sein kann.
Um das zweite Ziel zu erreichen, wird der Angreifer stets absichtlich ungezielt vorgehen, um die Statusinformation und damit das Zertifikat ungültig erscheinen lassen, d.h. er wählt eine zufällige SI, welche er unter der korrekten FileID für Zertifikat und Periode im Verzeichnis ablegt (in welchem zugleich die korrekte Statusinformation für diese Periode liegt, welche er aufgrund der Verfügbarkeitssicherung nicht vernichten kann).

Szenario 2.1.a : Gezielte Neupublikation alter SI mit Betrugsziel
In Szenario 2.1.a versucht der Angreifer, ein zurückgerufenes Zertifikat gültig erscheinen zu lassen, indem er eine alte Statusinformation zu demselben Zertifikat als neu ausgibt. Dafür entfernt er die FileID dieses Datensatzes und ersetzt sie durch eine FileID, welche korrekt gemäß *P2P-ZuSI*-Regelwerk für dasselbe Zertifikat und die *aktuelle* Periode erzeugt wurde.

Das Szenario wird in zwei Subszenarien unterteilt:

- *2.1.a.i: SI betrifft Schlüsselzertifikat:* Es werden zwei Zertifikate in *SN_DataStore* abgelegt: das erste ist von Entität *IA1* für Entität *E* ausgestellt (initiales Schlüsselzertifikat), das zweite von Trust Anchor *TA1* für *IA1*.
  Zum initialen Schlüsselzertifikat ist keine aktuelle Statusinformation vorhanden, da es nicht mehr gültig ist[6]. Zum Zertifikat für Inhaber *IA1* hingegen existiert eine gültige SI.
  Für diesen Fall wird die Autorisierung nicht durchlaufen: die Initialbelegung der Stelle *EE_Init* ist `("E","none")`, d.h. es ist kein Privilegtyp angegeben, der für die Autorisierung herangezogen werden soll. Die Ausführung von *processAA* endet also nach Verarbeitung der Subpage *authenticate* unter Verwendung des Schlüsselpaares von *E*, da kein initiales Privilegzertifikat vorliegt. Unzulässige Zustände sind also solche, in denen *processAA'V_AuthSuccessful* (also die Stelle *V_AuthSuccessful* auf der Page *processAA*) eine *e*-Marke enthält: dies würde bedeuten, dass die Authentifizierung erfolgreich beendet werden konnte, obwohl das initiale Schlüsselzertifikat ungültig ist.

- *2.1.a.ii: SI betrifft Privilegzertifikat:* Das Authentifizierungsverfahren kann erfolgreich durchgeführt werden. Hierfür wird eine minimale Zertifizierungskette angelegt, die nur ein Schlüsselzertifikat für End-Entität *E* ausgestellt von Trust Anchor *TA1* enthält. *E* kann sich dann mit seinem Schlüsselpaar authentifizieren.
  Das intiale Privilegzertifikat für End-Entität *E* ist von der Source of Authority

[6] Prinzipiell ist es für die Untersuchung des Szenarios unerheblich, für welches Zertifikat einer Kette keine korrekte SI existiert.

SoA1 ausgestellt, welche beim Prüfer sowohl als SoA für das entsprechende Privileg *Privilege1* als auch mit ihrem öffentlichen Schlüssel als Trust Anchor bekannt ist. Die Statusinformation für das initiale Privilegzertifikat ist durch einen Angreifer aus einer korrekten SI einer früheren Periode erzeugt und publiziert worden.
Die Stelle *EE_Init* wird mit `("E","Privilege1")` belegt. Unzulässige Zustände sind all jene, in welchen die Stelle *TopLevelModel'V_AASuccessful* eine Marke enthält: dies ist Beweis dafür, dass das A&A-Verfahren erfolgreich durchgeführt werden konnte, obwohl das initiale Privilegzertifikat zurückgerufen ist.

In Tabelle 7.2 werden die Ergebnisse der Zustandsraumanalyse für Szenario 2.1 präsentiert. Da keine unzulässigen Zustände existieren, sind die skizzierten Angriffe nicht erfolgreich.

Szenario 2.1.b : Publikation zufällig zusammengestellter SI mit Betrugsziel

Der Angreifer versucht hier, ein zurückgerufenes Zertifikat gültig erscheinen zu lassen, indem er zufällige Daten als SI mit zum Zertifikat passender FileID publiziert, so dass sie auf Anfrage nach einer aktuellen SI zurückgeliefert werden. Auch hier sind wieder zwei Subszenarien wie in 2.1.a möglich:

- *2.1.b.i: SI betrifft Schlüsselzertifikat*
- *2.1.b.ii: SI betrifft Privilegzertifikat*

Abgesehen von der nun differierenden Statusinformation für das jeweilige Initialzertifikat ist das Initial Marking identisch mit dem aus 2.1.a.i bzw. 2.1.a.ii.
Die Ergebnisse dieses Szenarios sind ebenfalls in Tabelle 7.2 aufgetragen. Dabei ist zu sehen, dass sie mit denen aus 2.1.a identisch sind: in beiden Fällen existieren keine unzulässigen Zustände.

Szenario 2.1.c : Publikation zufällig zusammengestellter SI mit Ziel DoS
Die Analyse von 2.1.b hat bereits gezeigt, dass zufällig zusammengestellte Statusinformationen nicht akzeptiert werden. Ein Angreifer kann nun versuchen, mit einer solchen gefälschten Statusinformation einen Prüfer zu betrügen, so dass dieser ein Zertifikat irrtümlich als ungültig annimmt.
Die Initialbelegung ist analog zu 2.1.b aufgebaut. Der einzige Unterschied liegt darin, dass in jedem der beiden Subszenarien zusätzlich zur zufällig zusammengestellten SI noch eine korrekte Statusinformation vorhanden ist. Dabei liegt die modifizierte Statusinformation in der Liste aus *SN_DataStore vor* der korrekten SI, um den für den Angreifer erfolgversprechendsten Fall zu simulieren, nämlich den, dass bei einer Anfrage mit `getFirst` zuerst die nicht korrekte SI zurückgeliefert wird.

Wie Tabelle 7.2 erwartungsgemäß zeigt, kann aber auch dieser Angriff nicht erfolgreich sein. Der Prüfer stellt fest, dass er mit `getFirst` keine gültige Statusinformation erhält, und wiederholt die Anfrage mit `getAll`.
Dabei erhält er - aufgrund der sichergestellten Verfügbarkeit aller publizierten Datensätze - dann neben dem falschen Datensatz immer auch den korrekten und kann mit der Kettenerzeugung fortfahren. Der Angriff hat also keine Auswirkungen auf das Ergebnis des

A&A-Prozesses, sondern erzeugt lediglich einen geringen Zusatzaufwand für den Prüfer. Die propagierte Anfragepolitik (bei Misserfolg stets Anfrage an alle Knoten aus $\mathcal{N}_b$) verhindert also den Erfolg des Angriffs.

Fall	**Beschreibung**	**Unzulässige Zustände**	**# Zustände**	**# Unzulässige Zustände**
Ziel: Ungültiges Zertifikat als gültig erscheinen lassen (Betrug)				
2.1.a.i	Alte SI für Schlüsselzertifikat wird als aktuell ausgegeben	*processAA'* *V_AuthSuccessful* <> *empty*	31	0
2.1.a.ii	Alte SI für Privilegzertifikat wird als aktuell ausgegeben	*TopLevelModel'* *V_AASuccess* <> *empty*	54	0
2.1.b.i	Zufällige SI für Schlüsselzertifikat wird als aktuell ausgegeben	*processAA'* *V_AuthSuccessful* <> *empty*	31	0
2.1.b.ii	Zufällige SI für Privilegzertifikat wird als aktuell ausgegeben	*TopLevelModel'* *V_AASuccess* <> *empty*	54	0
Ziel: Gültiges Zertifikat als ungültig erscheinen lassen (Denial of Service)				
2.1.c.i	Falsche SI für Schlüsselzertifikat wird publiziert, um Zertifikat als ungültig erscheinen zu lassen	*TopLevelModel'* *V_AANack* *= 1 `[e]*	31	0
2.1.c.ii	Falsche SI für Privilegzertifikat wird publiziert	*TopLevelModel'* *V_AANack* *= 1 `[e]*	52	0

Tabelle 7.2: Ergebnisse der Zustandsraumanalyse - Fall 2.1: Manipulation einer SI

7.2.2.2 Szenario 2.2: Manipulation eines Zertifikats

Auch ein Angreifer, welcher ein Zertifikat manipulieren will, kann damit zwei grundlegende Ziele verfolgen:

1. *Manipulation mit dem Ziel der Störung:* Hierbei zielt der Angreifer darauf ab, den A&A-Prozess durch die Bereitstellung eines manipulierten Zertifikats so zu beeinflussen, er fälschlicherweise abgebrochen wird.

2. *Manipulation mit dem Ziel des Betrugs:* Ein Angreifer mit diesem Ziel verändert Zertifikate, um im Zuge eines A&A-Prozesses aktiv betrügen zu können. So kann er beispielsweise versuchen, ein Privilegzertifikat für eine andere Entität so zu fälschen, dass er selbst (als End-Entität auf AAI-Schicht auftretend) dieses Privileg erfolgreich beanspruchen kann.

Für die Angriffsszenarien wird angenommen, dass jeweils nur ein Zertifikat in einer Zertifizierungskette verfälscht ist. Ein Angreifer könnte zwar mehrere solche Zertifikate in das Verzeichnis einbringen, die gemeinsam eine Zertifizierungskette bilden, aber er kann *nicht* verhindern, dass es in einer von einem Prüfer akzeptierten Kette immer *mindestens* ein Zertifikat geben muss, welches korrekt und vom Angreifer nicht modifizierbar ist: Jede validierbare Kette beginnt nämlich mit einem Trust-Anchor- oder einem SoA-Zertifikat, welches direkt beim Prüfer - und *nicht* im Verzeichnis - vorliegt und von V selbst ausgestellt ist.
Unabhängig davon, wie viele manipulierte Zertifikate also in einer Kette vorhanden sind und wie sich diese positionieren - es kann auch das Initialzertifikat selbst gefälscht sein - gibt es also stets mindestens eine Stelle in der Kette, in welcher ein manipuliertes auf ein korrektes Zertifikat folgt. Dies ist der kritische Punkt in der Validierung: wenn hier ein gefälschtes Zertifikat akzeptiert würde, wäre der Angriff gelungen. Daher genügt es, *einen* solchen Übergang in der Analyse zu berücksichtigen.

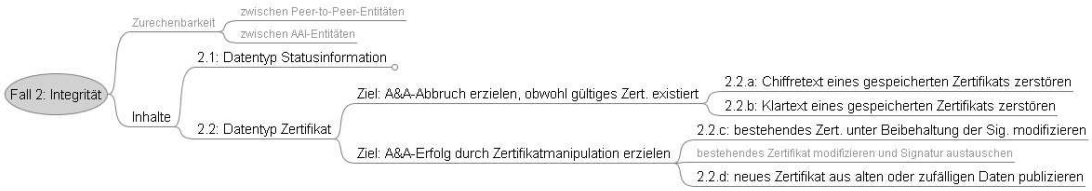


Abbildung 7.11: Sicherheitsziel Integrität: Angriffe auf Zertifikate

Zuerst werden nun die Fälle erläutert, in denen ein Angreifer auf die Störung des A&A-Prozesses abzielt. Darunter fallen die Subszenarien 2.2.a (Zerstörung eines Chiffrats) und 2.2.b (Zerstörung eines Klartextes). Danach folgt die Betrachtung der Modifikationen mit dem Ziel des Betrugs in Subszenarien 2.2.c (Manipulation von bestehenden Zertifikaten) und 2.2.d (Erzeugung eines neuen Zertifikats).

Szenario 2.2.a : Zerstörung eines Chiffrats
Der Angreifer zerstört ein Zertifikat in seiner verschlüsselten Form. Diese beschädigte Version wird in *SN_DataStore* wieder *vor* korrekte Version des entsprechenden Zertifikats geschrieben (günstigster Fall für den Angreifer). Unzulässige Zustände sind also alle, in denen der A&A-Prozess eine Marke in *TopLevelModel'V_AANack* enthält, also der Prozess erfolglos abgebrochen wurde.
Auch hier existieren wieder zwei Subszenarien 2.2.a.i und 2.2.a.ii, bei denen sich die Beschädigung jeweils auf das initiale Schlüssel- bzw. Privilegzertifikat bezieht. Die Stelle *SN_DataStore* ist neben den beschädigten Datensätzen mit korrekten Zertifikaten und Statusinformationen für die Erstellung einfacher Zertifizierungsketten wie in 2.1.a belegt, wobei diesmal für das Initialzertifikat eine gültige SI vorliegt.
Tabelle 7.3 zeigt die Ergebnisse der Zustandsraumanalyse. Es können keine unzulässigen Zustände erreicht werden. Sobald der Prüfer über das Misslingen der Entschlüsselung informiert wurde (Stelle *ZuSI_DecError*), fragt er den Datensatz erneut mit `getAll` an und erhält daraufhin stets auch eine korrekte Antwort.

Szenario 2.2.b : Zerstörung eines Klartextes
Ein Angreifer kann statt des Chiffrats auch die Klartext-Inhalte eines Zertifikats zerstören und es dann regulär verschlüsseln. Dies fällt dann erst in der Validierung des Zertifikats auf, aber noch nicht während der Entschlüsselung auf *P2P-ZuSI*-Ebene. Das Szenario ist wie 2.2.a mit dem einzigen Unterschied aufgebaut, dass nun die Inhalte des Zertifikats unlesbar gemacht wurden. Wie Tabelle 7.3 zeigt, kann der Angreifer auch hier wie erwartet den A&A-Prozess nicht erfolgreich stören: das modifizierte Zertifikat kann zwar entschlüsselt werden, spätestens bei der Signaturprüfung stellt V aber die Modifikation fest und initiiert eine neue Anfrage mit `getAll`.

Szenario 2.2.c : Manipulation von Zertifikatinhalten mit Betrugsziel
In diesem Szenario wirkt der Angreifer auf die Inhalte von bestehenden Zertifikaten ein, um im A&A-Prozess für ihn vorteilhafte Ergebnisse zu erreichen.
Im Subszenario 2.2.c.i greift er direkt auf Ebene des initialen Schlüsselzertifikats an. Dabei manipuliert er ein Zertifikat für eine End-Entität $E2$, indem er den enthaltenen Schlüssel gegen seinen eigenen austauscht. Sein Ziel ist, sich in der Authentifizierung als $E2$ auszugeben. Dies ist aber wie in Tabelle 7.3 gezeigt niemals möglich: zuerst wird zwar die Subpage *deriveKeyCertChain* erfolgreich mit dem gefälschten initialen Schlüsselzertifikat durchlaufen, dann aber in *validateKeyChainSignatures* die Validierung erfolglos abgebrochen, da die Signaturprüfung fehlschlägt. Die Kettenerstellung wird erneut durchlaufen, wobei diesmal das korrekte Zertifikat für $E2$ verarbeitet wird. Diesmal kann die Kette zwar erfolgreich validiert werden, in der Subpage *authenticate* kann aber der Angreifer schließlich nicht authentifiziert werden, da er nicht über das Schlüsselpaar des Angriffsziels $E2$ verfügt.
Im Subszenario 2.2.c.ii bezieht sich der Angriff auf das initiale Privilegzertifikat: der Angreifer modifiziert ein Zertifikat des Inhabers $E2$, indem er stattdessen E als Inhaber einträgt. Damit versucht er, einen erfolgreichen Durchlauf des A&A-Prozesses zu erreichen, in dem End-Entität E - fälschlicherweise - mit Privileg *Privilege1* autorisiert wird. Auch hier wird aber der Betrug in jedem Falle offenbar und das A&A-Verfahren durch den Prüfer abgebrochen.

Prinzipiell sind auch andere Manipulationsangriffe auf Zertifikate möglich, z.B. der Austausch des Ausstellers. Da es aber für den Ausgang der Zertifikatvalidierung unerheblich ist, an welcher Stelle der Inhalt manipuliert ist, sind die Subszenarien 2.2.c.i und 2.2.c.ii als exemplarisch zu verstehen. Würde sich hier ein Angriff ergeben, wären auch andere Manipulationen möglich. Dies ist aber nicht der Fall.

Die Erfolglosigkeit der Angriffe ergibt sich dadurch, dass ein Angreifer, wenn er Zugriff auf den Klartext eines Zertifikats hat, zwar die Inhalte bearbeiten kann, aber nicht in der Lage ist, die Signatur entsprechend zu verändern: hierfür wäre der korrekte Signaturschlüssel des Ausstellers nötig, den ein Angreifer aber nicht kennen kann. Wäre das Schlüsselpaar einer IA kompromittiert und könnte ein Angreifer damit falsche Zertifikate nach Belieben signieren, so wäre dies ein reiner Angriff auf AAI-Ebene und unabhängig vom Aufbau des Verzeichnisses.
Aus diesem Grunde fällt auch die in Abbildung 7.11 grau markierte Angriffsvariante aus, in welcher ein Angreifer ein bestehendes Zertifikat verändert und neu signiert. Auch dies ist ein reiner AAI-Angriff: der Angreifer kann nur ein kompromittiertes (s.o.) oder sein eigenes Schlüsselpaar verwenden. Im zweiten Fall entstünde ein neues Zertifikat, welches vom Angreifer als IA signiert ist - der Erfolg der Zertifizierung gefälschter Daten hängt ausschließlich davon ab, ob der Angreifer innerhalb der AAI als berechtigt zur Ausstellung dieser Zertifikate angesehen wird.

Szenario 2.2.d : Erstellung eines neuen Zertifikats mit Betrugsziel
In diesem Szenario versucht ein Angreifer, ein neues Zertifikat zusammenzustellen und zu publizieren. Es sind mehrere Herangehensweisen denkbar:

- Ein früher gültiges, inzwischen aber ungültiges Zertifikat soll als aktuell ausgegeben werden, indem der Gültigkeitszeitraum geändert wird. Dies führt direkt auf Szenario 2.2.c zurück, in welchem aber die Nichtexistenz erfolgreicher Angriffe gezeigt wurde.
- Ein ganz neues Zertifikat wird aus Teilen anderer Zertifikate zusammengestellt, wobei eine „fremde“ Signatur wiederverwendet wird. Auch dieser Angriff ist äquivalent zu Szenario 2.2.c.
- Ein Zertifikat wird zusammengestellt und selbst signiert: Dies ist ein reiner AAI-Angriff (s.o.) und wird daher nicht betrachtet.
- Ein Zertifikat wird mit einer zufälligen Signatur erzeugt: Das initiale Schlüssel- bzw. Privilegzertifikat wird vom Angreifer selbst zusammengestellt, mit einer zufälligen Signatur versehen und publiziert. Es ist unmittelbar einsichtig, dass auch dieser Angriff unmöglich erfolgreich sein kann. Dies bestätigt auch die - der Vollständigkeit halber durchgeführte - Untersuchung des Zustandsraumes für die Subszenarien 2.2.d.i (initiales Schlüsselzertifikat vom Angreifer erstellt) und 2.2.d.ii (initiales Privilegzertifikat vom Angreifer erzeugt).

Fall	**Beschreibung**	**Unzulässige Zustände**	**# Zustände**	**# Unzulässige Zustände**
Ziel: Denial of Service durch Zerstörung von Daten				
2.2.a.i	Zerstörung des Chiffrats eines Schlüsselzertifikats	*TopLevelModel' V_AANack = 1`[e]*	48	0
2.2.a.ii	Zerstörung des Chiffrats eines Privilegzertifikats	*TopLevelModel' V_AANack = 1`[e]*	70	0
2.2.b.i	Zerstörung des Klartextes eines Schlüsselzertifikats mit folgender (korrekter) Verschlüsselung	*TopLevelModel' V_AANack= 1`[e]*	48	0
2.2.b.ii	Zerstörung des Klartextes eines Privilegzertifikats mit folgender (korrekter) Verschlüsselung	*TopLevelModel' V_AANack= 1`[e]*	70	0
Ziel: Betrug durch Manipulation von Zertifikatinhalten				
2.2.c.i	Manipulation eines Schlüsselzertifikats mit dem Ziel, sich in der AAI-Schicht als Entität `"E"` auszugeben	*processAA' V_AuthSuccessful <> empty*	106	0
2.2.c.ii	Manipulation eines Privilegzertifikats mit dem Ziel, ein „fremdes“ Privileg zu beanspruchen	*TopLevelModel' V_AASuccess <> empty*	48	0
2.2.d.i	Neuzusammenstellung eines Schlüsselzertifikats mit gefälschter Signatur	*processAA' V_AuthSuccessful <> empty*	75	0
2.2.d.ii	Neuzusammenstellung eines Privilegzertifikats mit gefälschter Signatur	*TopLevelModel' V_AASuccess <> empty*	50	0

Tabelle 7.3: Ergebnisse der Zustandsraumanalyse - Fall 2.2: Manipulation eines Zertifikats

7.3 Fazit

Die stochastische Analyse der Datenverfügbarkeit in Kademlia-basierten Systemen zeigte bereits in Abschnitt 4.3, dass die Verfügbarkeit von Zertifikaten und Statusinformationen durch die Wahl eines sinnvollen Replikationsfaktors (und ggf. Republishing-Intervalls) in ausreichendem Maße gesichert werden kann. Somit gilt das Sicherheitsziel der Verfügbarkeit als erreicht.

Auf Basis dieser Ergebnisse wurde im aktuellen Kapitel die Einhaltung der Sicherheitsziele Vertraulichkeit und Verfügbarkeit untersucht. Es wurde anhand einer Erweiterung des in Kap. 6 entwickelten HCPN das Vorgehen eines Angreifers mit Angriffsziel Vertraulichkeit dargestellt. Dies zeigt, dass die Vertraulichkeit von Datensätzen nicht gegenüber einem böswilligen Speicherknoten geschützt werden kann. Sich daraus ergebende Einschränkungen hinsichtlich der Einsatzszenarien von *P2P-ZuSI* werden in Kapitel 8 diskutiert.
Ein Angreifer kann aber außer dem reinen Informationsgewinn keinen Vorteil daraus ziehen, die gespeicherten Daten im Klartext zu kennen: er kann zwar gezielt bei ihm gespeicherte Daten löschen oder beschädigen, aber sie nicht aus dem Netz entfernen, da die Replikations- und Republishing-Mechanismen von Kademlia dies verhindern.
Weiterhin kann er, wie die formale Analyse in Form einer Untersuchung der Zustandsräume des hierarchischen gefärbten Petrinetzes für entsprechende Initial Markings bestätigt, keine Zertifikate oder Statusinformationen so manipulieren, dass das Ergebnis eines A&A-Prozesses verfälscht wird: es ist ihm weder möglich die Erstellung einer potentiell herleitbaren Zertifizierungskette[7] durch die Publikation gefälschter Daten zu verhindern noch durch Modifikation von Daten die erfolgreiche Zusammenstellung und Validierung falscher Zertifizierungsketten zu erreichen.

Durch die allgemein in zertifikatbasierten AAI - auch in solchen *ohne* verteiltes Verzeichnis - eingesetzten Techniken der Integritätssicherung, d.h. Signatur von Zertifikaten und Hashing von Statusinformationen, entstehen also durch den Einsatz eines verteilten Verzeichnisses zur Datenverwaltung *keinerlei* gravierende[8] Nachteile hinsichtlich des essentiellen Sicherheitsziels Integrität.

P2P-ZuSI bietet also hinsichtlich der Verfügbarkeit und der Integrität starken Schutz gegen böswillige Systemteilnehmer und kann daher als sicheres Verzeichnissystem für zertifikatbasierte Authentifizierungs- und Autorisierungsinfrastrukturen ohne hohe Vertraulichkeitsanforderungen (siehe Kapitel 8) eingesetzt werden.

[7] d.h. einer solchen, für die alle notwendigen Zertifikate und Statusinformationen im Verzeichnis existieren

[8] Die Erzeugung von geringem Zusatzaufwand für einen Prüfer, der nach Entdecken eines fehlerhaften Datensatzes eine neue Anfrage starten muss, ist keinesfalls als gravierend anzusehen.

Kapitel 8

Einsatzszenarien

Das Paradigma der Dezentralisierung von netzwerkbasierten Informationssystemen wird voraussichtlich in den nächsten Jahren weitere Verbreitung erfahren. Das vorgestellte *P2P-ZuSI*-System zur Verwaltung von AAI-Daten in einem Kademlia-basierten Peer-to-Peer-Netzwerk kann in verschiedenen Formen in bestehende dezentralisierte Systeme integriert oder als eigenständige Anwendung realisiert werden. Auch für einige Systeme mit zentralisierter Organisationsstruktur ist *P2P-ZuSI* eine interessante Alternative, um Authentifizierung und Autorisierung effizient und skalierbar umzusetzen.

In diesem Kapitel werden einige mögliche Einsatzszenarien für *P2P-ZuSI* präsentiert. Dabei werden nicht nur verschiedene Motivationen zur Entscheidung für ein Peer-to-Peer-Verzeichnis, sondern auch die Rahmenbedingungen hinsichtlich der Eigenschaften der Benutzergruppe, zu erwartendem Fluktuations- und Angriffsverhalten und entstehender Vorteile und Schwierigkeiten vorgestellt.

8.1 Offene Benutzergruppe: Internet

8.1.1 Existierende Web-of-Trust-Systeme: PGP

Ein naheliegender Anwendungsfall ergibt sich aus dem offensichtlichen Paradigmenbruch, der aktuell zwischen dezentralisierten Web-of-Trust-Architekturen in PKI (und vermutlich in Zukunft auch in AAI allgemein) und zentraler Datenspeicherung herrscht. Pretty Good Privacy (PGP) [PGP] ist dafür ein hervorragendes Beispiel: auf PKI-Ebene stellen sich beliebige, gleichberechtigte Entitäten gegenseitig Schlüsselzertifikate aus. Jeder Inhaber eines Zertifikats besitzt dabei die Möglichkeit, dieses selbständig zurückzurufen. Die Benutzergruppe ist für alle Internetnutzer offen.
Es handelt sich hier also um ein Paradebeispiel einer Anwendung mit vollständig dezentralisierter Organisationsstruktur. Die Veröffentlichung der Schlüsselzertifikate wird aber auf zentralen Servern realisiert, die beispielsweise von Universitäten oder Forschungseinrichtungen wie dem MIT oder dem Deutschen Forschungsnetz (DFN, Betreiber des deutschen Root Key Servers) betrieben werden.
Der Einsatz von *P2P-ZuSI* kann diesen Paradigmenbruch auflösen und die organisatorische Verantwortung für die Daten in die Hände der Benutzer legen. Da Schlüsselzertifikate keine vertraulichen Daten beinhalten, ist die verteilte Zertifikatspeicherung ohne Risiko.

So wird nicht nur organisatorische Unabhängigkeit erreicht und eine vollständig dezentralisierte Plattform geschaffen, sondern es sinken auch die Gesamtkosten für das System, während sich die Skalierbarkeit verbessert. Insbesondere wäre der Betrieb multipler redundanter Server für die verschiedenen Länder hinfällig. Die regelmäßige Erzeugung von Statusinformationen ersetzt dann die bisherigen situativ ausgestellten Rückrufe durch die Zertifikatinhaber, bei denen bereits jetzt die Kontrolle für Rückrufe liegt. Daher ist das NewPKI-System zur Berechnung von Statusinformationen (d.h. auf Anwendungsschicht) optimal in Kombination mit *P2P-ZuSI* einsetzbar: auch hier kontrollieren die Inhaber den Status ihrer Zertifikate selbst.
Die Tatsache, dass PGP-Schlüsselzertifikate nicht nur Signatur-Testschlüssel, sondern auch Chiffrierungsschlüssel für geheime Nachrichten enthalten können, ist insbesondere zu berücksichtigen. Solange ein Benutzer sein Chiffrier-Schlüsselpaar nutzen will, muss er also regelmäßig Statusinformationen publizieren[1]. Die Integration von *P2P-ZuSI* inklusive der Methoden für Erzeugung und Unterbindung (als Äquivalent zum derzeit notwendigen Rückruf) von Statusinformationen in bestehende Software könnte dabei das verteilte Verzeichnis nahezu transparent für den Nutzer umsetzen.

8.1.2 Erweiterung für Anwendungen im Internet: Filesharing und Groupware

Im Internet angesiedelte Filesharing-Systeme setzen sich vor allem aus Peers von Privatpersonen zusammen, welche beliebige Inhalte - oft Musik- und Videodateien - austauschen. Die Benutzergruppe ist unbeschränkt, d.h. prinzipiell kann jeder Rechner mit Internetzugang als Knoten betrieben werden. Problematisch an den existierenden Tauschbörsen wie eMule oder BitTorrent sind die häufig schlechten Download-Raten insbesondere bei großen Datensätzen (da immer noch viele Knoten Netzwerkverbindungen mit geringer Uploadgeschwindigkeit haben) und der illegale Tausch von urheberrechtlich geschützten Inhalten. In der letzten Zeit nimmt aber insbesondere die Verteilung unbedenklicher Inhalte wie frei verteilbarer Musik noch wenig bekannter Künstler oder unter GPL (General Public License) veröffentlichter Software zu.
Denkbar wäre in diesem Zuge, die legale Nutzung von Filesharing zu forcieren, indem Authentifizierungsdienste eingerichtet werden, welche den Zugriff auf bestimmte Daten reglementieren. Dies ist aber nur dann möglich, wenn das Datennetzwerk als unstrukturiertes Peer-to-Peer-Netzwerk realisiert ist: in einem strukturierten Netzwerk bewegen sich die publizierten Daten auf den wechselnden zuständigen Knoten, was es für den Besitzer eines Datensatzes unmöglich macht, eine Zugriffskontrolle zu realisieren.
Im Rahmen von BitTorrent beispielsweise ist der Einsatz von *P2P-ZuSI* zur Verwaltung von Zertifikaten möglich: hier werden Datenblöcke (Torrents) unstrukturiert gespeichert, während die Suche (Tracker) nach den Speicherknoten mit Hilfe eines Kademlia-Netzwerks realisiert ist. Auf letzteres könnte auch *P2P-ZuSI* aufgesetzt werden. Ein Knoten, welcher Torrents anbietet, kann dann anhand von Zertifikaten prüfen, ob ein Anfrageknoten zum Download berechtigt ist. Eine offene Frage ist hierbei allerdings, wie die Weiterverbreitung einmal heruntergeladener Torrents unterbunden oder einge-

[1] Dies ist bei kurzfristiger Abwesenheit auch im Rahmen der Publikation mehrerer Statusinformationen für zukünftige Perioden aufgrund der gewährleisteten Vorhaltezeit von Daten (siehe 4.3) möglich, sollte aber nur mit äußerster Vorsicht angewendet werden.

schränkt werden kann, da durch die offene, unreglementierte Benutzergruppe verstärkt mit Angriffen, aber auch mit nicht regelkonformem Verhalten hinsichtlich der Bereitstellung von nicht frei verteilbaren Datensätzen zu rechnen ist. Daher bieten sich als Objekte eher Datensätze an, deren unkontrollierte Weitergabe für autorisierte Entitäten nicht attraktiv ist. Dazu gehören z.B. interne Daten von Arbeitsgruppen wie Seminararbeiten, Rechercheergebnisse oder selbst erstellte Bild- und Videomaterialien.
Die Daten können dann direkt bei den Besitzern in Form von Torrents abgelegt werden und sind über den Kademlia-Tracker auffindbar. Fragt eine Entität nach dem entsprechenden Torrent, wird zunächst ein A&A-Verfahren zwischen dieser und dem Anbieter durchlaufen. Der Einsatz einer AAI ist nur dann sinnvoll, wenn die Daten bei mehreren Benutzern vorgehalten werden - ansonsten könnte der einzige Anbieter ja einfach lokal speichern, wer Zugriff hat.
Dies ist beispielsweise der Fall, wenn die autorisierte Benutzergruppe für bestimmte Daten durch jedes Mitglied eigenständig erweitert werden kann. Ein neuer Gruppenteilnehmer kann dann aufgenommen werden, wenn ein beliebiges Mitglied bereit ist, ihm ein (ggf. pseudonymisiertes) Schlüsselzertifikat auszustellen. Weiterhin kann z.B. mit Deskriptiven Attributzertifikaten gearbeitet werden, um dem neuen Teilnehmer das Attribut „Mitglied der Gruppe X“ zuzuordnen. Ein Privilegzertifikat, welches allen Mitgliedern der Gruppe X Zugriff auf bestimmte Daten erlaubt, ermöglicht dann die konkrete Autorisierung.
Es ist also möglich, Benutzergruppen in einem Filesharing-Netzwerk mit unstrukturierten und strukturierten Komponenten die Möglichkeit der Umsetzung von Zugriffskontrollmechanismen auf Basis einer AAI mit *P2P-ZuSI* als Verzeichnis zu geben.

Eine Erweiterung dieses Einsatzszenarios stellt P2P-Groupware dar. Unter Groupware versteht man eine Software, welche die Zusammenarbeit von Gruppen unterstützt, beispielsweise in Form von gemeinsamem Speicherplatz für Daten, Interaktionsmöglichkeiten wie Foren oder gemeinsamer Verwaltung von Kontaktdaten. Hier kann das oben skizzierte System zur Authentifizierung und Autorisierung der Gruppenmitglieder zum Einsatz kommen, wobei ebenfalls mit einer Datenspeicherung in einem unstrukturierten Netzwerk gerechnet wird. Der Unterschied zum obigen Szenario liegt insbesondere darin, dass hier keine Web-of-Trust-Architektur zu erwarten ist, sondern eine privilegierte Entität (z.B. Gruppenleiter) allein die Berechtigung zur Zertifizierung besitzt. Da *P2P-ZuSI* von der AAI-Architektur unabhängig ist, ergibt sich diesbezüglich aber kein Anpassungsbedarf.

Allgemein sind die in beiden Szenarien verwendeten Attribute und Privilegien kaum als vertraulich einzustufen: durch Pseudonymisierung bei Ausstellung von Schlüsselzertifikaten und Deskriptiven Attributzertifikaten zur Gruppenzuordnung von Entitäten kann die tatsächliche Identität der Gruppenmitglieder verborgen werden. Auch die Zugriffsprivilegien sind für Außenstehende vermutlich wenig interessant, so dass durch die Verteilung im Netzwerk nur geringe Risiken entstehen.

8.1.3 Datenschutz und Integritätssicherung gegenüber Dienstbetreibern und Dritten: Instant Messaging

Viele Instant-Messaging-Anwendungen wie ICQ [ICQ] sind zwar im Grundsatz P2P-basiert, verwenden aber zur Unterstützung zentrale Server des Betreibers, im Falle von

ICQ beispielsweise des Unternehmens AOL. Gerade diese Struktur ist sehr unsicher gegenüber Abhörmaßnahmen und wenig datenschutzfreundlich.
Neben Authentifizierung mit Hilfe von Signaturen erlauben Schlüsselzertifikate auch die Etablierung verschlüsselter Kommunikationskanäle, in welchen die Nachrichten stets nur vom Empfänger entschlüsselt werden können. Der Einsatz von *P2P-ZuSI* zur Verteilung solcher Zertifikate zwischen den Teilnehmern einer Instant-Messaging-Anwendung könnte die Vertraulichkeit des Dienstes gegenüber externen Angreifern, aber auch gegenüber dem Dienstbetreiber selbst signifikant verbessern.
Dafür ist sowohl ein eigenständiges Web of Trust als auch die Verwendung bestehender Dienste wie PGP mit dezentralisiertem Verzeichnis wie in 8.1.1 beschrieben möglich.

8.2 Geschlossene Nutzergruppe: Unternehmen und Organisationen

8.2.1 Serverlose Hochleistungsumgebungen: Cluster

Cluster sind Zusammenschlüsse von Computersystemen in einem Netzwerk, die gemeinsam eine rechen- oder datenintensive Anwendung anbieten. Aufgrund der geringen Kosten sind PC-Cluster vor allem in Universitäten auf dem Vormarsch [DSS05]. Durch die häufig lokale Beschränkung der angebotenen Anwendung (z.B. Zugriffe nur aus dem Netzwerk der Universität) sind solche Cluster in der Regel in kontrollierten, abgeschlossenen Netzwerkumgebungen angesiedelt. Daraus ergeben sich positive Faktoren für den Einsatz einer *P2P-ZuSI*-basierten AAI, wobei die Authentifizierung und Autorisierung entweder konkret auf die vom Cluster angebotene Anwendung bezogen sein kann oder aber nur die Plattform mit der Cluster-Anwendung teilt, aber den A&A-Dienst für einen anderen Zweck anbietet.
In jedem Fall ist die Verwendung von Cluster-Rechnern als Peer-to-Peer-Knoten möglich, wobei die Angriffswahrscheinlichkeit aufgrund der kontrollierten Umgebung vernachlässigbar gering ist. Ebenso ist erwartungsgemäß die Fluktuationswahrscheinlichkeit sehr niedrig, da die Teilnehmer eines Clusters in der Regel ständig online sind. Fluktuation beschränkt sich so auf Ausfälle von Computersystemen und Auslastung des Systems durch die Cluster-Anwendung, bei welcher *P2P-ZuSI* als Anwendung geringerer Priorität zu behandeln ist. Kann ein Cluster-Computer also nicht mehr seine primäre Anwendung und den Peer-to-Peer-Knoten gleichzeitig ausführen, wird die Knotensoftware deaktiviert. So ist die flexible Nutzung nicht ausgelasteter Teilnehmer möglich.
Die prinzipielle Plattformunabhängigkeit erlaubt den Betrieb von *P2P-ZuSI* auf Basis verschiedener Betriebs- und Dateisysteme. Voraussetzung ist aber, dass die verwendete Cluster-Software die Knoten nicht zu einem einzigen, rechnerübergreifenden Betriebssystem zusammenschließt - dann wäre ein P2P-Netzwerk nämlich sinnlos, da dieses gemeinsame Cluster-Betriebssystem als Plattform für die verteilte Speicherung von Daten genutzt werden könnte.
Sind die Nutzer der AAI nicht die Cluster-Rechner selbst, ist das Peer-to-Peer-Paradigma nicht vollständig erfüllt. In einem solchen Fall müssten Nutzer Zugriff auf bestimmte oder beliebige Cluster-Rechner erhalten, um mit Hilfe von deren P2P-Knoten die benötigten Daten zu beziehen. Dann liegt kein reines Peer-to-Peer-System mit gleichberechtigten Entitäten vor, sondern eine Art Client-Server-System, bei welchem die Server-Komponente

als P2P-System realisiert ist. Skalierbarkeit, Flexibilität und Kosteneinsparung sind aber deutliche Argumente für eine solche Lösung.
Im Rahmen der Nutzung existierender Cluster könnten Anwendungsmöglichkeiten für AAI mit *P2P-ZuSI*-Verzeichnis in Universitäten oder Banken erschlossen werden.

8.2.2 Umgebungen mit stabilen ungenutzten PC-Kapazitäten: Behörden und Unternehmen

Unternehmen und Behörden mit vielen Mitarbeitern und dementsprechend vielen vernetzten PC-Arbeitsplätzen bieten gute Möglichkeiten zur Einführung von *P2P-ZuSI*, beispielsweise um eine AAI für gemeinsam genutzte Ressourcen zu unterstützen. Dies kann beispielsweise ein virtuelles Laufwerk in Form eines verteilten Dateisystems auf Basis eines P2P-Netzwerks mit nicht ausgelasteten Bürorechnern als Peers sein (siehe Abschnitt 4.1). Die A&A-Komponente ermöglicht es dann beispielsweise, den Zugriff auf das Dateisystem an sich zu reglementieren (Beweis der Zugehörigkeit zur Nutzergruppe) und Änderungen an der Datenbasis nachzuvollziehen (Signatur durch den editierenden Knoten). Erwartungsgemäß werden hier hierarchische oder hybride AAI-Architekturen vorherrschen.
Die technische Netzwerkkonfiguration in einem solchen Umfeld schließt dann das P2P-System nach außen, d.h. gegen das Internet ab, so dass nur Rechner innerhalb des Unternehmens oder der Behörde als Knoten verwendet werden können. Verschiedene Subnetze, z.B. an unterschiedlichen Standorten, können beispielsweise per VPN (Virtual Private Network) verbunden werden.
Aufgrund der in der Regel statischen IP-Adressen der Rechner ist auch die Erzeugung einer mittel- bis langfristig gültigen NodeID aus der IP-Adresse möglich. So können auch Angriffe leicht nachvollzogen werden, zumal die Bediener von PCs hier oft keine Administrationsrechte für ihr System besitzen und damit eine Verschleierung von Fehlverhalten schwierig ist. Es ist zu erwarten, dass die Angriffswahrscheinlichkeit hier sehr niedrig ist und daher geringe Replikationsfaktoren nötig sind.
Zu berücksichtigen ist aber die Tageszeitproblematik: befinden sich die beteiligten Rechner in derselben Zeitzone, so ist zu erwarten, dass außerhalb der normalen Bürozeiten die Anzahl der Knoten stark zurückgeht [YTD06]. Daher ist eine *P2P-ZuSI*-Lösung zur AAI-Datenverwaltung nur dort einsetzbar, wo Knoten existieren, die außerhalb der Arbeitszeiten nicht abgeschaltet sind. Hierzu zählen beispielsweise PCs und Workstations in Entwicklungsumgebungen (IT-Abteilung) oder auch Intranet-Server und Router.
P2P-ZuSI ist also auch für Unternehmen und Behörden geeignet. Dabei ist zu berücksichtigen, dass in der AAI nur Daten verteilt gespeichert werden sollen, die hinsichtlich ihrer Inhalte nicht vertraulich sind. Privilegzertifikate mit Zugriffsrechten für bestimmte Verzeichnisse oder Deskriptive Attributzertifikate, welche einen Mitarbeiter einer Abteilung zuordnen und so indirekt dessen Berechtigungen festlegen, sind beispielsweise unkritisch.

8.2.3 Umgebungen mit ungenutzten heterogenen Kapazitäten: Universitäten

Während in Unternehmen und Behörden die Nutzer der beteiligten Knoten in der Regel feststehen, können in Universitäten die Benutzer häufig wechseln. Dies entspringt der

Tatsache, dass hier nicht nur Rechner in Büros als Knoten verwendet werden können, sondern auch die vielen PCs[2] in für Studenten zugänglichen Computerräumen. Dadurch wird einerseits die Fluktuationswahrscheinlichkeit im gesamten Netzwerk erhöht - solche Knoten bleiben erwartungsgemäß kürzer online und nehmen zudem nur unregelmäßig am P2P-Netzwerk teil - andererseits aber auch die durchschnittliche Angriffswahrscheinlichkeit, da Nicht-Mitarbeiter eher böswilliges Verhalten zeigen werden als Angestellte und Beamte. Im Schnitt werden aber sowohl Angriffs- als auch Fluktuationswahrscheinlichkeit noch weit unter den Gegebenheiten in einem Netzwerk mit offener Benutzergruppe liegen.
Im Bereich der Authentifizierung könnte hier (ebenso wie auch im Szenario aus 8.2.2) eine hierarchische AAI mit dezentralisierter Speicherung beispielsweise die UnP-Authentifizierung von Benutzern lokal und im Netzwerk ersetzen, wobei in diesem Fall die Existenz eines Schlüsselpaars auf einem externen Medium (z.B. SmartCard) nötig ist. Privilegzertifikate können die Autorisierung für Fernleihe, elektronische Zeitschriftenbibliotheken oder Druckdienste ermöglichen.
In einem solchen Umfeld wird also, ebenso wie in Unternehmen und Behörden, eine hierarchische oder hybride AAI-Architektur vorherrschen. Dadurch wird es möglich, für diese speziellen Anwendungsfälle das *P2P-ZuSI*-Konzept mit SmartCards zu kombinieren, da beispielsweise Universitätsverwaltungen allgemein vertrauenswürdige Instanzen darstellen. Wie auch in Unternehmen und Behörden liegt der Fokus hier nicht auf der vollständigen Dezentralisierung von Diensten, sondern auf praktischen Erwägungen der Kostenoptimierung und Skalierbarkeit.

8.3 Wegbereitung für neue Peer-to-Peer-Anwendungen

Auch wenn heute die meisten daten- oder rechenintensiven Anwendungen nach dem Client-Server-Paradigma umgesetzt werden, kann *P2P-ZuSI* durch das Angebot eines plattformunabhängigen, kostengünstigen und skalierbaren Datenverwaltungssystems für AAI-Daten die einfache Realisierung von Authentifizierung und Autorisierung in vielfältigen Systemumgebungen ermöglichen. Damit kann ein Anreiz geschaffen werden, die Dezentralisierung auch von sicherheitsrelevanten Diensten allgemein voranzutreiben und so die Vorteile von Peer-to-Peer auch in neuen Anwendungsbereichen zu nutzen.
Erwartungsgemäß wird sich die Dezentralisierung in vielen Fällen auf ein kontrolliertes Umfeld wie die nicht ausgelasteten PC-Ressourcen in Unternehmen beschränken (siehe 8.2.2, 8.2.3), so dass die Möglichkeiten der dezentralisierten Organisation hier nicht von Bedeutung sind, sondern v.a. teure und arbeitsintensive Server ersetzt werden sollen.
Auch standortübergreifende Anwendungen von Peer-to-Peer sind ein interessantes Anwendungsgebiet, sofern Möglichkeiten zur subnetzübergreifenden Knotenkommunikation existieren, wie z.B. ein Virtual Private Network.

Aber auch neue, von Privatpersonen oder lose organisierten Gruppen (z.B. der Open-Source-Community) gemeinsam angebotene Dienste im Internet können von *P2P-ZuSI* profitieren. A&A-Dienste können dann - wie auch im Beispiel PGP gezeigt - einer großen

[2] Im Jahr 2007 existieren beispielsweise an der Universität Regensburg fast 650 PCs in solchen Computer-Pools [RZURG].

Nutzergruppe zur Verfügung stehen und deren selbstorganisierte Kommunikation, Interaktion und Kollaboration fördern.

Zusätzlich ist *P2P-ZuSI* gut einsetzbar, wenn der Nutzen einer AAI erst evaluiert werden soll oder unklare Anforderungen hinsichtlich des entstehenden Aufwands, der Größe und Beschaffenheit von Nutzergruppen und dem Datenumfang bestehen. So werden die Einsatzhürden für A&A-Dienste prinzipiell gesenkt, da die Anschaffung eines leistungsfähigen Servers trotz unklarer Rahmenbedingungen dann nicht mehr nötig ist.

8.4 Fazit

Es wurden verschiedene Szenarien vorgestellt, in denen *P2P-ZuSI* praktisch eingesetzt werden kann. Abbildung 8.1 stellt diese hinsichtlich der erwarteten Fluktuation, Angreiferdichte und Offenheit der Benutzergruppe gegenüber.

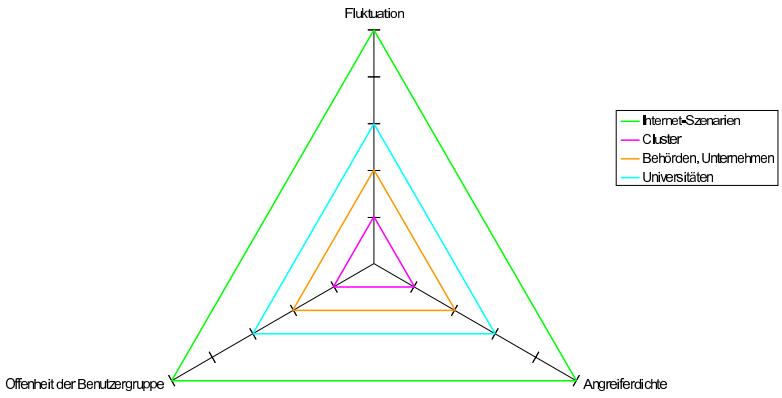


Abbildung 8.1: Taxonomie der Einsatzszenarien

Es ergibt sich die Möglichkeit zur Beseitigung des Paradigmenbruchs für Web-of-Trust-basierte AAI-Ansätze, wie am Beispiel PGP vorgestellt. Auch ist der Einsatz in File-sharing- und Groupware-Anwendungen im Internet gegeben. Dadurch wird die Interaktion von Teilnehmern in flexiblen und zielgerichteten Gruppenstrukturen erleichtert, was beispielsweise im universitären Umfeld für studentische Projekt- oder Referatsgruppen genutzt werden kann.

Vor dem Hintergrund ungenutzter Ressourcen und Kosten- sowie Skalierbarkeitsgesichtspunkten ist *P2P-ZuSI* als Verzeichnisdienst für zertifikatbasierte AAI auch in Unternehmen, Behörden, Universitäten u.ä. auf Bürocomputern, Workstations oder Clustern realisierbar, wobei hier die geringeren Fluktuations- und Angriffswahrscheinlichkeiten positiv zu werten sind, wobei aber der Dienst nur einer geschlossenen, fest definierten Benutzergruppe zur Verfügung steht. Während hier gezielt zusätzliche Sicherheitstechniken wie z.B. SmartCards eingesetzt werden können, welche eine vertrauenswürdigen Instanz benötigen, ist organisatorische Unabhängigkeit und Selbstorganisation der Teilnehmer kein Ziel und Inhalt der Anwendung. Der Einsatz von *P2P-ZuSI* wäre hier also grund-

legend anders motiviert als bei einer Realisierung im Internet mit offener Benutzergruppe.

In beiden Kategorien von Einsatzszenarien ist *P2P-ZuSI* eine interessante und zukunftsfähige Alternative zu serverbasierter Verwaltung von AAI-Daten.

Kapitel 9

Fazit

9.1 Ergebnisse der Arbeit

Um ein verteiltes Verzeichnis für AAI-Daten zu realisieren, wurde zunächst aus den zur Verfügung stehenden strukturierten Peer-to-Peer-Regelwerken das optimale Regelwerk, Kademlia, ausgewählt. Die Entscheidungskriterien bezogen sich daher auf die inhärenten Sicherheitseigenschaften der Kandidaten, die für ein verteiltes Datenverwaltungssystem von Vorteil sind.

Erstmals wurde die Verfügbarkeit von Datensätzen in Kademlia-Netzwerken auf einer stochastischen Basis formalisiert und unter realistischen Annahmen analysiert. Insbesondere die Berücksichtigung von böswilligem Verhalten hebt dieses Formelwerk deutlich von bisherigen unzureichenden Analyseansätzen ab.
Die Anwendung dieser Formeln ermöglicht nicht nur die Berechnung der Verfügbarkeitswahrscheinlichkeit zu jedem beliebigen Zeitpunkt nach Veröffentlichung eines Datensatzes, sondern auch die Optimierung des Replikationsfaktors anhand der gewünschten Verfügbarkeitsdauer und der Rahmenparameter (angenommene Fluktuations- und Angriffswahrscheinlichkeit). So kann je nach konkretem Einsatzszenario der optimale Wert für k gewählt werden.

Auf dieser Basis wurde ein Regelwerk definiert, welches die Nutzung eines Kademlia-Netzwerks als verteiltes Verzeichnis für AAI-Daten ermöglicht: das *P2P-ZuSI*-Regelwerk. Hier wurden Statusinformationen als Gültigkeitsbeweis von Zertifikaten gewählt, um falsche Gültigkeitsannahmen unmöglich zu machen. Zudem eignen sich diese kleinen Datensätze optimal für die Speicherung in einem Peer-to-Peer-System.
Das *P2P-ZuSI*-Regelwerk wird als abgeschlossene Schicht definiert, welche für die darüberliegende AAI-Anwendung die Details des darunter liegenden P2P-Netzwerks vollständig abstrahiert, indem sie nur Anfrage- und Publikationsmethoden für Zertifikate und Statusinformationen anbietet. Innerhalb der Regelwerk-Schicht werden dann die FileIDs auf Basis inhaltsbezogener Informationen über die Datensätze generiert, um die gespeicherten Daten für jeden möglichen Prüfer auffindbar zu machen. Zudem implementiert die Regelwerk-Schicht die einfache symmetrische Verschlüsselung der Zertifikate, um einen Basisschutz gegenüber dem Ausspähen von Daten zu bieten.

Die Kombination des Kademlia-basierten Netzwerk mit diesen definierten Regeln wird als *P2P-ZuSI* bezeichnet. Das Schichtenmodell erlaubt den Einsatz von *P2P-ZuSI* als Verzeichnisdienst mit beliebigen zertifikatbasierten Authentifizierungs- und Autorisierungsinfrastrukturen unabhängig von deren Architektur. Durch die Beschränkung auf vier von der AAI-Schicht zugreifbare Methoden zur Publikation und Anfrage von Datensätzen kann *P2P-ZuSI* ohne großen Anpassungsaufwand in zertifikatbasierte AAI, PKI oder PMI integriert werden.

Das aus dem abstrakten Schichtenmodell entwickelte konkrete formale Modell für die Kombination aus einer abstrakten zertifikatbasierten AAI und *P2P-ZuSI* als hierarchisches gefärbtes Petrinetz bildet die handelnden Entitäten und ihre Aktionen in rollenbasierter Form ab. Neben der computergestützten Simulation von Datenflüssen ermöglicht es auch die Weiterentwicklung von *P2P-ZuSI* und kann außerdem als Ausgangspunkt für eine softwaretechnische Umsetzung verwendet werden.

Der Nachweis der Sicherheitseigenschaften von *P2P-ZuSI* stellt die Arbeit auf ein solides Fundament. Zusätzlich zur bereits erwähnten stochastischen Analyse der Verfügbarkeit wurden Sicherheitsziele der Vertraulichkeit und Integrität und deren Erfüllung diskutiert. Die Vertraulichkeit von Zertifikaten kann - trotz der eingeführten Verschlüsselung - nicht sichergestellt werden, ist allerdings auch nur in wenigen Fällen notwendig.
Die Sicherheit gegenüber Integritätsangriffen durch böswillige Teilnehmer des P2P-Netzwerks wurde anhand der Zustandsraumanalyse des HCPN-Modells unter Verwendung geeigneter Initial Markings gezeigt.

Trotz der Verteilung von Daten auf nicht vertrauenswürdige Knoten bietet das entwickelte System also ein gutes Sicherheitsprofil, das vielfältige Einsatzmöglichkeiten im Internet, in Unternehmen oder Universitäten eröffnet.

9.2 Ausblick

9.2.1 Forschung

Im theoretischen Bereich kann das verteilte Datenverwaltungssystem noch an einigen Stellen erweitert werden, insbesondere auf P2P-Ebene. Ein offener Punkt ist beispielsweise die Adaption des Knotenauthentifizierungsmechanismus von P-Grid auf Kademlia. Auf dieser Grundlage kann dann auch ein Einsatzkonzept für Reputationsmanagement zwischen Peers (für die reine Bewertung von Transaktionsverhalten) erstellt werden.
Neue Berechnungsmethoden für NodeIDs sind gerade im Zusammenhang mit der Einführung von IPv6 und der damit zunehmenden Gefahr von Sybil Attacks ein wichtiger Punkt der zukünftigen Peer-to-Peer-Forschung.
Die Entwicklung neuer Metriken und verbesserter Untersuchungsmethoden für Knotenverfügbarkeit und Fluktuationswahrscheinlichkeit in realen Netzwerken ist ein notwendiger Schritt zur Herleitung genauerer Aussagen über die optimalen Replikationsfaktoren im praktischen Einsatz. Weiterhin ist zu überlegen, ob auch in Kademlia Erasure Coding statt klassischer Replikation eingesetzt werden kann, um insbesondere in stark fluktuierenden Netzwerken die Redundanz und somit die entstehende Datenmenge zu verringern.

Die Entwicklung weiterer Einsatzszenarien für *P2P-ZuSI* ist erstrebenswert, um dieses System möglichst vielseitig zu nutzen. Bei Szenarien mit geschlossenen Nutzergruppen ist die Möglichkeit der Verwendung zusätzlicher Sicherheitstechniken wie z.B. SmartCards ausführlich zu untersuchen. Insbesondere ist zu überlegen, ob für solche geschlossenen Systeme eine Möglichkeit zur Vertraulichkeitssicherung gefunden werden kann, welche sich durch gegenüber offenen Systemen im Internet abweichende Rahmenbedingungen (siehe 8.2.2, 8.2.3) auftun könnte.

9.2.2 Praxis

Zunächst ist *P2P-ZuSI* in eine quelloffene und plattformunabhängige Software umzusetzen. Dabei bietet sich Java als Programmiersprache der Wahl an. Als Kademlia-Implementierung sollte das in LimeWire [LIMEW] enthaltene Java-Framework *Mojito* verwendet werden: es stellt auf Basis der Java-Version 1.5 alle definierten Kademlia-Funktionen wie Lookup, Join usw. bereit und ist unter der GNU General Public License (GPL) veröffentlicht[1]. Es ist empfehlenswert, den Knotenbetrieb statt im Rahmen eines separaten Programms als *Dienst* zu realisieren, die automatisch gestartet und bis zur Abschaltung des Computersystems ausgeführt wird.
Die Software soll aus dem Schichtenmodell und dem HCPN, speziell der *P2P-ZuSI*-Regelwerk-Schicht, erzeugt werden und somit für die AAI-Schicht nur die definierten Schnittstellen (*put*- und *get*-Methoden) besitzen. Auch eine Basisimplementierung einer AAI mit einfachen Methoden zur (X.509-konformen) Zertifikaterstellung, der Generierung von Statusinformationen und Validierungsmethoden beim Prüfer ist zu erwägen, um Praxistests zu ermöglichen.
Anhand solcher „Feldversuche“ können dann weitere Untersuchungen durchgeführt werden, beispielsweise um die Benutzerakzeptanz in ausgewählten Testumgebungen zu bewerten und Performance- sowie Fluktuationsmessungen in einem realen Umfeld durchzuführen.
Mittelfristiges Ziel ist die Erstellung einer Open-Source-Software für *P2P-ZuSI*, welche leicht an bestehende und neue Systeme angeschlossen werden kann und somit den effizienten und zuverlässigen Einsatz von zertifikatbasierten AAI in dezentralisierten Umgebungen ermöglicht.

9.3 Abschluss

Die vorliegende Dissertation legt das theoretische Fundament für die konsequente Abbildung des Paradigmas der (insbesondere Peer-to-Peer-basierten) Dezentralisierung, welches aktuell stark an Bedeutung gewinnt, auf zertifikatbasierte Authentifizierungs- und Autorisierungsinfrastrukturen.
Die Sicherheitsuntersuchung belegt, dass die Nutzung eines verteilten Verzeichnisses trotz der hohen Wahrscheinlichkeit von Angriffen durch die Netzwerkteilnehmer auch für die Speicherung solcher sicherheitsrelevanter Daten möglich ist. Damit schafft *P2P-ZuSI*

[1] *Mojito* darf also frei verwendet und weiterentwickelt werden, sofern das entstehende Produkt wiederum unter GPL verfügbar gemacht wird.

einen neuen Anwendungsfall für das Peer-to-Peer-Paradigma und leistet zudem einen Beitrag zur Aufhebung von Inkonsistenzen in bestehenden dezentralisierten Diensten, die für A&A bisher auf zentralisierte Server angewiesen waren.

Anhang A

Mathematische Routinen der Stochastischen Analyse

Die folgenden Methoden zur stochastischen Analyse sind in C++ Notation beschrieben. Sie finden sich auch auf der beiliegenden CD im Verzeichnis `KadAnalyse`.

A.1 Iterative Berechnung der Wahrscheinlichkeitsverteilung

Da das rekursive Verfahren zur Berechnung der Wahrscheinlichkeitsverteilung von $\tilde{k}$ im Zeitpunkt t sich als zu aufwendig erwies, wurde ein iteratives Verfahren zu diesem Zweck entwickelt.

Dabei wird ein zweidimensionales Array mit $T + 1$ Zeilen und $K + 1$ Spalten (wegen $0 \leq \tilde{k} \leq K$) (siehe A.4) benutzt.
Die Berechnung beginnt in $t_i = 0$ ($0 \leq t_i \leq T$) gemäß Formel 4.7; die entstehende Wahrscheinlichkeitsverteilung wird in die 1. Zeile (`result_matrix[0]`) geschrieben.
Die Verteilung in $t_i = 1$ wird unter Verwendung von 4.11 erzeugt. Dabei werden die Werte aus der 1. Arrayzeile eingesetzt.
In $t_i = 2..t$ wird Formel 4.10 verwendet. Findet zwischen zwei Zeitpunkten t_{i-1}, t_i Republishing statt, wird zunächst aus der Verteilung in t_{i-1} gemäß Formel 4.15 die Verteilung im konkreten Republishing-Zeitpunkt t_r^* errechnet und diese als Grundlage für die Anwendung von 4.10 benutzt (im Quellcode als *Republishing Point* deklariert).

```
void calcProbabilityDistribution (double p_o_m, double p_a, int t, int k, int n)
{
  // distribution for t = 0
  for (int i = 0; i <= k; i++)
  {
    result_matrix[0][i] = gsl_sf_choose(k,i) *
                          pow(1-p_a, (double) i)*
                          pow(p_a,(double)k-i);
  }
  // distribution for t = 1
  for (int i = 0; i <= k; i++)
  {
```

```
  double result = 0;
  for (int j = 0; j <= k-i; j++)
  {
    result += result_matrix[0][k-j] * gsl_sf_choose(k-j, i) *
              pow(1-p_o_m, (double)i) *
              pow(p_o_m, (double)k-j-i);
  }
  result_matrix[1][i] = result;
}
// distribution within all other periods
// iteration of points in time
for (int ti = 2; ti <= t; ti++)
{
  // no republishing point
  if ((ti-1) % n != 0)
  {
    // iteration of replica numbers ~k
    for (int i = 0; i <= k; i++)
    {
      double result = 0;
      for (int j = 0; j <= k-i; j++)
      {
        result += gsl_sf_choose(i+j, i) *
                  pow(1-p_o_m, (double)i) *
                  pow(p_o_m, (double)j) *
          result_matrix[ti-1][i+j];
      }
      result_matrix[ti][i] = result;
    }
  }
  // republishing point:
  // distribution switches after republishing in t_r*,
  // i.e. changes take place in t_(r+1) = n+1, 2n+1, ...
  else
  {
    // i=0
    result_matrix[ti][0] = result_matrix[ti-1][0];
    // 1: REPUBLISHING - new distribution in t_r*
    // iteration on replica numbers
    for (int i = 1; i <= k; i++)
    {
      double result = 0;
      // probability that a node not having a replica
      // will now generate one
      double p_n_adds_1 = (double)((1- pow(1-p_o_m,(double)n)) * (1-p_a)) /
                          (double)(1-(pow((1-p_o_m),(double)n) * (1-p_a)));
      for (int j = 1; j <= i; j++)
      {
        double intermediate_result = result_matrix[ti-1][j] *
                                     gsl_sf_choose(k-j, i-j) *
                                     pow(p_n_adds_1, (double)i-j) *
                                     pow((1-p_n_adds_1), (double)k-i);
```

```
        result += intermediate_result;
      }
      result_matrix[ti][i] = result;
    }
    // 2. FLUCTUATION of nodes between t_r* and t_r
    for (int i = 0; i <= k; i++)
    {
      double result = 0;
      for (int j = 0; j <= k-i; j++)
      {
        result += gsl_sf_choose(i+j, i) *
                  pow(1-p_o_m, (double)i) *
                  pow(p_o_m, (double)j) *
                  result_matrix[ti][i+j];
      }
      result_matrix[ti][i] = result;
    }
  }
 }
}
```

A.2 Berechnung der Höhenlinie

Die Berechnung der 0.99-Höhenlinie der Funktion $p(\tilde{k} > 0)$ zu einem Zeitpunkt t wird experimentell durchgeführt.
Dafür wird die Methode *calculateLevelCurve* verwendet. Sie liefert für einen Bereich $k_{start} \leq k \leq k_{end}$ zu jedem k genau den Wert für p_{o_m} zurück, bei dem der Wert der Funktion gerade noch > 0.99 ist.

```
void calculateLevelCurve(double pom__step, double p_a, double
pom__start, double pom__end, int t, int n, int k_start, int k_end) {
  for (int i = k_start; i >= k_end; i--)
    {
      double last_seen = 1;
      for (double j = pom__start; j <= pom__end; j+=pom__step)
      {
        calcProbabilityDistribution(j, p_a, t, i, n);
        double current_seen = 1-result_matrix[t][0];
        if (last_seen >= 0.99 && current_seen < 0.99 && (j-pom__step > 0))
          {
            result_p[i] = (j-pom__step);
            break;
          }
        else
          last_seen = current_seen;
    }
  }
}
```

A.3 Heuristischer Ansatz zur Berechnung der Höhenlinie

Aufgrund der Tatsache, dass obiger Code bei einer Genauigkeit von 0.00001 (Schrittweite für p_{o_m}) eine Laufzeit von mehreren Wochen aufweist, wird ein auf einer Heuristik basierender optimierter Algorithmus eingesetzt:

```
void doHeuristicalCalculation(double pom__step, double p_a, double, double,
int t, int n, int k_start, int k_end)
{
    for (int k_step = k_end; k_step < k_start; ++k_step)
    {
        double delta = MAX(0, result_p[MAX(0,k_step-1)]) -
                       MAX(0, result_p[MAX(0,k_step-2)]);
        double current_pom__start = MAX(0, result_p[MAX(0,k_step-1)]+
                                    delta - POM_GAP;
        current_pom__start = MAX (0,current_pom__start);
        double current_pom__end = MAX(0, result_p[MAX(0,k_step-1)]) +
                                  delta + POM_GAP;
        calculateLevelCurve(pom__step,p_a,current_pom__start,
                            current_pom__end,t,n,k_step,k_step);
    }
}
```

Dabei wird ausgenutzt dass,

- die beschriebene Funktion streng monoton steigend ist (vgl. Abbildung 4.11) und
- sich der Wert der ersten Ableitung in einem engen Rahmen bewegt. Der Maximalwert wird offensichtlich bereits im Intervall $[10, 30]$ angenommen.

Diese Eigenschaften ermöglichen es das Iterationsintervall für die i-te Periode Anhand der interpolierten Steigung (ermittelt durch die Differenz des $(i-1)$-ten zum $(i-2)$-ten Ergebniswert) massiv zu beschränken.

A.4 Programmübersetzung, Initialisierung und Ausgabe

Die Mathematik-Bibliothek GSL (GNU Scientific Library) in der Version 1.9 ist Voraussetzung für die Übersetzung und Ausführung des abgebildeten Sourcecodes, ebenso `make` und die `gcc`. Die Eingabeparameter und Speicherbereiche werden wie folgt festgelegt:

```
#include <math.h>
#include <gsl/gsl_sf_gamma.h>

#define T         43200
#define K         120
#define N         60
#define P_A       0.5

#define POM_STEP  0.00001
#define POM_GAP   0.0005;
#define K_START   120
#define K_END     1
```

```
#define MAX(X,Y) ((X)>(Y)?(X):(Y))

double result_matrix[T+1][K+1];
double result_p [K+1];
```

Die eigentlich Berechnung wird dann mittels des Aufrufs

```
doHeuristicalCalculation(POT_STEP,P_A,POT_START,POT_END,T,N,K_START,K_END);
```

gestartet. Nach Ablauf des Programms beinhaltet das Array `result_p` die Wahrscheinlichkeiten der Ergebnismenge.

Anhang B

ML-Funktionen im Grundmodell

B.1 Standard-Methoden

```
(* multiple hashing: hashes a bitstring b i times *)
 fun H_M(b : BITSTRING, i: INT) : BITSTRING =
   if i > 0 then
     "H("^H_M(b, i-1)^")"
   else b

(* multiple hashing: hashes an integer b i times *)
fun H_M_I(b:INT, i:INT): BITSTRING =
  if i > 0 then
    "H("^H_M_I(b, i-1)^")"
  else INT.mkstr(b)
```

B.2 AAI-Methoden

B.2.1 Allgemeine Methoden

```
(* hashes the content of a certificate *)
fun hashCertcontent(c: CERTCONTENT):CONTENTHASH =
  "H("^(#1 c)^"+"^(#2 c)^"+"^(#3 c)^"+"^INT.mkstr(#4 c)
  ^"+"^INT.mkstr(#5 c)^"+"^(#6 c)^"+"^(#7 c)^")";

(* hashes a complete certificate including the signature *)
fun hashCert (c: CERT) : CERTHASH=
  let
    val cc = #1 c
    val sign = #2 c
  in
    ("H(("^(#1 cc)^"+"^(#2 cc)^"+"^(#3 cc)^"+"
    ^INT.mkstr(#4 cc)^"+"^INT.mkstr(#5 cc)^
    "+"^(#6 cc)^"+"^(#7 cc)^")+Signature("^
    (#1 sign)^"+("^(#1(#2 sign))^"+"^
    (#2(#2 sign))^")+"^
    SIGN_TYPE.mkstr(#3 sign)^"))")
  end;
```

```
(* sign the content of a certificate in a way that it can be
checked with the given public key *)
fun sign(c : CERTCONTENT, k : PUBKEY):SIGNATURE =
  (hashCertcontent(c), k, s);

(* sign a challenge in a way that it can be checked with
the given public key *)
fun signChallenge(c: CHALLENGE, k: PUBKEY):CHALLSIGNATURE =
  (c, k, s);
```

B.2.2 Methoden zur Zeitbehandlung

```
(* get the current period (i) of certificate validity *)
fun getCurrentPeriod(cert:CERT, date:DATE):PERIOD =
  (* i = current date - begin date of certificate validity + 1 *)
  (date+1-(#4(#1 cert)));
```

B.2.3 Methoden zur Bereinigung von Datenlisten

```
(* remove elements from list of si seeds (basic colour set
including random number, begin and end of validity) that are
no longer valid due to revocation or normal end of
certificate validity *)
fun cleanSiSeedList(sisl: SI_SEED_LIST,
date : DATE, revl : REVENT_LIST) =
  if length sisl > 0 then
    let val sis : SI_SEED = hd sisl
        (* REVEVENTs with matching CERTHASH *)
        val matchingrevos = List.filter(fn(i) =>
            #2 i = #1 sis) revl
    in
      (* normal end date of validity not yet reached *)
      if #4 sis > date then
        (* check if revocation event is available
        ending the validity of the respective certificate*)
        if length matchingrevos > 0 then
          if #1(hd matchingrevos) <= date then
            (* at least one revocation event exists for the
            specified certificate --> don't generate SI*)
            cleanSiSeedList(tl sisl, date, revl)
          else (* revocation event exists but is not yet valid *)
            sis::cleanSiSeedList(tl sisl, date, revl)
        else (* no revo event exists *)
          sis::cleanSiSeedList(tl sisl, date, revl)
      else (* end of validity reached --> don't generate SI *)
        cleanSiSeedList(tl sisl, date, revl)
    end
  else [];

(* remove elements from list of si seeds that are
not yet valid at the current date *)
```

```
fun cleanSiSeedListNotYetValid(sisl: SI_SEED_LIST,
date : DATE) : SI_SEED_LIST =
if List.length sisl > 0 then
  let val sis : SI_SEED = hd sisl
  in
    (* include only SI_SEEDs that belong to certs already valid *)
    if #3 sis >= date then
      sis::cleanSiSeedListNotYetValid(tl sisl, date)
    else cleanSiSeedListNotYetValid(tl sisl, date)
  end
else [];
```

B.2.4 Generator-Methoden

```
(* generate CERT_INFO (entity + attribute type)
for the issuer certificate of cert *)
fun generateIssuerCertInfo(cert:CERT):CERT_INFO =
  (#7(#1(cert)), #2(#1(cert)))

(* generate CERT_INFO for the current certificate *)
fun generateCertInfo(cert:CERT):CERT_INFO =
  (#1(#1 cert), #2(#1 cert))

(* generate CERT_INFO for the issuer of cert requesting
a public key certificate*)
fun generateKeyCertInfo(cert:CERT):CERT_INFO =
  (#7(#1(cert)), "pk");

(* generate a certificate *)
fun generateCert(cert_detail : CERT_DETAIL,
ee_detail : EE_DETAIL, ia_detail:IA_DETAIL,date: DATE):CERT =
  if (#2 ee_detail = #1 cert_detail) then
    let val content : CERTCONTENT = (#1 ee_detail, #2 ee_detail,
        #3 ee_detail,date,date+(#2 cert_detail)-1,
        H_M_I(#3 cert_detail, #2 cert_detail), #1 ia_detail)
    in (content, sign(content, #2(#2 ia_detail)), #4 cert_detail)
    end
  else no_cert

(* generate an SI_SEED from details on certificate, end entity and issuer *)
fun generateSiSeed(cert_detail : CERT_DETAIL, ee_detail : EE_DETAIL,
ia_detail : IA_DETAIL, date: DATE) : SI_SEED =
  if (#2 ee_detail = #1 cert_detail) then
    (hashCert(generateCert(cert_detail, ee_detail, ia_detail,date)),
     #3 cert_detail, date, date+(#2 cert_detail)-1)
  else no_si_seed;

(* merge two lists of SI_SEEDs *)
fun generateFusionListSiSeed(sl1 : SI_SEED_LIST, sl2 : SI_SEED_LIST):
SI_SEED_LIST =
  if List.length sl1 = 0 then sl2
  else
```

```
    if List.length sl2 = 0 then sl1
    else
      if not (List.exists(fn(i) => i = (hd sl1)) sl2) then
        generateFusionListSiSeed(tl sl1, (hd sl1)::sl2)
      else generateFusionListSiSeed(tl sl1, sl2);

(* generate SI_INFO (certificate hash, current period i) *)
fun generateSiInfo(cert:CERT, date:DATE):SI_INFO =
  (hashCert(cert),getCurrentPeriod(cert,date));

(* generate status information tokens for current period *)
fun generateCurrentSis(sisl: SI_SEED_LIST, date: DATE):
SI_WITH_INFO_LIST =
  if length sisl > 0 then
    let val sis : SI_SEED = hd sisl
        val si : SI = H_M_I(#2 sis,(#4 sis - #3 sis + 1) -
                      (date- #3 sis + 1))
        val si_info : SI_INFO = (#1 sis, date-(#3 sis) +1)
        val si_with_info : SI_WITH_INFO = (si_info, si)
    in
      si_with_info::generateCurrentSis(tl sisl, date)
    end
  else [];
```

B.2.5 Methoden zum Filtern und Extrahieren von Daten

```
(* extract a public key from a public key certificate*)
fun extractPubKey(keycert:CERT):PUBKEY =
  (#3(#1 keycert), #1(#1 keycert))

(* extract the public key of a trusted authority
(issuer of cert) from a TA list *)
fun extractPubKeyTa(tal: CERT_LIST, cert: CERT) : PUBKEY =
  let val ta_match_list = List.filter(fn(i) =>  extractPubKey(i) =
              #2(#2 cert)) tal
  in
    if List.length(ta_match_list) > 0 then
      (* since a maximum of one certificate per TA can be stored,
         it is enough to use the list head *)
      extractPubKey(hd ta_match_list)
    else no_pubkey
  end;

(* extract a public key from a list of already verified certificates *)
fun extractPubKeyViCert(vcl : CERT_LIST, cert : CERT) : PUBKEY =
  let val vc_match_list = List.filter(fn(i) =>  extractPubKey(i) =
              #2(#2 cert)) vcl
  in
    if List.length(vc_match_list) > 0 then
      (* since a maximum of one certificate per holder can be stored,
         it is enough to use the list head *)
      extractPubKey(hd vc_match_list)
```

```
    else no_pubkey
  end;

(* extract AUT_DATA (entity + verified privilege) from
a privilege certificate chain *)
fun extractAutData(cl : CERT_LIST): AUT_DATA =
  let val lastcert = List.last(cl)
      val aut_data = (#1(#1 lastcert),#2(#1 lastcert))
  in
    aut_data
  end;

(* return privilege certificate from a list of issuer certificates
where the privilege type is the same and the delegation level
bigger than that attested in cert; note that size of icerts
is never bigger than 2 *)
fun filterPrivCerts(cert: CERT, icerts: CERT_LIST, rm: REQ_METHOD,
      dec_error_occured : E_LIST): CERT =
  if (rm = getFirst andalso List.length(icerts) = 1
     orelse rm = getAll andalso List.length(icerts) = 1)
     andalso dec_error_occured <> []
     andalso #2(#1 cert) = #2(#1 (hd icerts))
  then
    (* delegation level in first icert is bigger than that in cert *)
    if checkDelegLevel(cert, hd icerts) = true
    then (hd icerts)
    else no_cert
  else
    if rm = getAll andalso List.length(icerts) = 2
       andalso #2(#1 cert) = #2(#1 (hd(tl icerts)))
    then
      if checkDelegLevel(cert, hd(tl icerts))
      (* delegation level in second icert is bigger than that in cert *)
      then (hd(tl icerts))
      else no_cert
    else no_cert
```

B.2.6 Boolean-Methoden

```
(* returns if status infos for date have already
been published *)
fun datePublished(date: DATE, dl: DATE_LIST):BOOL =
  List.exists(fn(i) => i = date) dl;

(* returns if a cert is a public key cert *)
fun isKeyCert(cert: CERT) : bool =
  if (#3 cert, #2 (#1 cert)) = (a, "pk")
  then true else false;


(* returns if a cert is a privilege cert *)
fun isPrivCert(cert: CERT) : bool =
```

```
  if (#3 cert = p)
  then true else false;

(* returns if a cert signature can be verified with the
given public key *)
fun checkSignaturePubKey(vcert:CERT, i_pubkey:PUBKEY):BOOL =
  let val sign = #2 vcert
      val needed_pubkey = #2 sign
      val cert_hash = #1 sign
  in
    if (needed_pubkey = i_pubkey andalso
        cert_hash = hashCertcontent(#1 vcert))
    then true else false
  end

(* returns if a cert signature can be verified
with the public key attested in the issuer certificate *)
fun checkSignature(vcert:CERT, icert: CERT):BOOL =
  (* icert: issuer certificate, vcert: cert to verify*)
  checkSignaturePubKey(vcert, extractPubKey(icert));

(* returns if siwil contains a valid status info for cert *)
fun checkForValidSi(cert :CERT, siwil: SI_WITH_INFO_LIST,
date:DATE):BOOL =
  if List.exists(fn(i) => (H_M(#2 i, getCurrentPeriod(cert, date)))
     = (#6(#1 cert))) siwil
  then true else false

(* returns if a list of certificates contains the specified cert *)
fun containsCert(cert, cert_list): BOOL =
  List.exists(fn(i) => i = cert) cert_list

(* returns if the issuer of cert is a registered trust anchor *)
fun issuerIsTrustAnchor(cert:CERT, tal: CERT_LIST):BOOL =
  let val issuer = #7(#1 cert)
  in
    if List.exists(fn(i) => #1(#1 i) = issuer) tal
    then true else false
  end

(* returns if the issuer of cert is a registered source of authority *)
fun issuerIsSoA(cert:CERT, sal: CERT_LIST):BOOL =
   let val issuer = #7(#1 cert)
   in
     if List.exists(fn(i) => #1(#1 i) = issuer) sal
     then true else false
   end



(* returns if there is a certificate for the issuer of cert in the
list of already verified certs *)
```

```
fun existsVerifiedIssuerCert(cert: CERT, ver_cert_list: CERT_LIST):BOOL =
  List.exists(fn(i) => extractPubKey(i) = #2(#2 cert)) ver_cert_list

(* returns if one item of certs is a verified cert *)
fun isOneVerifiedCert(certs:CERT_LIST, vcerts: CERT_LIST):BOOL =
  if List.length(certs) > 0 then
    if List.exists(fn(i) => i = (hd certs)) vcerts
    then true else isOneVerifiedCert(tl certs, vcerts)
  else false;

(* returns if the second cert of cert_list can be
validated using the issuer cert (first cert of cert_list) *)
fun validateCurrentSignature(cert_list: CERT_LIST):BOOL =
  if List.length(cert_list) > 1 then
    let val issuer_cert = hd cert_list
        val cert_to_validate = hd(tl cert_list)
    in checkSignature(cert_to_validate, issuer_cert)
    end
  else false;

(* returns if icert has a delegation level bigger than the
deleg level attested in cert *)
fun checkDelegLevel(cert: CERT, icert:CERT) : BOOL =
  let val ds_opt = (Int.fromString(#3(#1 cert)))
      val ids_opt = (Int.fromString(#3(#1 icert)))
  in
    if Option.valOf(ids_opt) > Option.valOf(ds_opt)
       andalso #2(#1 icert) = #2(#1 cert)
       andalso #7(#1 cert)  = #1(#1 icert)
    then true
    else false
  end;
```

B.2.7 Andere Methoden

```
(* returns the trust anchor being the issuer of cert *)
fun getTrustAnchor(cert:CERT, tal: CERT_LIST) : CERT =
  let val issuer = #7(#1 cert)
      val ta = List.filter(fn(i) => #1(#1 i) = issuer) tal
      in
        (* trust anchor list may contain a maximum of
        one cert for a specific issuer *)
        if List.length(ta) > 0 then (hd ta)
        else no_cert
      end;

(* returns the source of authority being the issuer of cert *)
fun getSoA(cert:CERT, soal: CERT_LIST) : CERT =
  let val issuer = #7(#1 cert)
      val soa = List.filter(fn(i) => #1(#1 i) = issuer) soal
  in
    if List.length soa = 1 then hd soa
```

```
    else no_cert
  end

(* returns the first key cert in a list of certificates *)
fun findFirstKeyCert(clist : CERT_LIST): CERT =
  if clist <> []
  then
    let val cert = hd clist
    in
      if isKeyCert(cert)
      then cert
      else findFirstKeyCert(tl clist)
    end
  else no_cert

(* returns the first privilege cert matching a certain entity *)
fun findMatchingPrivCert (clist : CERT_LIST, en : ENTITY) : CERT =
  if clist <> [] then
    let val cert = hd clist
    in
      if (#1(#1 cert)) = en
         andalso #3 cert = p
      then cert
      else findMatchingPrivCert(tl clist,en)
    end
  else no_cert

(* extracts the first key certificate and - if available -
a matching privilege certificate from a list of certificates *)
fun extractConcreteCerts(clist: CERT_LIST): CERT_LIST =
  if clist <> []
  then
    let val cert = findFirstKeyCert(clist)
    in
      if cert <> no_cert
      then
        let val pcert = findMatchingPrivCert(clist,#1(#1 cert))
        in
          if pcert <> no_cert
          then [cert,pcert]
          else [cert]
        end
      else []
    end
  else [];
```

B.3 ZuSI-Methoden

```
(* create a certificate file ID from a CERT_INFO token *)
fun generateCertInfoFid(ci:CERT_INFO):FID =
  "H("^(#1 ci)^"+"^(#2 ci)^")";

(* create a certificate file ID from a CERT token *)
fun generateCertFid(c:CERT):FID =
  "H("^(#1 (#1 c))^"+"^(#2 (#1 c)^")");

(* create a status info file ID from an SI_INFO token *)
fun generateSiFid(si_info:SI_INFO):FID =
  "H("^(#1 si_info)^"+"^PERIOD.mkstr(#2 si_info)^")";

(* generate a symmetric key for certificate encryption *)
fun generateSymKeyFromCert(c:CERT):SYMKEY =
  (#1(#1 c), #2(#1 c), symkey);

(* generate a symmetric key for certificate encryption from
entity and attribute type *)
fun generateSymKeyFromEntity(ent:ENTITY,at:ATT_TYPE)=(ent,at,symkey);

(* encrypt a certificate *)
fun encryptCert(c: CERT):ZuSI_CERT =
  (c,generateSymKeyFromCert(c),enc)

(* encrypt a list of certificates *)
fun encryptCerts(cl: CERT_LIST):ZuSI_CERT_WITH_INFO_LIST =
  if cl <> [] then
    let val cert = hd cl
        val cert_info = (#1(#1 cert),#2(#1 cert))
    in
      (cert_info,encryptCert(cert))::encryptCerts(tl cl)
    end
  else empty

(* decrpyt a certificate with known entity and attribute type *)
fun decryptCertFromEntity(z: ZuSI_CERT, en:ENTITY, at: ATT_TYPE):CERT =
  if (generateSymKeyFromEntity(en,at) = #2 z andalso #3 z = enc)
  then #1 z
  else no_cert;

(* decrypt a list of certificates with known entity and attribute type *)
fun decryptCertsFromEntity(zl: ZuSI_CERT_LIST, en:ENTITY,at:ATT_TYPE):
CERT_LIST =
  if zl <> []
  then
    if decryptCertFromEntity(hd zl, en, at) <> no_cert
    then decryptCertFromEntity(hd zl, en,at )::
      decryptCertsFromEntity(tl zl, en,at)
    else decryptCertsFromEntity(tl zl,en,at)
  else [];
```

B.4 Peer-to-Peer-Methoden

```
(* find out if list of P2P_DATA (each element consists of file ID + content)
contains a P2P_DATA element with fid *)
fun containsFid(list:P2P_DATA_LIST, fid:FID):BOOL =
List.exists(fn(i) => (#1 i) = fid) list;

(* filter all P2P_DATA elements from list with fid match *)
fun getAllDataWithFid(list:P2P_DATA_LIST, fid:FID):P2P_DATA_LIST =
let val f = List.filter(fn(i) => (#1 i) = fid) list
in
  if List.length(f) > 0
  then f
  else []
end;

(* filter first P2P_DATA element from list with matching fid;
this is a simplification of the real process where even with
getFirst multiple data elements with same fid can be returned *)
fun getFirstDataWithFid(list:P2P_DATA_LIST, fid:FID):
P2P_DATA_LIST =
let val f = List.find(fn(i) => (#1 i) = fid) list
in
  case f of
    SOME(f) => [f]
    | NONE => []
end;

(* convert P2P_DATA list to list of certificates *)
fun extractCerts(dlist: P2P_DATA_LIST):ZuSI_CERT_LIST=
if List.length(dlist) > 0 then
  let val d = #2(hd dlist)
  in
    case d of
      Cert(cert) => cert::extractCerts(tl dlist)
      | _ => extractCerts(tl dlist)
  end
else [];

(* convert P2P_DATA list to list of status information *)
fun extractSis(dlist: P2P_DATA_LIST,
slist: SI_INFO_LIST):ZuSI_SI_LIST=
if List.length(dlist) > 0 then
  let val d = #2(hd dlist)
      val sinfo = hd slist
  in
    case d of
      Si(si) => (sinfo,si)::extractSis(tl dlist, slist)
      | _ => extractSis(tl dlist,slist)
  end
else [];
```

Anhang C

Methoden und Initial Markings der HCPN-basierten Sicherheitsanalyse

C.1 Trust Anchors, SoAs und Konstanten

Diese Definitionen werden in allen Szenarien verwendet. Als Initialbelegungen werden `sec_ta_list`, `sec_soa_list` und `cur_date_sec` für die Stellen *V_TrustAnchors*, *V_SoAs* und *GlobalDate* gesetzt.

```
val key_cert_ta1 : CERT =
  let val content = ("TA1", "pk", "publ-TA1", 1, 100, H_M_I(0,100), "V")
  in (content,(hashCertcontent(content),("publ-V", "V"),s), p) end

val key_cert_ta2 : CERT =
  let val content = ("TA2", "pk", "publ-TA2", 1, 100, H_M_I(0,100), "V")
  in (content,(hashCertcontent(content),("publ-V", "V"),s), p) end

val key_cert_soa1 : CERT =
  let val content = ("SoA1", "pk", "publ-SoA1", 1, 100, H_M_I(0,100), "V")
  in (content,(hashCertcontent(content),("publ-V", "V"),s), p) end

val priv_cert_soa1 : CERT =
  let val content = ("SoA1", "Privilege1", "unlimited", 1, 100, H_M_I(1,100), "V")
  in (content,(hashCertcontent(content),("publ-V", "V"),s), p) end

(* initial marking for TopLevelModel'V_TrustAnchors *)
val sec_ta_list : CERT_LIST =
  [key_cert_ta1,key_cert_ta2,key_cert_ia2,key_cert_soa1]

(* initial marking for TopLevelModel'V_SoAs *)
val sec_soa_list  =[priv_cert_soa1]

(* initial marking for TopLevelModel'GlobalDate *)
val cur_date_sec = 3
```

C.2 Fall 1 : Vertraulichkeit

C.2.1 Methoden

```
(* decrypts encrypted Certificates using attribute type and a
list of possible holders *)
fun decryptCertFromElist(c: ZuSI_CERT, elist:ENTITY_LIST, at: ATT_TYPE):CERT =
  if elist <> []
  then
    let val en = hd elist
    in
      if generateSymKeyFromEntity(en,at) = #2 c andalso #3 c = enc
      then #1 c
      else decryptCertFromElist(c,tl elist, at)
    end
  else no_cert;

(* decrypts encrypted Certificates using a list of attribute types and a
list of possible holders *)
fun decryptCertFromLists(c: ZuSI_CERT, elist:ENTITY_LIST,
    alist: ATT_TYPE_LIST):CERT =
  if alist <> []
  then
    let val at = hd alist
        val cert_result = decryptCertFromElist(c,elist,at)
    in
      if cert_result <> no_cert
      then cert_result
      else decryptCertFromLists(c,elist,tl alist)
    end
  else no_cert;

(* extracts all Sis from a list of P2P_DATA *)
fun extractSisAttack(dlist: P2P_DATA_LIST): P2P_DATA_LIST =
  if List.length(dlist) > 0
  then
    let val d = hd dlist
        val data = #2 d
    in
      case data of
        Si(si) => (d)::extractSisAttack(tl dlist)
           | _ => extractSisAttack(tl dlist)
    end
  else [];


(* tries to find out the correct (certhash,period)-pair for an SI
using one certificate to hash, going through all possible past periods *)
fun tryAllFids(p2p_si : P2P_DATA, cert: CERT, i : PERIOD): SI_WITH_INFO =
if i > 0
then
  let val certhash = hashCert(cert)
```

```
      val fid = #1 p2p_si
      val si = #2 p2p_si
  in
    if fid = generateSiFid(certhash,i)
    then
      case si of
        Si(sii) => ((certhash,i),sii)
            | _ => tryAllFids(p2p_si,cert,i-1)
      else tryAllFids(p2p_si,cert,i-1)
    end
  else no_si_with_info;

(* tries to find out the correct (certhash,period)-pair for an SI
going through a list of certificates and all possible past periods *)
fun tryFidMatch(p2p_si: P2P_DATA, clist : CERT_LIST, date: DATE):
    SI_WITH_INFO =
  if List.length(clist) > 0
  then
    let val cert = hd clist
        val i = getCurrentPeriod(cert,date)
        val match = tryAllFids(p2p_si,cert,i)
    in
      if match = no_si_with_info
      then tryFidMatch(p2p_si, tl clist, date)
      else match
    end
  else (* return dummy data *) no_si_with_info;
```

C.2.2 Initial Marking

Dummy-Werte

```
val no_zusi_si = (("no_hash",0), no_si)
val no_si_with_info = (("no_hash",0),no_si)
```

Zertifikate und Statusinformationen

```
val key_cert_1 : CERT =
  let val content = ("E", "pk", "publ-E",2,4, H_M_I(400,3), "IA1")
  in (content,(hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val key_cert_2 : CERT =
  let val content = ("A", "pk", "publ-A",2,4, H_M_I(401,3), "IA1")
  in (content,(hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val key_cert_3 : CERT =
  let val content = ("IA1", "pk", "publ-IA1",2,4, H_M_I(402,3), "TA1")
  in (content,(hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end
```

```
val priv_cert_1 : CERT =
  let val content = ("E", "Privilege3", "12",2,4, H_M_I(403,3), "IA1")
  in (content,(hashCertcontent(content),("publ-IA1", "IA1"),s), p)
  end

val priv_cert_2 : CERT =
  let val content = ("IA2", "Privilege2", "3",2,4, H_M_I(404,3), "IA1")
  in (content,(hashCertcontent(content),("publ-IA1", "IA1"),s), p)
  end

val priv_cert_3 : CERT =
  let val content = ("IA2", "Privilege1", "3",2,4, H_M_I(405,3), "SoA1")
  in (content,(hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end

val zusi_key_cert_1 : P2P_DATA =
  (generateCertFid(key_cert_1),Cert(encryptCert(key_cert_1)))

val zusi_key_cert_2 : P2P_DATA =
  (generateCertFid(key_cert_2),Cert(encryptCert(key_cert_2)))

val zusi_key_cert_3 : P2P_DATA =
  (generateCertFid(key_cert_3),Cert(encryptCert(key_cert_3)))

val zusi_priv_cert_1 : P2P_DATA =
  (generateCertFid(priv_cert_1),Cert(encryptCert(priv_cert_1)))

val zusi_priv_cert_2 : P2P_DATA =
  (generateCertFid(priv_cert_2),Cert(encryptCert(priv_cert_2)))

val zusi_priv_cert_3 : P2P_DATA =
  (generateCertFid(priv_cert_3),Cert(encryptCert(priv_cert_3)))

val zusi_si_key_cert_1 :P2P_DATA =
  (generateSiFid(hashCert(key_cert_1),2),Si(H_M_I(400,1)))

val zusi_si_priv_cert_2 :P2P_DATA =
  (generateSiFid(hashCert(priv_cert_2),2),Si(H_M_I(404,1)))

(* initial marking for TopLevelModel'SN_DataStore *)
val sec_1_data_store =
  [zusi_key_cert_1,zusi_key_cert_2,zusi_si_key_cert_1,zusi_priv_cert_1,
   zusi_key_cert_3,zusi_si_priv_cert_2,zusi_priv_cert_2,zusi_priv_cert_3]
```

Daten des Angreifers

```
(* initial marking for TopLevelModel'SN_KnownTypes *)
val known_types : ATT_TYPE_LIST = ["pk","Privilege0", "Privilege1"]

(* initial marking for TopLevelModel'SN_KnownEntities *)
val known_entities : ENTITY_LIST = ["E","F","G","IA1","IA2","IA3",
                                    "TA1","TA2","TA3","SoA1","SoA2", "V"]
```

```
(* initial marking for TopLevelModel'SN_KnownCerts *)
val known_certificates : CERT_LIST = [key_cert_1]
```

C.3 Fall 2 : Integrität

C.3.1 Methoden der Zustandsraumanalyse

```
(* number of all states *)
SearchNodes(EntireGraph,
fn _ => true,
NoLimit,fn _ => 1,
0,op +)

(* illegal states 2.1.a.i,2.1.b.i,2.2.c.i, 2.2.d.i :
   authentication success states *)
SearchAllNodes(
fn n => (Mark.processAA'V_AuthSuccessful 1 n <> empty),
fn n => 1,
0,op +)

(* illegal states 2.1.a.ii, 2.1.b.ii, 2.2.c.ii, 2.2.d.ii :
   A&A success states *)
SearchNodes(EntireGraph,
fn n => (Mark.TopLevelModel'V_AASuccess 1 n <> empty),
NoLimit,
fn _ => 1,
0,
op +)

(* illegal states 2.1.c.i, 2.1.c.ii, 2.2.a.i, 2.2.a.ii, 2.2.b.i, 2.2.b.ii :
   NACK states *)
SearchNodes(EntireGraph,
fn n => (Mark.TopLevelModel'V_AANack 1 n <> 1`[]),
NoLimit,
fn _ => 1,
0,
op +)
```

C.3.2 Allgemeine Initial Markings für *EE_KeyPair* und *EE_Init*

```
(* private and public keys for E *)
val privkey_e : PRIVKEY = ("priv-E", "E");
val pubkey_e : PUBKEY = ("publ-E", "E");

(* initial marking for TopLevelModel'EE_KeyPair for all cases *)
val keyp_e : KEYPAIR = (privkey_e, pubkey_e);

(* initial marking for TopLevelModel'EE_Init for each subcase i except for 2.c.i *)
val init_i: ENTITY_PRIV_LIST = [("E","none")]
```

```
(* initial marking for TopLevelModel'EE_Init for subcase 2.c.i *)
val init_i: ENTITY_PRIV_LIST = [("E2","none")]

(* initial marking for TopLevelModel'EE_Init for each subcase ii *)
val init_ii: ENTITY_PRIV_LIST = [("E","Privilege1")]
```

C.3.3 Initial Markings Fall 2.1

Die folgenden Definitionen umfassen ausschließlich Zertifikate und Statusinformationen. Sie werden jeweils als Initialbelegung der Stelle *S_DataStore* gesetzt. In jedem Subcase wird zudem `init_i` oder `init_ii` (siehe C.3.2) für *EE_Init* gesetzt, um den A&A-Prozess anzustoßen.

C.3.3.1 Fall 2.1.a. Alte Statusinformation als aktuell ausgeben (Ziel: Gültigkeit vorspiegeln)

Fall 2.1.a.i

```
(* initial key cert issued by trust anchor *)
val sec_2_1ai_init_key_cert : CERT =
  let val content = ("E", "pk", "publ-E",2,4,H_M_I(31,3), "TA1")
  in (content,(hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_1ai_zusi_init_key_cert =
  (generateCertFid(sec_2_1ai_init_key_cert),
  Cert(encryptCert(sec_2_1ai_init_key_cert)))

(* attack: si from period 1 with new fid *)
val sec_2_1ai_zusi_si_init_key_cert : P2P_DATA =
 (generateSiFid(hashCert(sec_2_1ai_init_key_cert),2),Si(H_M_I(31,3)))

(* initial marking for TopLevelModel'SN_DataStore *)
val sec_2_1ai_store : P2P_DATA_LIST =
  [sec_2_1ai_zusi_init_key_cert,sec_2_1ai_zusi_si_init_key_cert]
```

Fall 2.1.a.ii

```
val sec_2_1aii_init_key_cert = sec_2_1ai_init_key_cert

val sec_2_1aii_zusi_init_key_cert =
  (generateCertFid(sec_2_1aii_init_key_cert),
   Cert(encryptCert(sec_2_1aii_init_key_cert)))

(* valid si for initial key cert *)
val sec_2_1aii_zusi_si_init_key_cert =
  (generateSiFid(hashCert(sec_2_1aii_init_key_cert),2),Si(H_M_I(31,1)))

val sec_2_1aii_init_priv_cert : CERT =
  let val content = ("E", "Privilege1", "2",2,4,H_M_I(32,3), "SoA1")
  in (content,(hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end
```

```
val sec_2_1aii_zusi_init_priv_cert =
  (generateCertFid(sec_2_1aii_init_priv_cert),
   Cert(encryptCert(sec_2_1aii_init_priv_cert)))

(* si from period 1 with new fid *)
val sec_2_1aii_zusi_si_init_priv_cert: P2P_DATA =
  (generateSiFid(hashCert(sec_2_1aii_init_priv_cert),2),Si(H_M_I(32,3)))

(* initial marking for TopLevelModel'SN_DataStore *)
val sec_2_1aii_store : P2P_DATA_LIST =
  [sec_2_1aii_zusi_init_key_cert,sec_2_1aii_zusi_si_init_key_cert,
   sec_2_1aii_zusi_init_priv_cert,sec_2_1aii_zusi_si_init_priv_cert]
```

C.3.3.2 Fall 2.1.b: Zufällige Statusinformation als aktuell ausgeben (Ziel: Gültigkeit vorspiegeln)

Fall 2.1.b.i

```
val sec_2_2bi_init_key_cert = sec_2_1ai_init_key_cert

val sec_2_1bi_zusi_init_key_cert = sec_2_1ai_zusi_init_key_cert

val sec_2_1bi_random_value = 4321

(* random si with current fid *)
val sec_2_1bi_zusi_si_init_key_cert: P2P_DATA =
  (generateSiFid(hashCert(sec_2_1bi_init_key_cert),2),
   Si(H_M_I(sec_2_1bi_random_value,1)))

(* initial marking for TopLevelModel'SN_DataStore *)
val sec_2_1bi_store : P2P_DATA_LIST =
  [sec_2_1bi_zusi_init_key_cert,sec_2_1bi_zusi_si_init_key_cert]
```

Fall 2.1.b.ii

```
val sec_2_1bii_init_key_cert = sec_2_1aii_init_key_cert
val sec_2_1bii_zusi_init_key_cert = sec_2_1aii_zusi_init_key_cert
val sec_2_1bii_zusi_si_init_key_cert = sec_2_1aii_zusi_si_init_key_cert
val sec_2_1bii_init_priv_cert = sec_2_1aii_init_priv_cert
val sec_2_1bii_zusi_init_priv_cert = sec_2_1aii_zusi_init_priv_cert

val sec_2_1bii_random_value = 5432

(* random si with current fid *)
val sec_2_1bii_zusi_si_init_priv_cert: P2P_DATA =
  (generateSiFid(hashCert(sec_2_1bii_init_priv_cert),2),
   Si(H_M_I(sec_2_1bii_random_value,1)))

(* initial marking for place TopLevelModel'SN_DataStore *)
  val sec_2_1bii_store : P2P_DATA_LIST =
    [sec_2_1bii_zusi_init_key_cert,sec_2_1bii_zusi_si_init_key_cert,
     sec_2_1bii_zusi_init_priv_cert,sec_2_1bii_zusi_si_init_priv_cert]
```

C.3.3.3 2.1.c: Zufällige Statusinformation als aktuell ausgeben (Ziel: Ungültigkeit vorspiegeln)

Fall 2.1.c.i

```
val sec_2_1ci_init_key_cert = sec_2_1aii_init_key_cert
val sec_2_1ci_zusi_init_key_cert = sec_2_1aii_zusi_init_key_cert

(* random si for key cert *)
val sec_2_1ci_zusi_si_init_key_cert_modification = sec_2_1bi_zusi_si_init_key_cert

(* valid si for key cert *)
val sec_2_1ci_zusi_si_init_key_cert_correct =
  (generateSiFid(hashCert(sec_2_1ci_init_key_cert),2),Si(H_M_I(31,1)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_1ci_store : P2P_DATA_LIST =
  [sec_2_1ci_zusi_init_key_cert,sec_2_1ci_zusi_si_init_key_cert_modification,
   sec_2_1ci_zusi_si_init_key_cert_correct]
```

Fall 2.1.c.ii

```
val sec_2_1cii_init_priv_cert = sec_2_1bii_init_priv_cert
val sec_2_1cii_zusi_si_init_priv_cert_modification = sec_2_1bii_zusi_si_init_priv_cert
val sec_2_1cii_zusi_init_key_cert = sec_2_1bii_zusi_init_key_cert
val sec_2_1cii_zusi_si_init_key_cert = sec_2_1bii_zusi_si_init_key_cert
val sec_2_1cii_zusi_init_priv_cert = sec_2_1bii_zusi_init_priv_cert

(* correct si for init_priv_cert *)
val sec_2_1cii_zusi_si_init_priv_cert_correct: P2P_DATA =
  (generateSiFid(hashCert(sec_2_1cii_init_priv_cert),2),Si(H_M_I(32,1)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_1cii_store : P2P_DATA_LIST =
  [sec_2_1cii_zusi_init_key_cert,sec_2_1cii_zusi_si_init_key_cert,
   sec_2_1cii_zusi_init_priv_cert,
   sec_2_1cii_zusi_si_init_priv_cert_modification,
   sec_2_1cii_zusi_si_init_priv_cert_correct]
```

C.3.4 Initial Markings Fall 2.2

C.3.4.1 Fall 2.2.a. Zerstörung des Chiffrats eines Zertifikats

Fall 2.2.a.i

```
(* wrongly encrypts a certificate)
fun encryptCertError(c: CERT):ZuSI_CERT = (c,generateSymKeyFromCert(c),error)

val sec_2_2ai_si_seed_init_key_cert = 100

val sec_2_2ai_init_key_cert : CERT =
  let val content = ("E","pk","publ-E",2,4,
            H_M_I(sec_2_2ai_si_seed_init_key_cert,3),"IA1")
```

```
  in (content, hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val sec_2_2ai_zusi_init_key_cert =
  (generateCertFid(sec_2_2ai_init_key_cert),
   Cert(encryptCert(sec_2_2ai_init_key_cert)))

(* initial key cert with modified cipher text *)
val sec_2_2ai_zusi_init_key_cert_modification =
  (generateCertFid(sec_2_2ai_init_key_cert),
   Cert(encryptCertError(sec_2_2ai_init_key_cert)))

val sec_2_2ai_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2ai_init_key_cert),2),
   Si(H_M_I(sec_2_2ai_si_seed_init_key_cert,1)))

val sec_2_2ai_si_seed_issuer_key_cert = 101

val sec_2_2ai_issuer_key_cert : CERT =
  let val content = ("IA1", "pk", "publ-IA1",1,4,
            H_M_I(sec_2_2ai_si_seed_issuer_key_cert,4), "TA1")
  in (content,(hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2ai_zusi_issuer_key_cert =
  (generateCertFid(sec_2_2ai_issuer_key_cert),
   Cert(encryptCert(sec_2_2ai_issuer_key_cert)))

val sec_2_2ai_zusi_si_issuer_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2ai_issuer_key_cert),3),
   Si(H_M_I(sec_2_2ai_si_seed_issuer_key_cert,1)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2ai_store : P2P_DATA_LIST =
  [sec_2_2ai_zusi_init_key_cert_modification,sec_2_2ai_zusi_init_key_cert,
   sec_2_2ai_zusi_si_init_key_cert,sec_2_2ai_zusi_issuer_key_cert,
   sec_2_2ai_zusi_si_issuer_key_cert]
```

Fall 2.2.a.ii

```
val sec_2_2aii_si_seed_init_key_cert = 110
val sec_2_2aii_si_seed_init_priv_cert = 111
val sec_2_2aii_si_seed_issuer_priv_cert = 112

val sec_2_2aii_init_key_cert : CERT =
  let val content = ("E", "pk", "publ-E", 2,4,
            H_M_I(sec_2_2aii_si_seed_init_key_cert,3), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2aii_zusi_init_key_cert : P2P_DATA =
  (generateCertFid(sec_2_2aii_init_key_cert),
```

```
   Cert(encryptCert(sec_2_2aii_init_key_cert)))

val sec_2_2aii_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2aii_init_key_cert),2),
   Si(H_M_I(sec_2_2aii_si_seed_init_key_cert,1)))

val sec_2_2aii_init_priv_cert : CERT =
  let val content = ("E", "Privilege1", "2",3,5,
            H_M_I(sec_2_2aii_si_seed_init_priv_cert,3), "IA2")
  in (content, (hashCertcontent(content),("publ-IA2", "IA2"),s), p)
  end

val sec_2_2aii_zusi_init_priv_cert =
  (generateCertFid(sec_2_2aii_init_priv_cert),
  Cert(encryptCert(sec_2_2aii_init_priv_cert)))

val sec_2_2aii_zusi_init_priv_cert_modification =
  (generateCertFid(sec_2_2aii_init_priv_cert),
   Cert(encryptCertError(sec_2_2aii_init_priv_cert)))

val sec_2_2aii_zusi_si_init_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2aii_init_priv_cert),1),
   Si(H_M_I(sec_2_2aii_si_seed_init_priv_cert,2)))

val sec_2_2aii_issuer_priv_cert : CERT =
  let val content = ("IA2", "Privilege1", "5", 1,5,
            H_M_I(sec_2_2aii_si_seed_issuer_priv_cert,5), "SoA1")
  in (content, (hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end

val sec_2_2aii_zusi_issuer_priv_cert =
  (generateCertFid(sec_2_2aii_issuer_priv_cert),
   Cert(encryptCert(sec_2_2aii_issuer_priv_cert)))

val sec_2_2aii_zusi_si_issuer_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2aii_issuer_priv_cert),3),
   Si(H_M_I(sec_2_2aii_si_seed_issuer_priv_cert,2)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2aii_store : P2P_DATA_LIST =
  [sec_2_2aii_zusi_init_key_cert,sec_2_2aii_zusi_si_init_key_cert,
   sec_2_2aii_zusi_init_priv_cert,sec_2_2aii_zusi_init_priv_cert_modification,
   sec_2_2aii_zusi_si_init_priv_cert,sec_2_2aii_zusi_issuer_priv_cert,
   sec_2_2aii_zusi_si_issuer_priv_cert]
```

C.3.4.2 Fall 2.2.b. Zerstörung des Klartextes eines Zertifikats

Fall 2.2.b.i

```
val sec_2_2bi_si_seed_init_key_cert = 200
val sec_2_2bi_si_seed_issuer_key_cert = 201
```

```
val sec_2_2bi_init_key_cert : CERT =
  let val content = ("E","pk","publ-E", 2,4,
            H_M_I(sec_2_2bi_si_seed_init_key_cert,3), "IA1")
  in (content, (hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val sec_2_2bi_init_key_cert_modification : CERT =
  let val content = ("E", "error", "publ-E-error", 2,4,
            H_M_I(sec_2_2bi_si_seed_init_key_cert,3), "IA1")
  in (content, (hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val sec_2_2bi_zusi_init_key_cert =
  (generateCertFid(sec_2_2bi_init_key_cert),
   Cert(encryptCert(sec_2_2bi_init_key_cert)));

(* modified initial key cert: plain text screwed up *)
val sec_2_2bi_zusi_init_key_cert_modification =
  (generateCertFid(sec_2_2bi_init_key_cert),
   Cert(encryptCert(sec_2_2bi_init_key_cert_modification)));

val sec_2_2bi_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2bi_init_key_cert),2),
   Si(H_M_I(sec_2_2bi_si_seed_init_key_cert,1)));

val sec_2_2bi_issuer_key_cert : CERT =
  let val content = ("IA1", "pk", "publ-IA1", 1,4,
            H_M_I(sec_2_2bi_si_seed_issuer_key_cert,4), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2bi_zusi_issuer_key_cert =
  (generateCertFid(sec_2_2bi_issuer_key_cert),
   Cert(encryptCert(sec_2_2bi_issuer_key_cert)));

val sec_2_2bi_zusi_si_issuer_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2bi_issuer_key_cert),3),
   Si(H_M_I(sec_2_2bi_si_seed_issuer_key_cert,1)));

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2bi_store : P2P_DATA_LIST =
  [sec_2_2bi_zusi_init_key_cert_modification,sec_2_2bi_zusi_init_key_cert,
   sec_2_2bi_zusi_si_init_key_cert,sec_2_2bi_zusi_issuer_key_cert,
   sec_2_2bi_zusi_si_issuer_key_cert]
```

Fall 2.2.b.ii

```
val sec_2_2bii_si_seed_init_key_cert = 210
val sec_2_2bii_si_seed_init_priv_cert = 211
val sec_2_2bii_si_seed_issuer_priv_cert = 212

val sec_2_2bii_init_key_cert : CERT =
```

```
  let val content = ("E", "pk", "publ-E", 2,4,
            H_M_I(sec_2_2bii_si_seed_init_key_cert,3), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2bii_zusi_init_key_cert =
  (generateCertFid(sec_2_2bii_init_key_cert),
   Cert(encryptCert(sec_2_2bii_init_key_cert)))

val sec_2_2bii_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2bii_init_key_cert),2),
   Si(H_M_I(sec_2_2bii_si_seed_init_key_cert,1)))

val sec_2_2bii_init_priv_cert : CERT =
  let val content = ("E", "Privilege1", "2", 3,5,
            H_M_I(sec_2_2bii_si_seed_init_priv_cert,3), "IA2")
  in (content, (hashCertcontent(content),("publ-IA2", "IA2"),s), p)
  end

(* modified initial privilege cert: plain text screwed up *)
val sec_2_2bii_init_priv_cert_modification : CERT =
  let val content = ("E", "Privilege1-error", "2-error", 1,5,
            H_M_I(sec_2_2bii_si_seed_init_priv_cert,3), "IA2")
  in (content, (hashCertcontent(content),("publ-IA2", "IA2"),s), p)
  end

val sec_2_2bii_zusi_init_priv_cert =
  (generateCertFid(sec_2_2bii_init_priv_cert),
   Cert(encryptCert(sec_2_2bii_init_priv_cert)))

val sec_2_2bii_zusi_init_priv_cert_modification =
  (generateCertFid(sec_2_2bii_init_priv_cert),
   Cert(encryptCert(sec_2_2bii_init_priv_cert_modification)))

val sec_2_2bii_zusi_si_init_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2bii_init_priv_cert),1),
   Si(H_M_I(sec_2_2bii_si_seed_init_priv_cert,2)))

val sec_2_2bii_issuer_priv_cert : CERT =
  let val content = ("IA2", "Privilege1", "5", 1,5,
            H_M_I(sec_2_2bii_si_seed_issuer_priv_cert,5), "SoA1")
  in (content, (hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end

val sec_2_2bii_zusi_issuer_priv_cert =
  (generateCertFid(sec_2_2bii_issuer_priv_cert),
   Cert(encryptCert(sec_2_2bii_issuer_priv_cert)))

val sec_2_2bii_zusi_si_issuer_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2bii_issuer_priv_cert),3),
   Si(H_M_I(sec_2_2bii_si_seed_issuer_priv_cert,2)));
```

```
(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2bii_store : P2P_DATA_LIST =
  [sec_2_2bii_zusi_init_key_cert,sec_2_2bii_zusi_si_init_key_cert,
  sec_2_2bii_zusi_init_priv_cert_modification,sec_2_2bii_zusi_init_priv_cert,
  sec_2_2bii_zusi_si_init_priv_cert,sec_2_2bii_zusi_issuer_priv_cert,
  sec_2_2bii_zusi_si_issuer_priv_cert]
```

C.3.4.3 Fall 2.2.c. Manipulation eines Zertifikats (Ziel: Betrug)

Fall 2.2.c.i

```
val sec_2_2ci_si_seed_init_key_cert = 120
val sec_2_2ci_si_seed_issuer_key_cert = 121

val sec_2_2ci_init_key_cert : CERT =
  let val content = ("E2", "pk", "publ-E2", 2,4,
            H_M_I(sec_2_2ci_si_seed_init_key_cert,3), "IA1")
  in (content, (hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

(* modified key cert: public key of E2 is replaced with public key of E *)
val sec_2_2ci_init_key_cert_modification : CERT =
  let val content = ("E2", "pk", "publ-E2", 2,4,
            H_M_I(sec_2_2ci_si_seed_init_key_cert,3), "IA1")
      val content_modification = ("E2", "pk", "publ-E", 2,4,
            H_M_I(sec_2_2ci_si_seed_init_key_cert,3), "IA1")
  in (content_modification, (hashCertcontent(content),("publ-IA1", "IA1"),s), a)
  end

val sec_2_2ci_zusi_init_key_cert =
  (generateCertFid(sec_2_2ci_init_key_cert),
   Cert(encryptCert(sec_2_2ci_init_key_cert)))

val sec_2_2ci_zusi_init_key_cert_modification=
  (generateCertFid(sec_2_2ci_init_key_cert_modification),
   Cert(encryptCert(sec_2_2ci_init_key_cert_modification)))

val sec_2_2ci_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2ci_init_key_cert),2),
   Si(H_M_I(sec_2_2ci_si_seed_init_key_cert,1)))

val sec_2_2ci_zusi_si_init_key_cert_modification : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2ci_init_key_cert_modification),2),
   Si(H_M_I(sec_2_2ci_si_seed_init_key_cert,1)))

val sec_2_2ci_issuer_key_cert : CERT =
  let val content = ("IA1", "pk", "publ-IA1", 1,4,
            H_M_I(sec_2_2ci_si_seed_issuer_key_cert,4), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2ci_zusi_issuer_key_cert =
```

```
  (generateCertFid(sec_2_2ci_issuer_key_cert),
   Cert(encryptCert(sec_2_2ci_issuer_key_cert)));

val sec_2_2ci_zusi_si_issuer_key_cert : P2P_DATA =
               (generateSiFid(hashCert(sec_2_2ci_issuer_key_cert),3),
               Si(H_M_I(sec_2_2ci_si_seed_issuer_key_cert,1)));

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2ci_store : P2P_DATA_LIST =
  [sec_2_2ci_zusi_init_key_cert_modification,sec_2_2ci_zusi_init_key_cert,
   sec_2_2ci_zusi_si_init_key_cert,sec_2_2ci_zusi_si_init_key_cert_modification,
   sec_2_2ci_zusi_issuer_key_cert,sec_2_2ci_zusi_si_issuer_key_cert]
```

Fall 2.2.c.ii

```
val sec_2_2cii_si_seed_init_key_cert = 130
val sec_2_2cii_si_seed_init_priv_cert = 131
val sec_2_2cii_si_seed_issuer_priv_cert = 132

val sec_2_2cii_init_key_cert : CERT =
  let val content = ("E", "pk", "publ-E", 2,4,
            H_M_I(sec_2_2cii_si_seed_init_key_cert,3), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2cii_zusi_init_key_cert =
  (generateCertFid(sec_2_2cii_init_key_cert),
   Cert(encryptCert(sec_2_2cii_init_key_cert)))

val sec_2_2cii_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2cii_init_key_cert),2),
               Si(H_M_I(sec_2_2cii_si_seed_init_key_cert,1)))


val sec_2_2cii_init_priv_cert_forE2 : CERT =
  let val content = ("E2", "Privilege1", "2",3,5,
            H_M_I(sec_2_2cii_si_seed_init_priv_cert,3), "IA2")
  in (content, (hashCertcontent(content),("publ-IA2", "IA2"),s), p)
  end

(* modified initial privilege cert: claim privilege of E2 for E *)
val sec_2_2cii_init_priv_cert_modification_forE : CERT =
  let val content = ("E2", "Privilege1", "2",3,5,
            H_M_I(sec_2_2cii_si_seed_init_priv_cert,3), "IA2")
      val content_modification = ("E", "Privilege1", "2",3,5,
            H_M_I(sec_2_2cii_si_seed_init_priv_cert,3), "IA2")
  in (content_modification, (hashCertcontent(content),("publ-IA2", "IA2"),s), p)
  end

val sec_2_2cii_zusi_init_priv_cert_forE2 =
  (generateCertFid(sec_2_2cii_init_priv_cert_forE2),
   Cert(encryptCert(sec_2_2cii_init_priv_cert_forE2)))
```

```
val sec_2_2cii_zusi_init_priv_cert_modification_forE =
  (generateCertFid(sec_2_2cii_init_priv_cert_modification_forE),
   Cert(encryptCert(sec_2_2cii_init_priv_cert_modification_forE)))

val sec_2_2cii_zusi_si_init_priv_cert_forE2 : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2cii_init_priv_cert_forE2),1),
   Si(H_M_I(sec_2_2cii_si_seed_init_priv_cert,2)))

(* valid si for the modified initial privilege cert *)
val sec_2_2cii_zusi_si_init_priv_cert_modification_forE : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2cii_init_priv_cert_forE2),1),
   Si(H_M_I(sec_2_2cii_si_seed_init_priv_cert,2)));

val sec_2_2cii_issuer_priv_cert : CERT =
  let val content = ("IA2", "Privilege1", "1", 1,5,
            H_M_I(sec_2_2cii_si_seed_issuer_priv_cert,5), "SoA1")
  in (content, (hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end

val sec_2_2cii_zusi_issuer_priv_cert =
  (generateCertFid(sec_2_2cii_issuer_priv_cert),
   Cert(encryptCert(sec_2_2cii_issuer_priv_cert)))

val sec_2_2cii_zusi_si_issuer_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2cii_issuer_priv_cert),3),
   Si(H_M_I(sec_2_2cii_si_seed_issuer_priv_cert,2)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2cii_store : P2P_DATA_LIST =
  [sec_2_2cii_zusi_init_key_cert,sec_2_2cii_zusi_si_init_key_cert,
   sec_2_2cii_zusi_init_priv_cert_modification_forE,
   sec_2_2cii_zusi_init_priv_cert_forE2,
   sec_2_2cii_zusi_si_init_priv_cert_forE2,
   sec_2_2cii_zusi_si_init_priv_cert_modification_forE,
   sec_2_2cii_zusi_issuer_priv_cert,sec_2_2cii_zusi_si_issuer_priv_cert]
```

C.3.4.4 Fall 2.2.d. Neuzusammenstellung eines Zertifikats (Ziel: Betrug)

Fall 2.2.d.i

```
val sec_2_2di_si_seed_init_key_cert = 200
val sec_2_2di_si_seed_issuer_key_cert = 201

(* newly assembled initial key cert with random signature *)
val sec_2_2di_init_key_cert_random : CERT =
  let val content = ("E", "pk", "publ-E",2,4,
            H_M_I(sec_2_2di_si_seed_init_key_cert,3), "IA1")
  in (content, (hashCertcontent(content),("publ-Rand", "Random"),s), a)
  end

val sec_2_2di_zusi_init_key_cert_random =
```

```
  (generateCertFid(sec_2_2di_init_key_cert_random),
   Cert(encryptCert(sec_2_2di_init_key_cert_random)))

val sec_2_2di_zusi_si_init_key_cert_random : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2di_init_key_cert_random),2),
   Si(H_M_I(sec_2_2di_si_seed_init_key_cert,1)))

val sec_2_2di_issuer_key_cert : CERT =
  let val content = ("IA1", "pk", "publ-IA1", 1,4,
            H_M_I(sec_2_2di_si_seed_issuer_key_cert,4), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2di_zusi_issuer_key_cert =
  (generateCertFid(sec_2_2di_issuer_key_cert),
   Cert(encryptCert(sec_2_2di_issuer_key_cert)))

val sec_2_2di_zusi_si_issuer_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2di_issuer_key_cert),3),
   Si(H_M_I(sec_2_2di_si_seed_issuer_key_cert,1)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2di_store : P2P_DATA_LIST =
  [sec_2_2di_zusi_init_key_cert_random,sec_2_2di_zusi_si_init_key_cert_random,
   sec_2_2di_zusi_issuer_key_cert,sec_2_2di_zusi_si_issuer_key_cert]
```

Fall 2.2.d.ii

```
val sec_2_2dii_si_seed_init_key_cert = 210
val sec_2_2dii_si_seed_init_priv_cert = 211
val sec_2_2dii_si_seed_issuer_priv_cert = 212

val sec_2_2dii_init_key_cert : CERT =
  let val content = ("E", "pk", "publ-E",2,4,
            H_M_I(sec_2_2dii_si_seed_init_key_cert,3), "TA1")
  in (content, (hashCertcontent(content),("publ-TA1", "TA1"),s), a)
  end

val sec_2_2dii_zusi_init_key_cert =
  (generateCertFid(sec_2_2dii_init_key_cert),
   Cert(encryptCert(sec_2_2dii_init_key_cert)))

val sec_2_2dii_zusi_si_init_key_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2dii_init_key_cert),2),
   Si(H_M_I(sec_2_2dii_si_seed_init_key_cert,1)))

val sec_2_2dii_init_priv_cert_rand : CERT =
  let val content = ("E", "Privilege1", "2",3,5,
            H_M_I(sec_2_2dii_si_seed_init_priv_cert,3), "IA2")
  in (content, (hashCertcontent(content),("publ-Rand", "Random"),s), p)
  end
```

```
val sec_2_2dii_zusi_init_priv_cert_rand =
  (generateCertFid(sec_2_2dii_init_priv_cert_rand),
   Cert(encryptCert(sec_2_2dii_init_priv_cert_rand)))

(* newly assembled initial privilege cert with random signature *)
val sec_2_2dii_zusi_si_init_priv_cert_rand : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2dii_init_priv_cert_rand),1),
   Si(H_M_I(sec_2_2dii_si_seed_init_priv_cert,2)))

val sec_2_2dii_issuer_priv_cert : CERT =
  let val content = ("IA2", "Privilege1", "5",1,5,
            H_M_I(sec_2_2dii_si_seed_issuer_priv_cert,5), "SoA1")
  in (content, (hashCertcontent(content),("publ-SoA1", "SoA1"),s), p)
  end

val sec_2_2dii_zusi_issuer_priv_cert =
  (generateCertFid(sec_2_2dii_issuer_priv_cert),
   Cert(encryptCert(sec_2_2dii_issuer_priv_cert)))

val sec_2_2dii_zusi_si_issuer_priv_cert : P2P_DATA =
  (generateSiFid(hashCert(sec_2_2dii_issuer_priv_cert),3),
   Si(H_M_I(sec_2_2dii_si_seed_issuer_priv_cert,2)))

(* initial marking for place TopLevelModel'SN_DataStore *)
val sec_2_2dii_store : P2P_DATA_LIST =
  [sec_2_2dii_zusi_init_key_cert,sec_2_2dii_zusi_si_init_key_cert,
   sec_2_2dii_zusi_init_priv_cert_rand,sec_2_2dii_zusi_si_init_priv_cert_rand,
   sec_2_2dii_zusi_issuer_priv_cert,sec_2_2dii_zusi_si_issuer_priv_cert]
```

Anhang D

Abkürzungsverzeichnis

AAI	Authentifizierungs- und Autorisierungsinfrastruktur
AOL	America Online
CA	Certificate Authority; Zertifizierungsstelle
CAN	Content-Addressable Network
CFS	Cooperative File System; Verteiltes P2P Dateisystem
CPN	Coloured Petri Net; gefärbtes Petrinetz
CPU	Central Processing Unit
CRL	Certificate Revocation List
CRS	Certificate Revocation Status Directory
CRT	Certificate Revocation Tree
CVT	Certificate Verification Tree
DFN	Deutsches Forschungsnetz
DHT	Distributed Hash Table; verteilt gespeicherte Tabelle mit Key-Value-Zuordnungen
DNS	Domain Name System; Hierarchisches Server-System zur Namensauflösung im Internet
EFECT	Easy Fast Efficient Certificate Technique
EE	End-Entität; Zertifikatinhaber in einer AAI
FIFO	First In - First Out
HCPN	Hierarchical Coloured Petri Net
HCRS	Hierarchical Certificate Revocation System
GCC	GNU Compiler Collection
GNU	GNU is Not Unix; Rekursives Akronym - heute i.A. für freie Software gebraucht
GPL	General Public License; Open-Source Lizenz Richard Stallman
GSL	GNU Scientific Library; Freie und weitverbreitete Mathematikbibliothek
IA	Issuing Authority; Aussteller von Zertifikaten in einer AAI
ID	Identifier; (eindeutiger) Bezeichner

IP	Internet Protocol; Vermittelbares Netzwerk Protokoll
IPSec	Internet Protocol Security; Abgesicheres Internet Protokoll
IPv6	Internet Protocol Version 6
ISDN	Integrated Services Digital Network; Digitales Telefonnetz
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union; Kommunikationsagentur der UNO
LNCS	Lecture Notes in Computer Science; Buchserie des Springer Verlags
LRM	Local Relational Model; Datenmodell im Bereich P2P
MIT	Massachusetts Institute of Technology
NAT	Network Address Translation; Adressumsetzung und Maskierung von IP-Adressen
NFS	Network File System; Netzwerk Dateisystem von Sun Microsystems
OCSP	Online Certificate Status Protocol
OSI	Open Systems Interconnection; Ursprünglich: Netzwerkimplementierung - heute: Netzwerkreferenzmodell
P2P	Peer-to-Peer
P2P-ZuSI	Peer-to-Peer Zertifikat- und Status-Informationssystem
PC	Personal Computer
PERMIS	PrivilEge and Role Management Infrastructure Standards Validation Project
PIER	P2P Information Exchange and Retrieval; P2P Anfragesystem
PGP	Pretty Good Privacy; Programm zur Verschlüsselung von Privatnutzer-Kommunikation mit integrierter Web-of-Trust-PKI
PKI	Public Key Infrastructure
PMI	Privilege Management Infrastructure
RA	Registration Authority
RBAC	Role Based Access Control; Rollenbasierte Zugriffsrechte
RFC	Request for Comments
RPC	Remote Procedure Call; netzwerkbasierte Ausführung einer Prozedur auf einem anderen Rechner
RSA	Rivest, Shamir und Adleman (Kryptosystem); Asymmetrisches Verschlüsselungsverfahren
SA	Status Authority; Verwalter des Zertifikatstatus in einer AAI
SAML	Security Assertion Markup Language; Markup-Sprache für Sicherheits-Informationen
SB	Suicide Bureau; Entität, die öffentliche Schlüssel von EEs verwaltet
SDSI	Simple Distributed Security Infrastructure
Seti	Search for Extra Terrestrial Intelligence
SHA	Secure Hash Algorithm; Klasse standardisierter kryptographischer Hash-Funktionen
SI	Statusinformation; positive Statusnachricht gemäß CRS und NewPKI
SoA	Source of Authority

SOAP	Simple Object Access Protocol; Leichtgewichtiges Übertragungsprotokoll für Webservices
SPKI	Simple Public Key Infrastructure
SSL	Secure Sockets Layer; Hybrides, verschlüsseltes Übertragungsprotokoll
SSO	Single Sign-On
TA	Trust Anchor; Vertrauensanker
TCP	Transmission Control Protocol; Verbindungsorientiertes Transportprotokoll
TLS	Transport Layer Security
TTL	Time To Live; Begriff für die Gültigkeitsdauer in der Netzwerktechnik
UDP	User Datagram Protocol; Verbindungsloses Transportprotokoll
UnP	Username Password
V	Verifier; Prüfer in einer AAI
VOMS	Virtual Organizational Membership Service
VPN	Virtual Private Netzwork
WoT	Web of Trust
XOR	Exclusive Or

Anhang E

Dateien zur Dissertation

Bestandteil der Dissertation ist ein ZIP-Archiv mit den folgenden Inhalten:

HCPN/	Verzeichnis für CPN-Modelle
HCPN/README.txt	Hinweise zu CPN Tools
HCPN/Graphik/	Graphikdateien der CPN-Modelle
HCPN/Graphik/Grundmodell_und_AS2/	EPS-Abbildungen der Pages des Grundmodells, ebf. zutreffend für Angriffsszenario 2 (da impliziter Angreifer)
HCPN/Graphik/AS1/	EPS-Abbildungen der vom Grundmodell abweichenden Pages in Angriffsszenario 1
HCPN/Modelle/	Modell-Dateien im .cpn-Format
HCPN/Modelle/ZuSI-Grundmodell.cpn	Grundmodell aus Kapitel 6
HCPN/Modelle/ZuSI-Sicherheitsanalyse-Case1.cpn	Angriffsszenario 1
HCPN/Modelle/ZuSI-Sicherheitsanalyse-Case2-1.cpn	Angriffsszenario 2.1
HCPN/Modelle/ZuSI-Sicherheitsanalyse-Case2-2ab.cpn	Angriffsszenarien 2.2.a und 2.2.b
HCPN/Modelle/ZuSI-Sicherheitsanalyse-Case2-2c.cpn	Angriffsszenario 2.2.c
HCPN/Modelle/ZuSI-Sicherheitsanalyse-Case2-2d.cpn	Angriffsszenario 2.2.d
Kad-Analyse/	Verzeichnis für stochastische Analyse von Kademlia
Kad-Analyse/gsl-1.9.tar.gz	GNU Scientific Library als Source-Tarball
Kad-Analyse/Makefile	Anweisungen für `make` zum Erstellen des Binärprogramms für die Kademlia-Analyse
Kad-Analyse/bin/	Speicherort des Analyse-Binärprogramms
Kad-Analyse/obj/	Speicherort der Objektdatei
Kad-Analyse/src/analysis.cpp	Quellcode

Das Archiv steht unter der Adresse http://www.koenen-dresp.de/research-wiebke/ zur Verfügung.

Literaturverzeichnis

[ABC02] Adya A., Bolosky W., Castro M. et al.: FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. Proceedings of the 5th OSDI,2002.

[ACC03] Alfieri R., Cecchini R., Ciaschini V., dell'Agnello L., Frohner A., Gianoli A., Lörentey K., Spataro F.: VOMS, an Authorization System for Virtual Organizations. Proc. 1st European Across Grids Conference, Santiago de Compostela, LNCS 2970, 2003.

[ACE96] Avior A., Calamoneri T., Even S., Litman A., Rosenberg A.: A Tight Layout of the Butterfly Network. ACM Symposium on Parallel Algorithms and Architectures, 1996, pp. 170-175.

[ACM02] Ajmani S., Clarke D., Moh C. et al.: ConChord: Cooperative SDSI Certificate Storage and Name Resolution. First International Workshop on Peer-to-Peer Systems (IPTPS), LNCS 2429, 2002, pp. 141-154.

[ADH03] Aberer K.,Datta A., Hauswirth M.: Beyond "Web of Trust": Enabling P2P E-commerce. Proceedings of the IEEE Conference on Electronic Commerce (CEC'03), 2003.

[ALO98] Aiello W., Lodha S., Ostrovsky R.: Fast Digital Identity Revocation (Extended Abstract). Advances in Cryptology - CRYPTO'98, Springer Verlag, 1998, pp. 137-152.

[AlS04] Alshamsi A., Saito T.: A Technical Comparison of IPSec and SSL. Cryptology ePrint Archive, Report 2004/314, 2004.

[AnR06] Anceaume E., Ravoaja A.: Incentive-based Robust Reputation Mechanism for P2P Services. Proceedings of the 10th International Conference On Principles Of Distributed Systems (OPODIS), 2006, pp. 305-319.

[AnS04] Androutsellis-Theotokis S., Spinellis D.: A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, Vol. 36, No. 4., 2004, pp. 335-371.

[APH02] Aberer K., Punceva M., Hauswirth M., Schmidt R.: Improving Data Access in P2P Systems. IEEE Internet Computing Vol. 6 No. 1, 2002, pp. 58-67.

[BDE00] Bolosky W., Douceur J., Ely D., Theimer M.: Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. ACM SIGMETRICS Performance Evaluation Review Archive Vol. 28, Issue 1, 2000, pp. 34-43.

[BaN04] Baker M., Nottingham M.: The "Application/SOAP+XML" Media Type. RFC[1] 3902, Network Working Group. http://www.ietf.org/rfc/rfc3902.txt, 2004.

[BSI06] Bundesamt für Sicherheit in der Informationstechnik: IT-Grundschutz-Kataloge. 8. Ergänzung, http://www.bsi.de/gshb/deutsch/index.htm, 2006.

[BSV02] Bhagwan R., Savage S., Voelker G.: Replication Strategies for Highly Available Peer-to-Peer Storage Systems. Technical Report CS2002-0726, UCSD, 2002.

[BTC04] Bhagwan R., Tati K., Cheng Y., Savage S., Voelker G.: Total Recall: System Support for Automated Availability Management. Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Vol. 1, 2004.

[CDG02] Castro M., Druschel P., Ganesh A.: Security for Structured Peer-to-Peer Overlay Networks. Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA, 2002.

[CGP05] Cerri D., Ghioni A., Paraboschi S., Tiraboschi S.: ID Mapping Attacks in P2P Networks. Proceedings of the IEEE Globecom '05, 2005.

[Cha02] Chadwick D.: An X.509 Role-based Privilege Management Infrastructure. Global Infosecurity, 2002.

[CHP93] Christensen S., Petrucci L.: Towards a Modular Analysis of Coloured Petri Nets. Proceedings of the 13th Int. Conf. Application and Theory of Petri Nets (ICATPN'92), LNCS 616, 1992, pp 113-133.

[CKP05] Cantor S., Kemp J., Philpott R., Maler E.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, 2005.

[CLL02] Chu J., Labonte S., Levine N.: Availability and Locality Measurements of Peer-to-Peer File Systems. Proceedings of SPIE, Vol. 4868, Scalability and Traffic Control in IP Networks II, 2002, pp. 310-321.

[DiA99] Dierks T., Allen C.: RFC 2246 - The TLS Protocol Version 1.0: Network Working Group, http://www.faqs.org/rfcs/rfc2246.html, 1999

[DiH06] Dinger J., Hartenstein H.: Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. First International Conference on Availability, Reliability and Security (ARES 2006), 2006, pp. 756-763.

[DKK01] Dabek F., Kaashoek M., Karger D. et al.: Wide-area Cooperative Storage with CFS. Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Canada, 2001.

[DLM02] Dawson E., Lopez J., Montenegro A., Okamoto E.: A New Design of Privilege Management Infrastructure for Organizations using Outsourced PKI. 5th International Conference on Information Security (ISC'02), LNCS 2433, 2002, pp. 136-149.

[1] RFC - Request for Comments

[Dou02] Douceur J.: The Sybil Attack. Proceedings of the IPTPS02 Workshop, Cambridge, USA, 2002.

[Dre05] Dresp W.: Security Analysis of the Secure Authentication Protocol by Means of Coloured Petri Nets. Proceedings of CMS '05, LNCS 3677, 2005, pp. 230-239.

[DSS05] Dongarra J., Sterling T., Simon H., Strohmaier E.: High Performance Computing: Clusters, Constellations, MPPs and Future Directions. Computing in Science and Engineering, Vol. 7, Nr. 2, 2005, pp. 51-59.

[DTM95] Doyle E., Tavares S., Meijer H.: Automated Security Analysis of Cryptographic Protocols Using Colored Petri Net Specifications. Workshop on Selected Areas in Cryptography, SAC '95 Workshop Record, 1995, pp. 35-48.

[Feh93] Fehling R.: A Concept of Hierarchical Petri Nets with Building Blocks. Advances in Petri Nets '93, LNCS 674, 1993, pp. 148-168.

[FKP04] Fahrmeir L., Künstler R., Pigeot I.: Statisk - Der Weg zur Datenanalyse. 5. Auflage. Springer Verlag, 2004.

[Gee02] Geels D.: Data Replication in OceanStore. U.C. Berkeley Master's Report, Technical Report UCB//CSD-02-1217, 2002.

[GGG03] Gummadi K., Gummadi R., Gribble S., Ratnasamy S., Shenker S., Stoica I.: The Impact of DHT Routing Geometry on Resilience and Proximitry. Proceeding of the ACM SIGCOMM, 2003, pp. 381-394.

[GGM00] Gassko I., Gemmell P., MacKenzie P.: Efficient and Fresh Certification. Public Key Cryptography 2000, LNCS 1751, 2000, pp. 342-353.

[Gut03] Gutmann P.: PKI Technology Survey and Blueprint.

[HaR02] Hand S., Roscoe T.: Mnemosyne: Peer-to-Peer Steganographic Storage. Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Boston, 2002.

[HML03] Huebsch R., Hellerstein J., Lanham N., Thau Loo B., Shenker S., Stoica I.: Querying the Internet with PIER. Proceedings of 19th International Conference on Very Large Databases, 2003.

[HoP01] Housley R., Polk T.: Certificate Revocation Lists. Planning for PKI, John Wiley and Sons, 2001.

[HPF02] Housley R., Polk W., Ford W., Solo D.: Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile. PKIX Working Group, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc3280.txt, 2002.

[IRD02] Iyer S., Rowstron A., Druschel P.: Squirrel: A Decentralized Peer-to-Peer Web Cache. Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing, 2002, pp. 213-222.

[Jen06] Jensen K.: Beitrag im Thread *[CPNTools-support] Formal definition to Hierarchical CPN* der CPN Tools Mailing List, http://mailman.daimi.au.dk/pipermail/cpntools-support/2006-July/002217.html, 25. Juli 2006.

[Jen92] Jensen K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1: Basic concepts. Monographs in Theoretical Computer Science - an EATCS Series, Springer Verlag, 1992.

[KBC00] Kubiatowicz J., Bindel D., Chen Y. et al.: OceanStore: An Architecture for Global-Scale Persistent Storage. Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000.

[KLL97] Karger D., Lehman E., Leighton T., Levine M., Lewin D., Panigrahy R.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. ACM Symposium on Theory of Computing, 1997, pp. 654-663.

[Koc98] Kocher P.: On Certificate Revocation and Validation. FC: International Conference on Financial Cryptography, LNCS 1465,1998, 172-177.

[KSG03] Kamvar S., Schlosser M., Garcia-Molina H.: The EigenTrust Algorithm for Reputation Management in P2P Networks. Proceedings of the 12th International World Wide Web Conference, Budapest, 2003.

[LOP03] Lopez J., Oppliger R., Pernul G.: Authentication and Authorization Infrastructures (AAI): A Comparative Survey. Computers and Security, Vol. 23, No. 7, 2004, pp. 578-590.

[MAM99] Myers M., Ankney R., Malpani A., Galperin S., Adams C.: OCSP: Online Certificate Status Protocol. Internet Eng. Task Force RFC 2560, www.ietf.org/rfcrfc2560.txt, 1999.

[MaM02] Maymounkov P., Mazieres D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. Proceedings of IPTPS 2002, pp. 53-65.

[Mau96] Maurer U.: Modelling a Public Key Infrastructure. Proceedings of the 1996 European Symposium on Research in Computer Security (ESORICS96). LNCS 1146, 1996, pp. 325-350.

[MBR03] Manku G., Bawa M., Raghavan P.: Symphony: Distributed Hashing in a Small World. Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS), 2003 pp. 127-140.

[Mer79] Merkle, R.: Secrecy, Authentication, and Public Key Systems. Ph.D. Dissertation, Dept. of Electrical Engineering, Stanford University, 1979.

[Mic02] Micali S.: NOVOMODO: Scalable Certificate Validation and Simplified PKI Management. Proceedings of the 1st Annual PKI Research Workshop, 2002, pp 15-25.

[Mic96] Micali S.: Efficient Certificate Revocation. 1996

[MMG02] Muthitacharoen A., Morris R., Gil T. et al.: Ivy: A Read/Write Peer-to-Peer File System. Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[MNR02] Malkhi D., Naor M., Ratajczak D.: Viceroy: A Scalable and Dynamic Emulation of the Butterfly. Proceedings of the 21st Annual Symposium on Principles of Distributed Computing, New York, 2002, pp. 183-192.

[MOV96] Menezes A., van Oorschot P., Vanstone S.: Handbook of Applied Cryptography. CRC Press, 1996.

[MTH97] Milner R., Tofte M., Harper R., MacQueen D.: Definition of Standard ML (Revised). MIT Press, 1997.

[NaN98] Naor M., Nissim K.: Certificate Revocation and Certificate Update. Proceedings 7th USENIX Security Symposium, 1998.

[NeH04] Németh E., Hangos K.: Multi-Scale Process Model Description by Generalizd Hierarchical CPN Models. Research Report SCL-002, 2004.

[NyB99] Nyström M.,Brainard J.: An X.509-Compatible Syntax for Compact Certificates. CQRE(Secure), LNCS 1740, 1999, pp. 76-93.

[Ora01] Oram A. et al.: Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. O'Reilly, Sebastopol, 2001.

[Pet62] Petri, C.: Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962.

[RiL96] Rivest R., Lampson L.: SDSI - A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/ cis/sdsi.html, 1996.

[Riv98] Rivest R.: Can We Eliminate Certificate Revocation Lists? Proceedings 2nd International Conference on Financial Cryptography, 1998, pp. 178-183.

[RFH01] Ratnasamy S., Francis P., Handley M. et al.: A scalable Content Addressable Network. Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, 2001 pp. 161-172.

[RoD01a] Rowstron A., Druschel P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, 2001, pp. 329-350.

[RoD01b] Rowstron A., Druschel P.: PAST: A large-scale, persistent peer-to-peer storage utility. HotOS VIII, Schoss Elmau, Germany, 2001.

[Sch95] Schneier B.: Applied Cryptography, 2nd ed. John Wiley and Sons, 1995.

[Sch74] Schreiber F.R.: Sybil - The True Story of a Woman Possessed by Sixteen Separate Personalities. 1974.

[SGM01] Serafini L., Mylopoulos J., Giunchiglia F., Bernstein P.A.: The Local Relational Model: Model and Proof Theory. Technical Report 0112-23, ITC-IRST, 2001.

[SHS01] U.S. Department of Commerce, National Institute of Standards and Technology NIST: Federal Information Processing Standards Publication 180-2, Secure Hash Standard. National Technical Information Service, 2001.

[SHZ05] Song S., Hwang K., Zhou R., Kwok Y. K.: Trusted P2P Transactions with Fuzzy Reputation Aggregation. IEEE Internet Computing Vol. 9, 2005, pp. 24-34.

[SiM02] Sit E., Morris R.: Security Considerations for Peer-to-Peer Distributed Hash Tables. IPTPS'01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer, 2002, pp. 261 - 269.

[SKK02] Saito Y., Karamanolis C., Karlsson M. et al.: Taming Aggressive Replication in the Pangaea Wide-Area File System. SIGOPS Operating Systems Review Vol. 36, ACM Press, 2002, pp. 15-30.

[SML01] Stoica I., Morris R., Liben-Novell D. et al.: Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. Proceedings of SIGCOMM'01, San Diego, CA, 2001, pp. 149-160.

[SrL04] Srivatsa M., Liu L.: Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. Proceedings of the 20th IEEE Annual Computer Security Applications Conference ACSAC, 2004 pp. 252-261.

[Sti02] Stinson D.: Cryptography - Theory and Practice (Discrete Mathematics and its Application). 2nd Edition, Taylor & Francis, 2002.

[TaV06] Tati K., Voelker G.M.: On Object Maintenance in Peer-to-Peer Systems. Proceedings of the 5th International Workshop on Peer-to-Peer Systems IPTPS'06, 2006.

[Val98] Valmari A.: The State Explosion Problem. Lectures on Petri Nets I: Basic Models, LNCS 1491, 1998, pp. 429-528.

[WeK02] Weatherspoon H., Kubiatowicz J.: Erasure Coding vs. Replication: A Quantitative Comparison. First International Workshop on Peer-to-Peer Systems, LNCS 2429, 2002, pp. 328-338.

[Woe06] Wölfl T.: Formale Modellierung von Authentifizierungs- und Autorisierungsinfrastrukturen. Dissertation. Deutscher Universitätsverlag, 2006.

[YTD06] Yang Z., Tian J., Dai Y.: Towards a more accurate availability evaluation in peer-to-peer storage systems. Networking, Architecture, and Storages, 2006. NAS '06, Aug. 2006

[ZKJ01] Zhao B., Kubiatowicz J., Joseph A.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. UCB/CSD-01-1141, UC Berkeley, 2001.

[ZhM04] Zhou W., Meinel C.: Implement Role Based Access Control with Attribute Certificates. Proceedings of 6th International Conference on Advanced Communication Technology ICACT'04, Vol.1, 2004, pp. 536-541.

[Zho03] Zhou J.: Efficient Signature Validation Based on a New PKI. Proceedings 4th International Conference E-Commerce and Web Technologies, LNCS 2738, 2003, pp. 94-103.

[Zim80] Zimmermann H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. IEEE Transactions on Communications COM-28, No. 4, 1980.

[AAR] Verteilte Authentifizierung, Autorisierung und Rechteverwaltung AAR. http://aar.vascoda.de/

[AKENT] AKENTI. http://dsd.lbl.gov/security/Akenti/

[BITTO] BitTorrent. http://bittorrent.com/

[COMAR] Comarch. http://www.comarch.pl/

[CPNML] CPN/ML Introduction. http://wiki.daimi.au.dk/cpntools-help/cpn_ml.wiki/

[CPNTO] CPN Tools. http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[EMKAD] eMule Kademlia Mods. http://www.kademlia-mods.de/

[EXEEM] eXeem BitTorrent Program. http://exeem.com/

[FARSI] Farsite Storage System. http://research.microsoft.com/sn/Farsite/

[FPKI] http://csrc.nist.gov/pki/publickey.html

[GNUTE] Gnutella. http://www.gnutella.com

[ICQ] ICQ. http://www.icq.com/

[INTER] Internet2 Project Page. http://www.internet2.edu/

[KENOS] Kenosis Kademlia-Style Network. http://kenosis.sourceforge.net/

[KERBE] KERBEROS Project. http://web.mit.edu/kerberos/www/

[LIBER] Liberty Alliance. http:/www.projectliberty.org/

[LIMEW] LimeWire Project. http://www.limewire.org/

[NAPST] Napster. http://www.napster.com

[OCEAN] OceanStore. http://oceanstore.cs.berkeley.edu/

[PASSP] Microsoft .NET Passport. http://www.passport.net/

[PERMI] PERMIS. http://sec.isi.salford.ac.uk/permis/

[PGP] Internationale PGP Homepage. http://www.pgpi.org/

[RZURG] CIP-Pool-Liste des Rechenzentrums der Universität Regensburg. http://www.uni-regensburg.de/Einrichtungen/RZ/Benutzer/Allgemein/CIP/CIP-Rechner.phtml

[SESAM] SESAME Project. https://www.cosic.esat.kuleuven.be/sesame/

[SHIBB] Shibboleth. http://shibboleth.internet2.edu/

[SWITC] Switch AAI. http://www.switch.ch/aai/

[VICAP] Viceroy. http://www.cs.huji.ac.il/labs/danss/anatt/viceroy.html

Abbildungsverzeichnis

Tabellenverzeichnis