Sicherheitsaspekte von Web Services

- Diplom -

Mirko Richter
Technische Universität Dresden

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Einleitender Text - 2be done at the end :)

1.1 Webservice

Mit der Entwicklung der erst wenige Jahre alten Webservice-Technologie wurde ein Standard geboren, welcher in der gesamten Computerindustrie breite Unterstützung erfährt und die Großen der Softwarebranche in noch nie da gewesener Art und Weise eint.
Die wichtigsten und wohl auch bekanntesten Vertreter in dieser Allianz sind IBM, Microsoft, Sun, HP, Oracle, SAP und Novell. Neben diesen existieren natürlich noch eine Vielzahl weitere Unternehmen, die sich in diesem Bereich engagieren.
Doch was sind eigentlich Web Services und was macht sie so erfolgreich?
IBM definiert Web Services wie folgt:

> "*Web Services are self-contained, modular applications that can be described, published, located and invoked over a network, generally, the World Wide Web.*"
>
> "Webservices sind in sich abgeschlossene, modulare Anwendungen, welche über ein Netzwerk, standardmäßig das Internet, beschrieben, veröffentlicht, lokalisiert und verwendet werden können."

Diese Definition umschreibt einen Webservice im Kontext seiner ihn standardmäßig umgebenden Architektur sehr treffend, jedoch soll in diesem und im folgenden Abschnitt zwischen einem Webservice als alleinstehenden Diensterbringer[1] und dem verwendeten Architektur-Prinzip (SOA s.Kap. 1.2) unterschieden werden.

Webservices sind Komponenten, welche in Netzwerken (z.B. dem Internet) verfügbar gemacht werden können und sich auf Grund ihrer fest definierten Schnittstelle sowie ihrer loser Kopplung bestens zur Wiederverwendung eignen. Lose Kopplung bedeutet hier, dass erst zur Laufzeit eine Bindung an andere Services erfolgt. Dadurch wird es ermöglicht, selbst komplexe Anwendungen zur Laufzeit bausteinartig zusammenzusetzen.
Werden Webservices im Internet verfügbar gemacht, können auch andere Unternehmen oder Personen auf diese zugreifen (dynamisches Binden) und diese verwenden, *ohne sie kopieren zu müssen.*
Dadurch entstehen vernetzte Strukturen, in denen Dienste ihrem durch den Entwickler vorgesehenen Zweck oder durchaus auch einer Zweckentfremdung zugeführt werden können und möglicherweise einen Baustein für eine größere und komplexere Struktur darstellen. "Zweckentfremdung" kann dabei im positiven (intelligente, konstruktive und legale Verwendung z.B. in einem anderen Kontext als vorgesehen) als auch im negativen (Missbrauch z.B. für Verbrechen) Sinn verstanden werden. Grundsätzlich sollte man sich genau darüber im Klaren sein, *welche* Services man öffentlich zur Verfügung stellt und wie man sich durch AGBs oder ähnliche Regelwerke gegen mögliche Missbrauchsabsichten schützt.

Webservices verwenden für die Kommunikation untereinander ein Transportprotokoll namens "SOAP" (s.Kap. 3.2), welches unabhängig vom jeweils verwendeten Host-Transportprotokoll (z.B. HTTP; s.Kap. 3.3) funktioniert und auf Grund seines XML-Formats (s.Kap. 3.1) für den Menschen wesentlich besser lesbar ist, als z.B. Binärdatenströme.
Eine der wichtigsten Vorzüge von SOAP ist dessen modular definierter Aufbau. Um die

[1] Die Begriffe "Service" und "Dienst" werden im Kontext dieser Arbeit gleichbedeutend mit "Webservice" verwendet.

Komplexität der resultierenden Webservice-Implementationen möglichst gering zu halten, wurden sämtliche Protokollerweiterungen in einer Art und Weise definiert, die es ermöglicht sie *nur bei Bedarf* in eine Kommunikation einzubeziehen. Dadurch können flexible, webservicekonforme und vor allem *schlanke* Protokolle entwickelt und umgesetzt werden.

Ein weiterer interessanter Punkt, welcher sowohl positive als auch negative Eigenschaften in die Definition von Webservice einbringt, ist die standardmäßige Verwendung von Host-Transportprotokollen für den SOAP-Datenaustausch (s.Kap. 3.3). Auf der einen Seite steht die einfache Etablierung von Webservices in einem Unternehmen und auf der anderen Seite die Tatsache, dass Services potentiell unbemerkt durch herkömmliche Firewalls hindurch erreichbar sind.
Ersteres wird dadurch ermöglicht, dass die zugrunde liegenden Ports standardmäßig für den Mail- und Internetverkehr freigeschaltet sind, was zu einem verminderten initialen Administrationsaufwand für die Einführung von Webservices führt. Dadurch wird leider prinzipiell jedem direkten Angriff auf den Service Tür und Tor geöffnet, da herkömmliche Firewalls eingehende Daten nicht filtern. Die Entwicklung von intelligenteren Firewalls, die in SOAP-Nachrichten "hineinschauen" und aufwändig anhand komplexer Regeln filtern, steckt zur Zeit noch in den Kinderschuhen.

CORBA-Verfechter werden sich jetzt sicherlich fragen, wo die eigentliche Neuerung von Webservices liegt. Technologisch betrachtet sind die Unterschiede zwischen CORBA und Webservices nicht sehr groß. CORBA besitzt ebenfalls die Basisbausteine "Service Provider", "Service Requester" und "Service Broker" sowie standardisierte Schnittstellen und ein standardisiertes Protokoll.
Vermutlich war man mit CORBA auch kurz davor, die gleichen Möglichkeiten zu eröffnen, wie sie heute bei den Webservices existieren. Es wurden jedoch einige vermarktungstechnische Fehler gemacht, welche das sehr wahrscheinlich verhinderten.
So wurde z.B. versucht eine gänzlich neues Protokoll (IIOP) im Internet durchzusetzen, anstatt ein existierendes zu verwenden und es wurde dabei bis CORBA 3 auch noch versäumt den Kommunikationsport zu standardisieren. Netzwerkadministratoren waren somit gezwungen, zu jedem Kommunikationspartner einen neuen Port in der Firewall zu öffnen, was über kurz oder lang zum sprichwörtlichen "Schweizer Käse" im digitalen Verteidigungsbollwerk führte. Sicherlich war auch die unzureichende Spezifikation von Sicherheitsaspekten ein weitere Grund, warum sich der Eine oder der Andere von CORBA ab- und den Webservices zuwendete.

1.2 Serviceorientierte Architektur und Webservices

SOA (Service Oriented Architecture) ist ein Architektur-Prinzip, welches meist im Zusammenhang mit Webservices erwähnt wird, aber prinzipiell auch mit anderen Technologien, wie z.B. JMS, EJB oder CORBA umsetzbar ist. Oftmals stellen andere Technologien sogar auf Grund ihres leichter verarbeitbaren binären Protokolls (CORBA), ihrer Zusagen über garantierte Auslieferung (JMS) oder anderen Faktoren für kleine oder firmeninterne Projekte die besser Alternative dar.

Die wichtigste Grundidee einer SOA lässt sich mit der Aussage "*publish, find and bind*"[2] [On02] ausdrücken und bezieht sich dabei allgemein auf Services.
Technologieunabhängigkeit, Unabhängigkeit untereinander, eine gute Dokumentation, Metadaten für den automatisierten Zugriff und eine abgeschlossenen Funktionalität sind Grund-

[2] veröffentlichen, finden, binden

eigenschaften, welche durch jeden Service im SOA-Kontext erfüllt sein sollten. Damit wird gewährleistet, dass unabhängige Dienstleistungen so verfügbar gemacht werden können, dass Interessierte sie weitgehend automatisiert auffinden und verwenden können.

Aus Entwicklersicht lässt sich für SOA sagen, dass sie eine evolutionäre Weiterentwicklung der verbreiteten CBD (Component Based Development) darstellt. Konsequent durchdachte Komponentenarchitekturen können daher mit einem geringen Aufwand in die SOA-Welt portiert werden.
Die Schnittstellengranularität von Services kann im Allgemeinen als etwas "gröber" als die von Komponenten angesehen bzw. auch mit ihr gleichgesetzt verstanden werden. Alle weiteren Kerneigenschaften von Komponenten [Ri04] werden von Services ebenfalls erfüllt.

Die verbreitete Verwendung von *Webservices* in serviceorientierten Architekturen ist auf die Unterstützung der weltweit größten Softwarefirmen (s.Kap. 1.1) für dieses Konzept zurückzuführen. Dieser durch die Mehrheit der Industrie angenommene Standard könnte in Zukunft gänzlich neue Geschäftsideen, beruhend auf der geschickten Verbindung unterschiedlicher Services, ermöglichen.
Z.B. erregte Erik Benson im Juli 2002 auf seiner Seite *mockerybird.com* Aufsehen, als er Webservices vom Book Watch Service, Amazon, Google und einiger Weblogs (Anzahl der Nennungen von Büchern) verwendete, um so zusätzlicher Informationen zu Büchern anbieten zu können. Das Ergebnis war eine Anwendung, welche ihre Daten unter dem Name "All Consuming" (*allconsuming.net*) im Internet zur Verfügung stellt.

Im Kontext dieser Arbeit ist die Verwendung des Begriffes "SOA" stets im Zusammenhang mit "Webservices" zu verstehen.

2 Hintergrund

Dieses Kapitel stellt verschiedene Begriffe und Konzepte, welche für das Verständnis der vorliegenden Arbeit von Bedeutung sind, aber das Thema "Webservices" nur indirekt anschneiden, näher vor und besitzt somit einen herausgelösten Status innerhalb dieser Diplomarbeit.
Leser, welche mit den Begrifflichkeiten vertraut sind, können dieses Kapitel problemlos überspringen und bei Bedarf im Verlauf der Arbeit zum Nachschlagen verwenden.

2.1 OSI-Architektur

Um zu verstehen, auf welcher OSI-Ebene die im Folgenden vorgestellten Sicherheitsmechanismen angesiedelt sind, soll hier das OSI-Modell im Überblick vorgestellt werden.
Durch dieses Modell wird die Behandlung von zur Netzwerkkommunikation bestimmter Daten von der sendenden Applikation bis zum Versand über eine physikalische Leitung in 7 Schichten (Abb. 1; [On02]) aufgeteilt. Es wird dabei zwischen folgenden Schichten unterschieden:

Schicht 7 Die **Anwendungs-Schicht** repräsentiert Dienste, welche Endanwendungen direkt unterstützen und ermöglicht den Zugriff auf diese (z.B. SOAP-Nachrichten, DNS, Telnet, Finger).

Schicht 6 Die **Präsentations-Schicht** ist verantwortlich für Protokoll- und Zeichen-Umwandlungen, Ver- und Entschlüsselung, Kompression etc. und muss in

Schicht-Nummer	Schicht-Name	Webservice-Technologien
Schicht 7	Anwendung	HTTP, SMTP, SOAP
Schicht 6	Präsentation	Encrypted Data, Compressed Data
Schicht 5	Sitzung	POP/25, SSL
Schicht 4	Transport	TCP / UDP
Schicht 3	Netzwerk	IP-Packete
Schicht 2	Daten-Verbindung	PPP, 802.11, etc.
Schicht 1	Physikalisch	ADSL, ATM, etc.

Abbildung 1: OSI-Schichten-Modell

einem Netzwerkprotokoll nicht zwingend vorhanden sein. Die Hauptaufgabe besteht also in der Umsetzung von Applikations-Datenformaten in Datenformate, welche von den Schichten 1-5 verstanden und dann weiterverwendet werden können (z.B. XML-Encryption).

Schicht 5 Die **Sitzungs-Schicht** erstellt, unterhält und beendet Sitzungen (Sessions). Dazu muss die agierende Person eindeutig identifiziert werden können (z.B. SSL).

Schicht 4 Die **Transport-Schicht** ist verantwortlich für die Aufteilung von Datenströmen in Pakete beim Schicken und ihre fehlerfreie Zusammensetzung beim Empfangen. Fehler (z.B. Duplikate oder Paket-Verluste) werden entsprechend behandelt (z.B. TCP/UDP).

Schicht 3 Die **Netzwerk-Schicht** ist verantwortlich für die Übersetzung von logischen Netzwerkadressen in ihre physischen Pendants und für das Routing der Pakete. Die Behandlung von Datenstaus und Routing-Fehlern fallen ebenfalls in ihren Aufgabenbereich (z.B. IP, IPX).

Schicht 2 Die **Daten-Verbindungs-Schicht** wandelt Pakete in Bits und umgekehrt. Sie ist verantwortlich für die Festlegung der Übertragungsmethode und für die fehlerfreie Übermittlung von Datenrahmen über die physikalische Schicht (z.B. PPP).

Schicht 1 Die **Physikalische Schicht** überträgt Bitströme über das physikalische Kabel und definiert entsprechende Techniken (z.B. ISDN, ADSL).

2.2 Daten maschinenunabhängig darstellen - *ASN.1*

Der ASN.1-Standard (Abstract Syntax Notation One) ist eine Entwicklung einer Vielzahl von, unter dem ASN.1 Consortium [ASN.1] zusammengefassten, Einzelpersonen und Unternehmen (Nokia, Fujitsu, Cisco, NEC, Lucent etc.), welcher in seiner ersten Version 1984 veröffentlicht wurde. Er umfasst eine Sammlung von Techniken und Notationsvorschriften, um unterschiedlichste Datenstrukturen (Strings, Integer, Arrays etc.) in einer binären, möglichst kleinen[3] und maschinenunabhängigen Art und Weise darzustellen. Durch diese Kompaktheit genießt ASN.1 den Ruf, sehr komplex und durch den Menschen nur sehr schwer lesbar zu sein.
Bis heute findet er in einer Vielzahl von Protokollen, Standards (z.B. S/MIME, X.509, X.400) und Neuentwicklungen Verwendung.

[3] im Sinne der Datenmenge

Anhand der Produktpalette einiger unterstützender Unternehmen lässt sich bereits erkennen, dass ein wichtiges Aufgabenfeld in der Kommunikationsindustrie liegen muss. So wird ASN.1 beispielsweise bei Handys, der ISDN-Telefonie und Audio- sowie Videostreamingtechnologien eingesetzt.
Durch seine enge Verbindung mit X.509-Zertifikaten besitzt dieser Standard auch weiterhin Relevanz im Bereich der Webservicesicherheit.

2.3 Grundlagen Kryptografie

Die Kryptografie als Wissenschaft stellt neben der Krypt(o)analyse einen Teilbereich der Kryptologie (Wissenschaft, oder griechisch Weisheit, der Geheimschriften) dar.
In modernen IT-Systemen ist die Kryptografie nicht mehr wegzudenken, da ein physischen Schutz vor Datenkriminalität oftmals nicht umgesetzt werden kann bzw. auch nicht soll [Pf00].
Prof. Pfitzmann unterscheidet in [Pf00] kryptografische Systeme nach drei Kriterien:

- nach ihrem **Zweck**:

  Konzelationssysteme, für die Geheimhaltung von Nachrichten (Schutzziel *Vertraulichkeit*; s.Kap. 4.1.1)

  Authentikationsysteme, für den Schutz von Nachrichteninhalten gegen unbemerkte Veränderung (Schutzziel *Integrität*; s.Kap. 4.1.2).
  Digitale Signatursysteme stellen einen Spezialfall dar, da mit dessen Hilfe die *Beweisbarkeit*, dass eine Nachricht von einem bestimmten Sender verfasst wurde, gegenüber Außenstehenden (z.B. einem Gericht) gegeben ist (Schutzziele *Integrität* und *Verbindlichkeit*; s.Kap. 4.1.3).

- nach der **Schlüsselverteilung**:

  symmetrisch, falls beide Parteien (Sender und Empfänger) über den selben oder zumindest über einen mit polynomialen Aufwand aus dem jeweils anderen Schlüssel bestimmbaren Schlüssel verfügen.

  asymmetrisch, falls eine Partei über ein Geheimnis, den öffentlichen und den privaten Schlüssel und die andere Partei nur über den öffentlichen Schlüssel verfügt. Die Bestimmung des privaten Schlüssels aus dem öffentlichen Schlüssels ist im Optimalfall nur unter sehr großem Aufwand möglich (informationstheoretische Sicherheit kann mit asymmetrischen Systemen nie erreicht werden).

- und nach ihrem **Sicherheitsgrad**:

  informationstheoretische Sicherheit, falls bewiesen ist, dass das Brechen eines als solches eingestuften kryptografischen Systems durch keine Berechnung sondern allenfalls durch Probieren erreicht werden kann. Zu jedem möglichen Klartext[4] kann dabei ein Schlüssel gefunden werden, der diesen auf den zu brechenden Schlüsseltext[5] abbildet (z.B. Vernam Chiffre, auch bekannt als "one-time-pad").

  kryptografisch starke Sicherheit, falls bewiesen ist, dass das vollständige Brechen[6] eines so eingestuften kryptografischen Systems ein in der Komplexitäts-

[4] Ausgangsdaten für eine kryptografische Operation.
[5] Ergebnis einer kryptografischen Operation.
[6] Berechnen des geheimen Schlüssels aus öffentlich verfügbaren Informationen (z.B. dem öffentlichen Schlüssel oder abgehörten Schlüsseltexten)

theorie als "schwer"[7] eingestuftes Problem (z.B. Faktorisierung) lösen würde (z.B. Pseudo-one-time-pad mit s^2-*mod-n*-Generator).

wohluntersuchte Sicherheit, falls es (noch) keinen Beweis dafür gibt, dass die zugrunde liegenden Annahmen für kryptografisch starke Sicherheit ausreichen, der Algorithmus jedoch von möglichst vielen Personen möglichst lange untersucht worden ist. Solche Systeme werden mit zunehmendem Alter entweder gebrochen oder vertrauenswürdiger (z.B. RSA).

2.3.1 Symmetrische Kryptografie

In der symmetrischen Kryptografie verfügen alle Teilnehmer (in der Regel zwei) über den selben Schlüssel oder zumindest über Schlüssel, die mit polynomialen Aufwand auseinander ableitbar sind.
Das offensichtliche Problem hierbei liegt in der Schlüsselverteilung. Jeder verschlüsselten Kommunikation, die über ein unsicheres Netz aufgebaut wird, muss ein geheimer Schlüsselaustausch vorausgehen. Dies kann z.B. passieren, wenn sich die Teilnehmer persönlich treffen.
Ein weiterer Nachteil ist die Tatsache, dass ein Teilnehmer, um mit n anderen Teilnehmern vertraulich kommunizieren zu können, mit diesen n Schlüssel (mit jedem jeweils einen) vertraulich austauschen und verwalten muss.
Demgegenüber steht jedoch der im Vergleich zur asymmetrischen Kryptografie relativ geringe Aufwand für Ver- und Entschlüsselungsoperationen und eine geringere Schlüssellänge.
Vertreter für symmetrische Konzelationssysteme wären z.B. die Vernam-Chiffre oder der DES (Data Encryption Standard) und für Authentikationssysteme die Authentikationscodes oder ebenfalls der DES.
Jedes symmetrische Konzelationssystem kann auch als symmetrisches Authentikationssystem und umgekehrt verwendet werden [Pf00].

2.3.2 Asymmetrische Kryptografie

Im Gegensatz zur symmetrischen Kryptografie benötigt ein Teilnehmer, der von n anderen Teilnehmern vertraulich Nachrichten erhalten möchte nur ein einziges Schlüsselpaar, bestehend aus einem privaten und einem öffentlichen Schlüssel. Der private Schlüssel ist dabei nur dem Empfänger und der öffentliche Schlüssel möglicherweise vielen anderen Teilnehmern bekannt[8].
Ein Netz, in dem n Teilnehmer untereinander vertraulich Nachrichten empfangen und senden können, benötigt demzufolge nur n Schlüsselpaare, die auch über einen unsicheren Kanal (aber: Achtung vor Mann-in-der-Mitte-Angriffen; s.Kap. 5.2.5) ausgetauscht werden können. Dem gegenüber steht eine relativ große Schlüssellänge, die mit besseren Algorithmen und schnelleren Prozessoren[9] entsprechend wachsen muss, sowie ein hoher Rechenaufwand (2 bis 3 Größenordnungen ineffizienter als symmetrische Verfahren [Pf00]) für Ver- und Entschlüsselungsoperationen.
Vertreter für asymmetrische Konzelationssysteme wären z.B. RSA (benannt nach seinen

[7] Als "schwer" kategorisierte Probleme liegen aller Wahrscheinlichkeit nach in NP (nichtdeterministisch polynomial). NP bedeutet im Prinzip: eine geratene Lösung kann schnell (polynomial) überprüft werden. Der interessierte Leser sei im Zusammenhang von Komplexitätstheorie und Kryptografie auf [Pf00] verwiesen.

[8] Es sind allerdings auch verschiedene andere Spielregeln denkbar. Z.B. Austausch des Geheimnisses mit einer zweiten Person für symmetrische Kryptografie.

[9] Man muss darauf achten, dass mit aktuellem Kenntnisstand von Algorithmen und der verfügbaren Hardware kein vollständiges Durchsuchen des Schlüsselraumes (Testen leicht!) möglich ist.

Erfindern Rivest, Shamir und Adleman) oder ein System mit s^2-*mod-n*-Generator.

Bei asymmetrischen Authentikationssystemen spricht man auch von *digitalen Signatursystemen*. Ihr Hauptvorteil besteht darin, dass ein Teilnehmer A für eine von Teilnehmer B gesendete und signierte Nachricht auch vor einem Dritten (vorausgesetzt dieser verfügt ebenfalls über den zugehörigen Testschlüssel von B) beweisen kann, dass sie tatsächlich von B verfasst wurde. Man spricht daher auch von einer *digitalen Unterschrift*. Vertreter hierfür wären RSA, DSA oder GMR (benannt nach seinen Erfindern Goldwasser, Micali und Rivest).
Grundsätzlich lässt sich auch hier sagen, dass jedes asymmetrische Konzelationssystem auch als digitales Signatursystem und umgekehrt verwendet werden kann [Pf00].

2.3.3 Hybride Systeme

Um die Vorteile der symmetrischen und asymmetrischen Kryptografie ausnutzen und die Nachteile möglichst gering halten zu können werden in der Praxis häufig so genannte *hybride Systeme* eingesetzt.
Dabei werden asymmetrische Konzelationssysteme nur dazu verwendet, den Schlüssel eines symmetrischen Konzelations- oder Signatursystems vertraulich an den gewünschten Kommunikationspartner zu übertragen. Mit Hilfe dieses Schlüssels können dann die eigentlichen Nutzdaten wesentlich effizienter ver- und entschlüsselt bzw. signiert und geprüft werden.
Damit wird das bequeme Schlüsselmanagement von asymmetrischen Systemen mit der Effizienz von symmetrischen Systemen verbunden.
Das entstandene hybride kryptografische System kann im Ergebnis dann natürlich nur so stark sein, wie das schwächste zu seiner Bildung verwendete.

2.3.4 Schlüsselaustausch

Der Schlüsselaustausch ist der wichtigste und gleichzeitig der gefährlichste Schritt für den Aufbau einer vertraulichen Kommunikationsbeziehung. Ohne ihn kommt diese Beziehung nicht zu Stande und bei einem Fehler dabei (es werden z.B. der Schlüssel selbst oder nicht für die Öffentlichkeit gedachte Informationen über ihn bekannt) kann der Inhalt der als vertraulich eingestuften Konversation teilweise oder sogar komplett in falsche Hände geraten.
Es ergeben sich also folgende Möglichkeiten für einen vertraulichen Schlüsselaustausch:

- Die Kommunikationspartner treffen sich persönlich und tauschen dabei Datenträger mit den Schlüsseln symmetrischer oder asymmetrischer kryptografischer Systeme untereinander aus.

- Bezug der öffentlichen Schlüssel asymmetrischer Systeme von einer öffentlich zugänglichen Stelle (z.B. dem Internet). Für den Beweis der Authentizität des Schlüssels muss eine entsprechend vertrauenswürdige Signatur (s.Kap. 2.3.5) über den Schlüssel selbst und die Information, wem er zuzuordnen ist, vorhanden sein.

2.3.5 Signaturen

Signaturen stellen ein Hilfsmittel zur Sicherstellung von Datenintegrität und Authentizität des Senders dar. Jede Veränderung an den übermittelten Daten soll dabei mit sehr hoher Wahrscheinlichkeit festgestellt werden können.

Bei einem symmetrischen Signatursystemen sind beide Kommunikationspartner in der Lage gültige Signaturen zu erstellen. Der Nutzen dieser Art von Signatursystemen beschränkt sich daher auf die bloße Wahrung der Datenintegrität und den Beweis der Authentizität des Senders einzig und allein gegenüber dem Empfänger.

Mit Hilfe asymmetrischer Signatursysteme können so genannte *digitale Signaturen* erstellt werden, welche ebenfalls Datenintegrität und Sendeauthentizität gegenüber dem Empfänger sicherstellen.
Weiterhin ermöglichen sie es, auf Grund der Tatsache, dass nur der Sender einer Nachricht in der Lage ist, eine korrekte Signatur zu erstellen, vor einer dritten Partei die Herkunft einer Nachricht zu beweisen. Der Testschlüssel für eine Signatur kann/muss dabei öffentlich zur Verfügung gestellt werden. Dies ermöglicht es *jedem* (sofern er im Besitz des zugehörigen Testschlüssels ist und weiß, wem dieser gehört) zu überprüfen, ob eine konkrete Nachricht von einer bestimmten Person verfasst worden ist.
Der Besitzer des Signierschlüssels kann den Versand einer Nachricht also nicht mehr leugnen, wenn der Test der zugehörige Signatur mit einem dem Dritten (z.B. einem Gericht) ebenfalls bekannten zugehörigen öffentlichen Testschlüssel positiv ausfällt.
Die digitale Signatur kann somit in Softwaresystemen auch als Kriterium für die Identifikation eines Nutzers verwendet werden.

2.3.6 Hash-Funktionen

Eine Hash-Funktion ist eine Abbildungsfunktion, welche eine große Eingabe (z.B. eine Klartextnachricht) auf eine kleinere Ausgabe (den so genannten Hash) mit meist festgelegter Größe abbildet.
Eine wichtige Anforderung an diese Funktion ist, dass sie ihren Urbildraum (alle möglichen Eingaben) in etwa gleichmäßig auf ihren Bildraum (alle möglichen Ausgaben) abbildet.
Moderne Hash-Funktionen werden auch als "one-way"-Funktionen bezeichnet, da es möglichst schwer sein soll, zu einem vorgegebenen Hash eine Eingabe zu finden, die durch die selbe Hash-Funktion wieder auf diesen Hash abgebildet würde. Weiterhin soll es sehr schwer sein, zu einer vorgegebenen Eingabe (z.B. einer Textnachricht in deutsch) und ihrem Hash eine andere sinnvolle Eingabe zu finden, welche den selben Hash besitzt.

Der Berechnungsaufwand für solche Hash-Funktionen ist generell geringer als der für digitale Signaturen. Daher werden diese verkürzenden Hashfunktionen in der Kryptografie eingesetzt, um den Aufwand für das Erstellen von digitalen Signaturen auch bei großen Texten in etwa konstant zu halten. Dazu wird über der Nachricht ein Hash berechnet, dessen digitale Signatur dann an die Nachricht angehangen wird. Bei der Überprüfung der Authentizität der Nachricht berechnet der Empfänger mit einer festgelegten Hash-Funktion den Hash dieser Nachricht und überprüft dann die Signatur.
Beispiele für Hash-Funktionen sind MD5 oder SHA.

2.4 Kanäle

Ein Kanal ist im Zusammenhang mit der Informationstechnologie eine Verbindung zwischen zwei Knoten, über welche miteinander, unter Einhaltung bestimmter Anforderungen, kommuniziert werden kann. Knoten sind dabei Computersysteme, welche einen Kanal entweder *physikalisch* (z.B. direkte Verbindung über ein Kabel) oder *logisch* zueinander herstellen können. Da es in der Welt globalen Vernetzung immer seltener vorkommt, dass zwei System direkt durch ein Kabel miteinander verbunden werden können, gewinnen logische Kanäle

immer mehr an Bedeutung.
Zur Installation eines logischen Kanals wird grundsätzlich eine Kanalaufbaunachricht von der anfragenden an die gerufene Partei übermittelt, in welcher verschiedene Eigenschaften des aufzubauenden Kanals übertragen werden. Diese Eigenschaften können z.B. die gewünschte Breite des Kanals (bit/s), bestimmte Sicherheitsparameter oder ähnliches beschreiben.
Zum Aufbau eines *sicheren Kanals* wird, falls nicht bereits durch das verwendete Protokoll vorgeschrieben, eine Methode des Schlüsselaustauschs vereinbart (z.B. Zertifikate oder Diffie-Hellmann). Mit Hilfe des ausgehandelten asymmetrischen Schlüssels wird anschließend üblicherweise aus Gründen der Performance ein Schlüssel für ein symmetrisches kryptografisches System ausgetauscht, mit dem dann die eigentlichen Kommunikationsdaten gegen Abhören oder Verändern geschützt werden können.

2.5 Asymmetrische Kryptografie standardisieren - *PKCS*

Die so genannten PKCS (Public-Key Cryptography Standards) sind eine Sammlung von Spezifikationen um die Entwicklung von asymmetrischen kryptografischen Systemen zu beschleunigen. Ausgearbeitet werden diese seit 1991 durch die RSA-Laboratories [RSALab] der RSA-Security Gruppe und ein Vielzahl von Entwicklern, welche sich mit der Sicherheit von Softwaresystemen beschäftigen.
Heutzutage befinden sich Beiträge aus der PKCS-Reihe (erkennbar an der Namensgebung "PKCS#X", wobei das X für die Nummer, derzeit 1 bis 15, der jeweiligen Spezifikation steht) in einer Vielzahl von ausgereiften Produkten und Industriestandards (z.B. SSL, S/MIME).
Die beiden Spezifikation #7 und #10 sollen auf Grund ihrer Wichtigkeit für diese Arbeit kurz vorgestellt werden:

PKCS#7 Der *Cryptographic Message Syntax Standard* beschreibt, wie die Syntax von Daten aussehen sollte, damit diese kryptografischen Operationen unterzogen werden kann (z.B. verschlüsseln oder digital signieren).

PKCS#10 Der *Certification Request Syntax Standard* beschreibt, wie die Syntax einer Zertifikats-Anforderung aussehen sollte. Dazu gehört der Name oder die Kennzeichnung der anfragenden Partei und eine Menge weiterer Attribute.

2.6 Kanonisation

Die Kanonisation (engl. "canonicalization"; auch bekannt unter der Bezeichnung "c14n") wird in vielen Bereichen der Informatik angewendet (Progammierung, Datendarstellung, Algorithmen etc.).
In dieser Arbeit bezieht sich die Kanonisation als Vorgang einzig und allein auf die Art und Weise der Darstellung von Daten im XML-Format (s.Kap. 3.1). Dabei werden syntaktisch äquivalente Daten in abweichender Darstellungsform in ein Format gebracht, in welchem sowohl ihre Syntax als auch ihre Darstellung identisch sind.
Betrachtet man zwei syntaktisch identische XML-Dokumente mit einer unterschiedlichen Darstellung ergeben sich folgende mögliche Unterschiede:

1. unterschiedliche Reihenfolge von Attributen
2. unterschiedliche Verwendung von Whitespaces
3. unterschiedliches Format von Zeilenumbrüchen (z.B. Windows vs. Unix)

Obwohl beide Dokumente durchaus korrektes XML enthalten können, würden sich die berechneten Hashwerte mit sehr hoher Wahrscheinlichkeit unterscheiden (und wenn nicht, hätte man gleich einen Grund, die Sicherheit des Hash-Algorithmus anzuzweifeln). Um also ein digitales Signatursystem, welches auf einer Hash-Funktion basiert, zu konstruieren muss man sicherstellen, dass die Darstellung eines XML-Dokuments vor jeder Ausführung einer Hash-Funktion identisch ist. Da oben genannte Unterschiedsmerkmale die Korrektheit eines XML-Dokuments nicht beeinträchtigen und gerade bei der Kommunikation über weitere XML verarbeitende Knoten (so genannte "Intermediaries") die Möglichkeit einer unvorhersehbaren Umformatierung nicht vernachlässigt werden kann, muss dafür eine Lösung gefunden werden.
Die erste Kanonisationsmethode (Whitspaces, Attribut-Reihenfolge und Zeilenumbrüche standardisieren), welche sich dieser Problematik annahm, war die "inclusive canonicalization". Diese Methode hatte das Problem, die Namensraum-Deklarationen mit einzubeziehen. Wurde das signierte Element in ein anderes XML-Dokument mit anderen Namensräumen eingefügt, so wurden diese bei der Berechnung des Hashs mit einbezogen und führten zu falschen Ergebnissen.
Da das Einsetzen von XML-Signaturen in andere XML-Dokumente in der Welt von SOAP (in der Regel unzählige eigene Namensraum-Deklarationen) einen gewöhnlichen Vorgang darstellt, wurde die "exclusive canonicalization" erfunden. Mit deren Hilfe wurde dieses Problem behoben und es wird jedem Nutzer von XML-Kanonisation nahegelegt, diesen Algorithmus in seinen Anwendungen zu verwenden [On02].

2.7 Digitale Zertifikate

Ein digitales Zertifikat kann mit einem Reisepass verglichen werden. Es enthält neben dem öffentlichen Schlüssel eines asymmetrischen kryptografischen Systems typischerweise den Namen der Entität (CN:), den Namen der zugehörigen Organisation (O:), den Ländercode (C:) und eventuell weitere Eigenschaften des Zertifikatsgegenstandes.
Den eigentlich wichtigsten Bestandteil eines digitalen Zertifikates stellt dessen Legitimation, sozusagen der Stempel der ausstellenden Behörde, als Beweis der Echtheit und der Richtigkeit der enthaltenen Informationen, dar.
In der digitalen Welt erfolgt dies mit Hilfe von digitalen Signaturen (s.Kap. 2.3.5). Der Signierende bezeugt damit die Authentizität des Zertifikatsgegenstandes und die Korrektheit der enthaltenen Daten.
Dadurch ergibt sich folgendes Problem der Vertrauenswürdigkeit von solchen Signaturen: Wessen Signatur ist notwendig, um für den jeweiligen Verwendungsfall zu beweisen, dass das angegebene Zertifikat tatsächlich der Entität X gehört, wie behauptet?
Die Lösung für dieses Dilemma ist die Einrichtung so genannter PKIs (s.Kap. 2.8). Konkrete Instanzen werden auch gemeinhin als *CAs* (Certificate Authorities) bekannt sind. Jedem von diesen Instanzen signierten Zertifikat kann so viel Vertrauen entgegen gebracht werden, wie man bereit ist dieser Instanz selbst zu vertrauen. Dies kann von kleinen PKI-Infrastrukturen für wenige Anwender bis hin zu globalen CAs (z.B. [Verisign, Thawte]) reichen.
Mit Hilfe eines als vertrauenswürdig eingestuften Zertifikats ist es dann möglich, mit dessen Inhaber eine kryptografisch untermauerte Kommunikationsbeziehung aufzubauen.
Im Folgenden sollen zwei Standards für den Aufbau von digitalen Zertifikaten näher vorgestellt werden. Der Standard X.509 (s.Kap. 2.7.1) wird dann im weiteren Verlauf dieser Arbeit als Grundlage für alle verwendeten Zertifikate dienen.
Um nicht dem Eindruck zu unterliegen, es gäbe nur einen Spezifikation für digitale Zertifi-

kate, wird im Anschluss daran PGP (s.Kap. 2.7.2) vorgestellt.

2.7.1 X.509

X.509 ist ein Zertifikat-Format welches hauptsächlich für die Verwendung in Protokollen der OSI-Familie (Open System Interconnect) vorgesehen war. Ungeachtet des begrenzten Erfolgs der OSI-Protokolle stellen die X.509-Zertifkate heute die Grundlage der meisten PKIs (s.Kap. 2.8) dar. Dieser Standard befindet sich zur Zeit in der Version 3 und entsprechende Zertifikate enthalten folgende Informationen:

Version: Die Version des verwendeten X.509-Standards.

Serielle Nummer: Der Aussteller des Zertifikats ist dafür verantwortlich, eine für sich eindeutige Nummer an jedes erstellte Zertifikat zu vergeben. Dies ermöglicht ihm das einfachere Auffinden bei Anfragen und das leichtere Verwalten der so genannten CRL (Certificate Revocation List). Diese beinhaltet die Nummern aller als ungültig markierten Zertifikate.

Signatur Algorithmus Identifikator: Beinhaltet einen Identifikator, mit dessen Hilfe festgehalten wird, welcher Algorithmus durch die CA für die Signatur verwendet wurde.

Name des Ausstellers: Der X.500[10] konforme Name des Ausstellers. Normalerweise ist dies eine CA, aber prinzipiell können mit jedem beliebigen Zertifikat andere Zertifikate signiert werden.

Gültigkeitszeitraum: Jedes Zertifikate besitzt eine bestimmte maximale zeitliche Gültigkeit, definiert durch einen Start- und einen Endzeitpunkt. Nach Ablauf dieser Zeitspanne wird das Zertifikat automatisch auf die CRL gesetzt. Der eingestellt Gültigkeitszeitraum hängt z.B. von der Stärke des zum enthaltenen öffentlichen Schlüssel gehörenden asymmetrischen Algorithmus, der Länge des privaten Schlüssels oder finanziellen Beschränkungen ab. Bei einer Kompromittierung des privaten Schlüssels oder einem anderen Einfluss kann dieser Gültigkeitszeitraum im Rahmen einer PKI-Infrastruktur jederzeit verkürzt werden.

Name des Subjekts: Name der Entität, deren im Zertifikat enthaltener öffentlicher Schlüssel durch dieses Zertifikat als zugehörig beglaubigt wurde. Das Format des Namens folgt ebenfalls der X.500 Spezifikation.

Informationen über den öffentlichen Schlüssel des Subjekts: Enthält den öffentlichen Schlüssel der namentlich genannten Entität, Informationen darüber, mit welchem asymmetrischen kryptografischen System dieser angewendet werden soll und welche zusätzlichen Parameter für die Verwendung benötigt werden.

Erweiterungen: Zusätzliche Attribute, welche mit dem Zertifikats-Besitzer (Subjekt) verbunden werden können.

Zusätzlich zu den eben genannten Daten enthält eine Zertifikat als seinen wichtigsten Bestandteil eine durch jeden Nutzer selbst zu überprüfende Signatur des Ausstellers.
Alle Daten innerhalb des Zertifikats werden mit Hilfe von ASN.1 (s.Kap. 2.2) und DER (Definite Encoding Rule) kodiert. DER beschreibt eine Art und Weise für das Abspeichern und Übertragen der Zertifikatsdaten.

[10] Eindeutiger Name der Entität (Distinguished Name) enthält den Namen (CN), die Organisations-Einheit (OU), die Organisation (O) und das Land (C).

2.7.2 PGP

PGP (Pretty Good Privacy) ist eine Modell-Entwicklung von Phil Zimmermann [Zi05] als Reaktion auf die aus seiner Sicht unnötig komplexen und autoritären Vorgehensweisen, welche benötigt werden, um eine CA zu formen.
Im PGP-Modell darf jeder Nutzer Zertifikate ausstellen und beliebige andere unterzeichnen (signieren). Im Laufe der Zeit entsteht durch die gegenseitigen Signaturen innerhalb einer Community von PGP-Nutzern ein so genanntes "web of trust" (Netz des Vertrauens).
Die Struktur der Zertifikate folgt dabei der X.509-Spezifikation.

2.8 Öffentliche Schlüssel verwalten - *PKI*

Wie bereits bei der Vorstellung von Zertifikaten in Kapitel 2.7 kurz angeschnitten, handelt es sich bei PKI (Public Key Infrastructure) um eine Infrastruktur, welche es ermöglichen soll, die vertrauenswürdige Verwendung der flexiblen und anpassungsfähigen Technologien für digitalen Signaturen und für asymmetrische Verschlüsselung der breiten Öffentlichkeit zugänglich zu machen.
Der Erste, der Lösungsvorschläge für das Problem der Verteilung von öffentlichen Schlüsseln publik machte, war Whitfield Diffie [Di05]. Sein Vorschlag war es, ein zentrales Verzeichnis bereitzustellen, aus dem sich jeder Interessierte den öffentlichen Schlüssel seines gewünschten Kommunikationspartners herunterladen konnte. Die Einfachheit dieses Prinzips überzeugte, jedoch wurde bald festgestellt, dass das Internet zu dieser Zeit noch lange nicht für jeden permanent verfügbar war und auch nicht über die nötige Zuverlässigkeit verfügte. Eine Lösung für diesen Nachteil brachten die Überlegungen von Lauren Kohnfelder [Ko78]. Die Idee waren Zertifikate, welche durch die System-Betreiber verifiziert (signiert) werden. Diese können völlig unabhängig von einem zentralem Verzeichnis verteilt werden. Die Idee einer PKI war geboren.
Eine PKI verwaltet und beglaubigt öffentliche Schlüssel von Entitäten (Personen, Unternehmen, Maschinen, Email-Adressen etc.) und macht diese verfügbar. Dadurch wird es ermöglicht, eine Identität aus der realen Welt mit einer online-Identität zu verbinden. Obwohl diese Anforderung nicht sehr aufwendig scheint, so hat doch die Vergangenheit gezeigt, das bei der Umsetzung vielfältige Probleme auftraten. Eines davon ist die vielfach unterschätzte Komplexität des Verwaltungsaufwandes. Ein Name kann z.B. in der realen Welt durchaus mehreren Personen zugeordnet sein und eine reale Person kann prinzipiell über beliebig viele Namen (eq. online-Identitäten) verfügen.
In der modernen digitalen Welt, in welcher man sich, auf Grund der Vielzahl der Nutzer, nicht mehr persönlich treffen kann (bzw. will), um vertrauliche Informationen auszutauschen, ist es sehr wichtig über einen Mechanismus zu verfügen, welcher es einem ermöglicht, der Identität seines Gegenüber in dem Umfang zu vertrauen, wie es für die gewünschte Aktion notwendig ist.
Diese Aufgabe übernehmen Anbieter einer PKI. Die wichtigste Anforderung an diese ist die eindeutige Überprüfung einer Identität und die Feststellung, dass die Entität auch tatsächlich über den zum öffentlichen Schlüssel passenden privaten Schlüssel verfügt. Ein erfolgreicher Abschluss dieses Protokolls führt zur Erstellung eines Zertifikats, mit welchem der PKI-Provider die Identität des Zertifikatsgegenstandes bezeugt.
Die Umsetzung einer PKI muss allerdings auch in der Lage sein, Zertifikate zu widerrufen. Dies kann notwendig werden, wenn ein privater Schlüssel in irgend einer Art und Weise kompromittiert wird, sich herausstellt, dass eine Information im Zertifikat falsch ist bzw. ihre Gültigkeit verloren hat oder wenn der Schlüsselbesitzer gegen Regeln des PKI-Providers verstößt.

Das eigene Vertrauen in ein solches Zertifikat kann nur so groß sein, wie man der jeweiligen CA selbst vertraut. Die "richtige CA" variiert somit von Fall zu Fall und hängt von der jeweiligen Verwendung ab. Wenn man sich über die Identität bzw. das Zertifikat einer Person X, welche man noch nie getroffen hat, sicher sein will, wäre eine Signatur einer vertrauenswürdigen globalen CA sinnvoll, wohingegen bei einer Identitätsprüfung des firmeninternen Mailservers ein Zertifikat mit Signatur der Firmen-CA ausreichend sein sollte (und im Gegensatz zu Verisign, Thawte und Co nichts kostet).
Mit Hilfe jeden Zertifikats können prinzipiell beliebig viele weitere Zertifikate unterschrieben werden, wodurch ein so genannter *Zertifizierungsbaum* (s.Abb. 2) aufgespannt wird. Das Vertrauen (V) in ein beliebiges Zertifikat in diesem Baum sollte allerdings nur dem

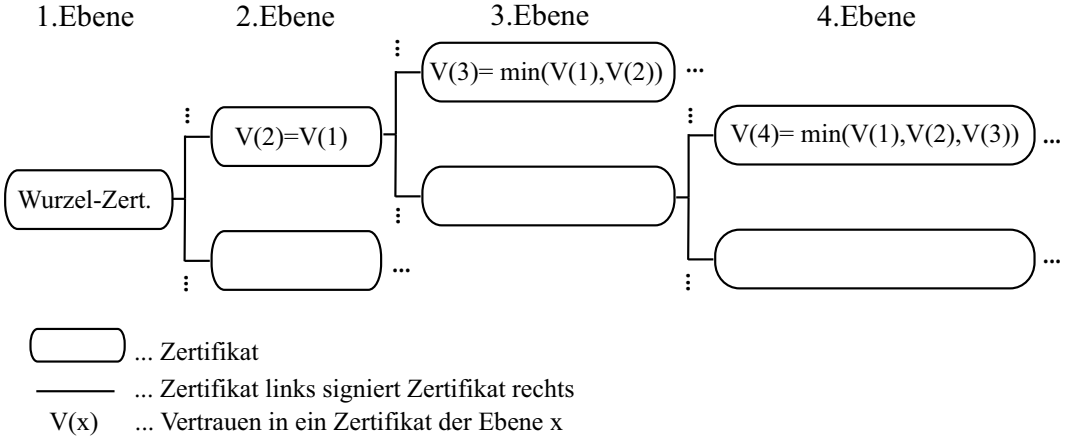


Abbildung 2: Zertifizierungsbaum mit Vertrauensabhängigkeiten

geringsten ($min()$) entgegengebrachten Vertrauen aller Zertifikatsinhaber auf dem Pfad zur Baumwurzel entsprechen.
Die Größe V lässt sich natürlich nicht in Zahlen ausdrücken und muss anhand einer Vielzahl von Kriterien subjektiv bewertet werden.
Am Rande sei auch die Möglichkeit eines Zertifizierungsnetzes (PGP-Prinzip; s.Kap. 2.7.2) erwähnt, da hier keine CA verwendet wird. Bei diesem Prinzip existiert keine strenge Hierarchie, wie in Abbildung 2 gezeigt und eine Vertrauenseinstufung erfolgt hier anhand der Information, ob eine vertrauenswürdig eingestufte Entität ein Zertifikat direkt unterschrieben hat, oder ob wenigstens ein "vertrauenswürdiger" hierarchischer Bezug zu ihr besteht. Ein hierarchischer Bezug entsteht in diesem Fall, wenn man sich anschaut, von wem zur Signatur verwendete Zertifikate signiert wurden und dies immer weiter zurückverfolgt (es entsteht ein Zertifizierungsbaum). Erlangt man Informationen über die "Zwischensignierer" kann man anhand derer unter Umständen eine eigene Vertrauenseinschätzung abgeben, welche vielleicht für den einen oder anderen Anwendungsfall genügt.

3 Webservice - Basistechnologien

Im den nun folgenden Kapiteln sollen Technologien, welche im Zusammenhang mit Webservices verwendet werden, aber noch keinerlei direkten Bezug zu Sicherheitsaspekten beinhalten, vorgestellt werden. Spezifikationen, welche für die Bearbeitung der einzelnen Teilaufgaben dieser Diplomarbeit relevant sind, werden in den folgenden Unterkapiteln ausführlicher und detailierter als andere beschrieben.

Auf Grund des derzeitigen Engagements vieler Organisationen, Unternehmen und Einzelpersonen im Bereich Web Service und der damit verbundenen permanenten Neuvorstellung von Protokollen, Spezifikationen etc., wird dabei keinerlei Anspruch auf Vollständigkeit erhoben.

3.1 Nachrichtenformat - *XML und XSD*

XML (eXtensible Markup Language) [XML] und XSD (XML Schema Definition) [XSD] werden durch das World Wide Web Consortium [W3C] entwickelt und stellen die Grundbausteine einer jeden auf Webservices beruhenden Architektur dar. Sämtliche Standards für Kommunikationsprotokolle und Beschreibungssprachen und der weitaus größte Teil aller weiteren Produkte rund um Webservices basieren auf dieser Art und Weise Daten darzustellen.
Da die Beschreibung aller Möglichkeiten von XML bzw. XSD den Rahmen dieser Arbeit sprengen würde und ausreichend umfangreiche sowie leicht verständliche Literatur (z.B. [Mc01]) über dieses Thema verfügbar ist, werden nur die wichtigsten, für das Verständnis der weiteren Arbeit grundlegend notwendigen Notations- und Ausdrucksmöglichkeiten erklärt.

XML ist ein textbasiertes Datenformat und wurde in seiner ersten Version 1998 veröffentlicht. Zur Zeit existieren zwei auf Grund unterschiedlicher Behandlung spezieller Zeichen *inkompatible* Version nebeneinander (XML 1.0 Third Edition und XML 1.1).
Mit Hilfe von XML werden Daten hierarchisch strukturiert und mit Bezeichnerelementen bzw. -attributen versehen.

Beispiel 1:
```
<Person Vorname="Mirko" Nachname="Richter"/>
```

Das Beispiel zeigt ein Element "Person" mit seinen Attributen "Vorname" und "Nachname". Das Zeichen "/" (hier als vorlesztes Symbol) kennzeichnet das Ende des Elements. Das Element im Beispiel besitzt nur Attribute und keinen Inhalt. Falls Inhalt verwendet werden soll, teilt sich ein Element in ein Beginn- und ein End-Tag[11] auf:

Beispiel 2:
```
<Person Vorname="Mirko" Nachname="Richter">
  <!-- INHALT -->
</Person>
```

Ein Tag, dessen Inhalt mit einem "/" beginnt und dem sich der Elementnamen anschließt, kennzeichnet neben der Notation im Beispiel 1 ebenfalls das Ende eines Elements. Jedes XML-Element muss auf eine der beiden Arten abgeschlossen sein. Der Schriftzug "INHALT" wird im Beispiel auf Grund der umgebenden Zeichengruppen "<!–" und "–>" als Kommentar behandelt.
Einzelne Elemente können beliebig ineinander verschachtelt sein. Dazu könnte z.B. der

[11] Ein *Tag* ist eine Zeichenkette beginnend mit "<" und endend mit dem nächsten Auftreten von ">".

platzhaltende Kommentar im Beispiel 2 durch ein XML-Element ersetzt werden.

Im Gegensatz zu vielen anderen (binären) Kommunikationsprotokollen hat diese Struktur die Eigenschaft, für den Menschen lesbar zu bleiben[12]. Die selbstbeschreibenden Element- und Attributnamen wirken sich allerdings nachteilig auf die Dokumentlänge aus.

Der XSD-Standard wurde 2001 vom W3C freigegeben. Ein XSD-Dokument beschreibt die Struktur eines XML-Dokuments[13] und folgt selbst den Darstellungsregeln von XML.
Ein XSD-Dokument beschreibt, welche Elemente in einem XML-Dokument auftauchen, welche Attribute sie enthalten und wie sie geschachtelt sein dürfen/müssen. Weiterhin wird festgelegt, ob ein Element bzw. Attribut leer sein kann, welche Datentypen an dieser Stelle erlaubt sind und welcher Wert als Standard-Wert verwendet wird, falls keine anderen Angaben im XML-Dokument gemacht werden.
Für den Fall, dass in einem XML-Dokument mehrere XSDs verwendet werden sollen, wurde das Konzept der Namensräume eingeführt. Entsprechend deklariert, wird ein Element oder ein Attribut durch voranstellen des jeweiligen im Dokument selbst definierten Prefixes als zugehörig und der jeweiligen XSD folgend markiert.

Beispiel 3: `<human:Person [...] xmlns:human="http://diplom.compago.de/h"/>`

Die Definition des Elements "Person" befindet sich mit allen seinen Attributen im Namensraum "http://diplom.compago.de/h", welcher durch das Prefix "human" referenziert wird.

Die für die weitere Arbeit verwendete Notation zur Darstellung von XML-Schemata orientiert sich an der in der XML-Spy IDE [XSpy] verwendeten grafischen Notation. Abbildung 3 stellt die wichtigsten verwendeten Notationselemente kurz vor.

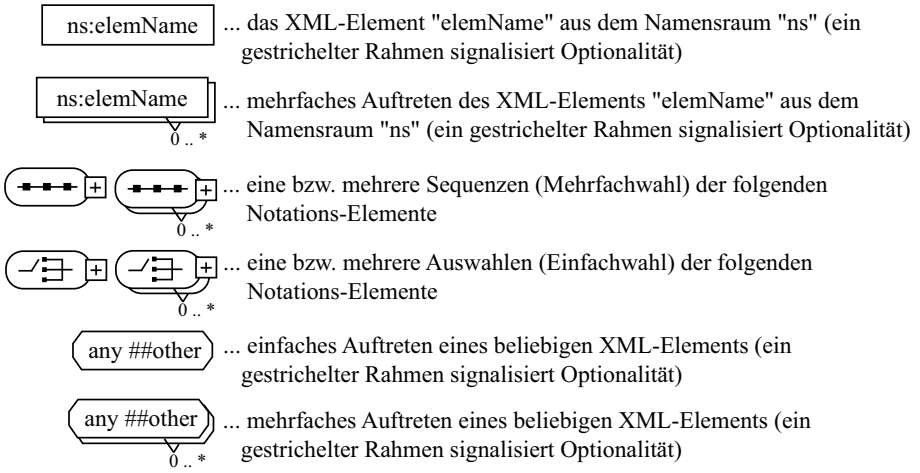


Abbildung 3: Legende für verwendete grafische XSD-Notation (XML-Spy)

Die resultierenden XSD-Grafiken werden "von links nach rechts gelesen"[14], wobei durch-

[12] Zumindest in den meisten Fällen. Schlecht gewählte Element- und Attribut-Namen sowie große Mengen von unterschiedlichen Namensräumen wirken diesem Vorteil entgegen.

[13] Auf die Strukturbeschreibungssprache DTD (Data Type Definition) [DTD] wird an dieser Stelle nicht weiter eingegangen, da sie weniger aussagekräftig ist, selbst kein valides XML darstellt und in dieser Arbeit keine Verwendung findet.

[14] Bedeutet: bei durch Linien verbundenen Grafikelementen stellen die weiter links stehenden Elemente übergeordnete Strukturen dar.

gezeichnete Linien notwendige und gestrichelte Linien optionale Verbindungen zwischen Einfach- bzw. Mehrfachauswahlen und XML-Elementen darstellen.

3.2 Transportprotokoll - *SOAP*

Die SOAP-Spezifikation [SOAP] wird ebenfalls durch das World Wide Web Consortium [W3C] entwickelt, basiert offiziell auf XML 1.0 und befindet sich seit Juni 2003 in der Version 1.2. Es definiert ein Rahmenwerk für die Übertragung von XML-Nachrichten über ein beliebiges anderes Protokoll (z.B. HTTP(S), SMTP etc.), welches allerdings in der Lage sein muss angemessen große Nutzdaten zu transportieren. Dadurch wird SOAP völlig protokollunabhängig und kann dieselben Inhalte in verschiedene Host-Protokolle verpacken.
Durch dieses Huckepack-Verfahren entsteht allerdings das Problem (oder auch der Vorteil), dass SOAP überall dort zur Anwendung kommen kann, wo das Host-Protokoll akzeptiert wird. Im Falle von z.B. HTTP führt dies dazu, dass herkömmliche Firewall-Mechanismen ("Port x blockieren") bei SOAP nicht mehr verwendet werden können, da Port 80 auf Grund angebotener HTTP-Services in den meisten Fällen als "offen" konfiguriert ist. Es müssen neue Vorgehensweisen erarbeitet werden (z.B. Überprüfung von SOAP-Nachrichten gegen eine XSD; jedoch für verschiedene Angriffstypen nicht ausreichend!). Über die Vor- und Nachteile dieser Festlegung wird in der Literatur und im Internet viel diskutiert (z.B [Ru02, JZ03]).
SOAP selbst ist auch ein vollständiges Protokoll (s.Abb. 4) mit einem Nachrichtenkopf (header) und einem Nachrichtenkörper (body). So genannte "Intermediaries" (Knoten, welche

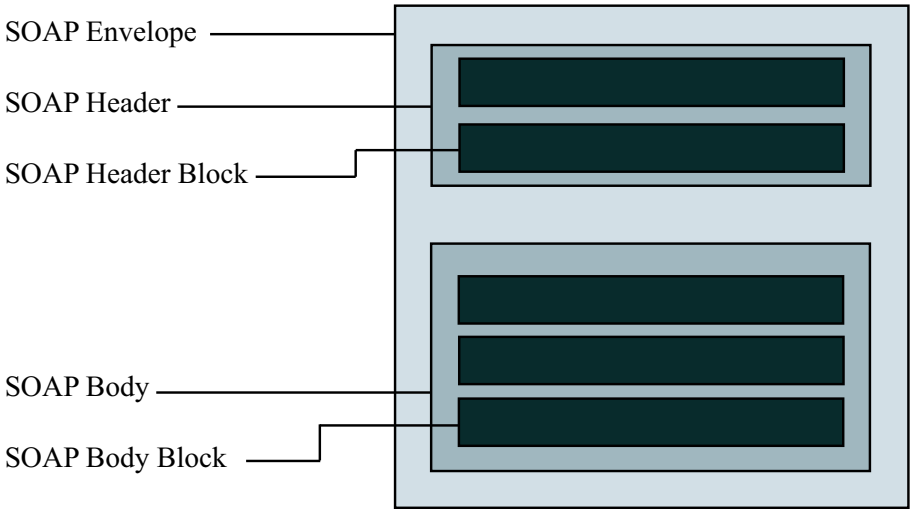


Abbildung 4: Struktur einer SOAP-Nachricht

Operationen auf der SOAP-Nachricht ausführen) und der Empfänger selbst werten den SOAP-Header wie bei jedem anderen Protokoll aus.

3.3 Host-Transportprotokoll - *HTTP(S), SMTP und Co*

Wie bereits in Kapitel 3.2 über SOAP erwähnt, kann als Host-Transportprotokoll prinzipiell jedes beliebige andere Protokoll verwendet werden, wenn es nur genügend Nutzdaten transportieren kann. Die dabei durch den OSI-Stack entstehende Verschachtelung wird in Abb.

5 angedeutet. In der Praxis haben sich allerdings die standardisierten Protokolle HTTP

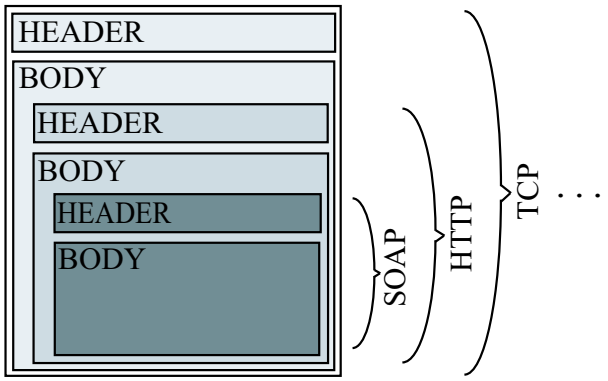


Abbildung 5: SOAP-Nachricht im OSI-Protokoll-Stack

(Hypertext Transfer Protocol), HTTPS (HTTP Secure) und SMTP (Simple Mail Transfer Protocol) für den Transport von SOAP-Nachrichten durchgesetzt.
Betrachtet man den OSI-Protocol-Stack [On02] stellt sich die Frage, auf welcher OSI-Schicht SOAP einzuordnen ist, wenn bereits HTTP(S) und SMTP auf Schicht 7 angesiedelt sind. Man könnte durchaus der Meinung sein, eine 8. Schicht einführen zu müssen. Dem ist jedoch nicht so, denn SOAP ist per Definition genau wie HTTP und SMTP ein Dienst der Applikations-Schicht und somit ebenfalls auf der 7. Schicht angesiedelt.

3.4 Webservice Syntax-Beschreibung - *WSDL*

WSDL (Web Service Description Language) [WSDL], ebenfalls eine Spezifikation des World Wide Web Consortiums [W3C], ist seit März 2001 in der Version 1.1[15] verfügbar, unterstützt SOAP 1.1 und beschreibt die Schnittstelle, welche von einem konkreten Webservice angeboten wird. Der Inhalt eines WSDL-Dokuments ist selbst wieder XML 1.0 konform, welches gegen eine entsprechend XSD validiert werden kann.
Webservices werden in WSDL mit Hilfe der folgende sechs Hauptelemente beschrieben:

Types: beschreibt Datentypen, welche in den auzutauschenden Nachrichten verwendet werden.

Messages: repräsentiert eine abstrakte Beschreibung der auszutauschenden Nachricht. Ein solches Element besteht aus mehreren logischen Teilen (parts), wovon jedes mit einer Definition aus einem Typsystem assoziiert ist.

Port Types: beschreibt eine Menge von abstrakten Operationen. Jede dieser Operation verweist dabei auf die Definition einer Eingangsnachricht und mehrerer Ausgangsnachrichten (message).

Binding: definiert ein konkretes Protokoll und Datenformat-Spezifikationen für Operationen und Nachrichten eines einzelnen Port Types.

Port: spezifiziert die Adresse für ein Binding und somit einen einzelnen Kommunikations-Endpunkt.

[15] Version 2.0 befindet sich zur Zeit noch im Zustand "Working Draft".

Service: setzt eine Menge zusammengehöriger Ports in Relation zueinander.

3.5 Erweiterte Webservice-Beschreibung - *WS-Policy*

Das Web Service Policy Framework (WS-Policy) definiert eine Grundmenge von Konstrukten, welche durch Webservices benutzt und erweitert werden können, um ihre Anforderungen und Möglichkeiten zu beschreiben. Die Spezifikation wird von einer Kooperation namhafter Firmen, wie IBM, Microsoft, BEA, SAP, Sonic Software und VeriSign entwickelt.
Die durch eine Maschine verarbeitbaren Policy-Ausdrücke bestehen aus einer Kombination so genannter Assertions (Erklärung), welche durch global eindeutige Namen identifiziert werden. Dadurch wird es Services ermöglicht, sich z.B. zusätzlich zur WSDL-Beschreibung (nur Beschreibung der Syntax einer Schnittstelle) darüber zu informieren, wie ein Webservice seine Schnittstelle verfügbar macht (z.B. Qualitätsmerkmale) oder welche Anforderungen dieser an den Aufrufenden stellt (z.B. Welche Sicherheitsbedürfnisse hat der Service? Implementiert er transaktionale Modelle? Welche kryptografischen Algorithmen werden unterstützt? etc.).
Ohne WS-Policy und nur mit Hilfe von WSDL sind Sicherheitsaspekte, transaktionale Anforderungen, verlässlicher Nachrichtenaustausch (reliable messaging) und viele andere Webservice-Spezifikationen in der Praxis einer SOA nicht umsetzbar.
WS-Policy arbeitet eng mit den Subframeworks *WS-PolicyAttachment* [WSPolAtt] und *WS-PolicyAssertions* [WSPolAss] zusammen. WS-PolicyAttachment definiert, wie Policies mit XML-Nachrichten und WSDL-Dokumenten verbunden werden können und WS-PolicyAssertions unterstützt Interoperabilität, in dem es eine grundlegende Menge von Policy-Aussagen spezifiziert.

3.6 Vorgänge koordinieren - *WS-Coordination*

WS-Coordination ist ein allgemeiner Mechanismus für das Management von Kommunikationsprotokollen verteilter Anwendungen.
Bei einem koordinierten Ablauf übernimmt der Koordinator als zentrale Rolle allgemein die Aufgabe, bestimmte Informationen in einer bestimmten Art und Weise an registrierte Dienste weiterzuleiten. Für die meisten Anwendungen wird dafür ein speziell angepasstes Protokoll ausgearbeitet, welches die geforderten Aufgaben effizient ausführen soll.
Eine Spezifikation für ein standardisierten Protokolls, das allgemeingültige Festlegungen darüber trifft, welche Informationen wie weitergegeben werden, um ein generisches Koordinationsmodell zu formen, würde einen ineffizienten, schlecht skalierbaren und schwer überschaubaren Berg an abstrakten Festlegungen, Ausnahmebehandlungen usw. zur Folge haben [LW03].
Aus diesem Grund adressiert WS-Coordination nur die grundlegenden Gemeinsamkeiten aller Systeme, welche bestimmte Abläufe koordinieren, unabhängig davon ob sie z.B. Transaktionen oder Geschäftsabläufe verwalten:

- Aktivierung eines Koordinators für das spezifische Koordinationsprotokoll einer konkreten Anwendung.
  Die WS-Coordination Spezifikation sieht dafür einen Aktivierungsservice vor, welcher neue Koordinatoren für konkrete Protokolle erstellt. Dieser Vorgang ist asynchron definiert, was bedeutet, dass sowohl die Schnittstelle des Aktivierungsservice als auch die des aufrufenden Service als "Callback"-Schnittstelle in der Spezifikation definiert ist (WSDL).

- Verteilung von Kontextinformationen zwischen den Services, welche die Anwendung formen.
  Der Kontext, welcher durch den Koordinator erstellt wird, enthält als zentrale Stelle für den Prozess sämtliche notwendigen Informationen, um die eigentliche Anwendung zu formen. Der Kontext ermöglicht es den unterschiedlichen Services den gewünschten Zielprozess zur Erfüllung einer Aufgabe zu identifizieren und mit ihm zu interagieren. Die Kontextinformation muss zur korrekten Zuordnung aus allen zugehörigen übermittelten Nachrichten ableitbar sein.
  Der WS-Coordination-Kontext wurde so entwickelt, dass er auf der einen Seite bestimmte wichtige Daten enthalten muss und auf der anderen Seite flexibel genug ist, um sich den unterschiedlichsten Anwendungsfällen anzupassen. Er besteht dabei aus folgenden Informationen:

  – einem Koordinationsidentifikator mit globaler Eindeutigkeit für einen konkreten Koordinator in der Form einer URI
  – einer Adresse für den Endpunkt eines Registrationsservices, wo sich Parteien, welche den Kontext erhalten, für das Protokoll registrieren können
  – eine Angabe der maximalen Lebenszeit (time-to-live) zur Festlegung, bis wann der Kontext als gültig zu betrachten ist
  – erweiterbare protokollspezifische Informationen für den Ablauf des aufgesetzten konkreten Koordinationsprotokolls

- Registration von Teilnehmern beim Koordinator, damit diese notwendige Protokollnachrichten erhalten können.
  Wenn ein Koordinator erstellt und eine Kontext angelegt wurde, wird ein zugehöriger Registrationsservice veröffentlicht, der es interessierten Diensten ermöglicht sich für diesen Koordinationskontext anzumelden. Auch dieser Dienst ist asynchron, was eine WSDL-Beschreibung in der Spezifikation für Schnittstellen auf beiden Seiten zur Folge hat.
  Ein registrierter Service erhält alle durch den Koordinator verteilten Nachrichten (z.B. "prepare to commit", "commit" bei Zwei-Phasen-Transaktionsprotokollen).
  Wo es durch die konkrete Implementation unterstützt wird, können die registrierten Services auch Nachrichten an den Koordinator übermitteln.
  Diese Schnittstelle kann auch dafür verwendet werden, um sich über den Stand des koordinierten Prozesses nur zu informieren.

- Eine Entität, welche den koordinierten Prozess zum Abschluss führt.
  Diese Rolle wird von dem Klienten eingenommen, welcher an einer bestimmten Stelle dem Koordinator mitteilt, das Protokoll mit allen registrierten Teilnehmern *jetzt* auszuführen und somit zum Abschluss zu bringen. Dieser Klient wird, je nach Anwendungsfall mehr oder weniger detailliert, nach Ablauf des Protokolls vom Koordinator über den Ausgang der Aktivität informiert.

Die WS-Coordination Spezifikation [WSCoor] spricht dabei nur allgemein von (verteilten) Aktivitäten, welche in einer bestimmten Art und Weise spezifiziert, initiiert, angelegt, gestartet und wieder beendet werden.
Die Struktur eines WS-Coordination-Ablaufs lässt sich als Baum darstellen, in dem der zentrale Koordinator, welcher später das "Ausführen"-Kommando erhält, die Wurzel bildet und alle registrierten Services als "Blätter" dargestellt werden.

In erweiterten Enterprise-Anwendungen ist zudem vorstellbar, dass andere (lokale) Koordinatoren von globalen Koordinatoren wie normale Services angesprochen werden, welche dann wiederum eigene registrierte Services über ein Kommando informieren. Der Strukturbaum kann somit beliebig tief weiterverzweigen, wobei Knoten durch Koordinatoren und Blätter durch Services ausgefüllt werden.
Der Vorteil solch einer [Koordinator->Koordinator]*-Kommunikation liegt dabei auf der Hand: Viele teilnehmende Services in einem Teilnetz (z.B. Intranet) können durch eine einzelne Nachricht an ihren lokalen Koordinator (z.B. über das Internet) informiert werden, was Bandbreite zwischen den Teilnetzen spart und dadurch die Performance erhöht.
Weiterhin unterliegt die im Teilnetz verwendete konkrete Protokollumsetzung zur Ausführung eines Kommandos keinen vorgeschriebenen Festlegungen. Der "lokale" Koordinator kann als dazu verwendet werden zwischen verschiedenen Protokollumsetzungen zu "übersetzen".

3.7 Webservices publizieren und verwalten - *SOA-Register*

Bei der Entwicklung einer SOA ist es von hoher Wichtigkeit einen Ort zur Verfügung zu stellen, wo verschiedene Informationen (Schnittstelle, Beschreibung, Protokoll, Lokalisation etc.) über Services zentral und durchsuchbar verwaltet werden können. Dieser Ort erhält damit die Bedeutung eines der wichtigsten Dreh- und Angelpunkte in einer serviceorientierten Architektur.
Im Folgenden werden zwei standardisierte Vertreter solcher Register näher vorgestellt. Auf den Register-Standard UDDI (s.Kap. 3.7.1) soll dabei etwas genauer eingegangen werden, da mit dessen Hilfe der Prototyp für die authentifizierenden Webservices (s.Kap. 7) erstellt wird.
Im Anschluss daran wird das Konzept des ebXML-Registers (s.Kap. 3.7.2) vorgestellt und mit dem UDDI-Register verglichen.

3.7.1 UDDI (Universal Description, Discovery and Integration)

Eine Lösung für ein entsprechendes Register, welche durch das OASIS-Konsortium [OASIS], bestehend aus Einzelpersonen und namhaften Enterprise Software Firmen (IBM, Microsoft, Hewlett Packard, SAP und andere), standardisiert und spezifiziert wird, ist das UDDI-Register [UDDI].
In diesem Register können angebotene Services publiziert, Schnittstellen, Protokolle etc. beschrieben und bereichsspezifische Taxonomien für Services etabliert werden.
Ein unabhängiges Konsortium aus den Unternehmen IBM, Microsoft, NTT Communications und SAP verwalten frei zugängliche UDDI-Register, die ihre öffentlichen Daten untereinander abgleichen. Dadurch wird ein globales Register (UBR (UDDI Business Registry, auch gemeinhin als "öffentliches Register" bezeichnet) publiziert, welches es in erster Linie allen Unternehmen weltweit ermöglichen soll, ihre angebotenen Dienste zugänglich zu machen, sich gegenseitig zu finden und so B2B[16]-Aktivitäten in Gang zu setzen.
Neben der Eintragung von Daten in ein Register durch den Veröffentlicher selbst, besteht auch die Möglichkeit, dass ein UDDI-Register Metadaten mit Hilfe von WS-MetadataExchange (s.S. 29) direkt vom Service anfordert.
Ein UDDI-Register besteht aus mindestens einem UDDI-Knoten (1:n-Beziehung), welcher per Spezifikation mindestens eines der folgenden (Knoten-) APIs unterstützen muss [OA04]:

UDDI Inquiry: API für Suchanfragen

[16] Business to Business.

UDDI Publication: API für das Eintragen von Service-Metadaten in ein Register

UDDI Security: API für Sicherheit (z.B. Signaturen)

UDDI Custody Transfer: API für die Festlegung/Änderung der Besitzrechte an den verwalteten Daten

UDDI Subscription: API für das Anmelden von Nutzern, die über Änderungen im Register auf dem Laufenden gehalten werden wollen

UDDI Replication: API für das Replizieren von Registern

Jeder Knoten bietet dadurch die Möglichkeit zur Interaktion mit seinen verwalteten Daten. Neben den vorgestellten Knoten-APIs existieren in der UDDI-Spezifikation auch noch folgende optionale Client APIs:

UDDI Subscription Listener: API für den Callback an Klienten, die über bestimmte Änderungen an der Registry informiert werden möchten

UDDI Value Set: API für die externe Validierung von UDDI-Daten (Neueinträge, Änderungen)

Bisher war immer nur die Rede von "UDDI-Daten" allgemein. Die folgenden Unterabschnitte stellen die Basis-Datentypen der UDDI-Spezifikation im Detail vor (Überblick s.Abb. 6; [OA04]).

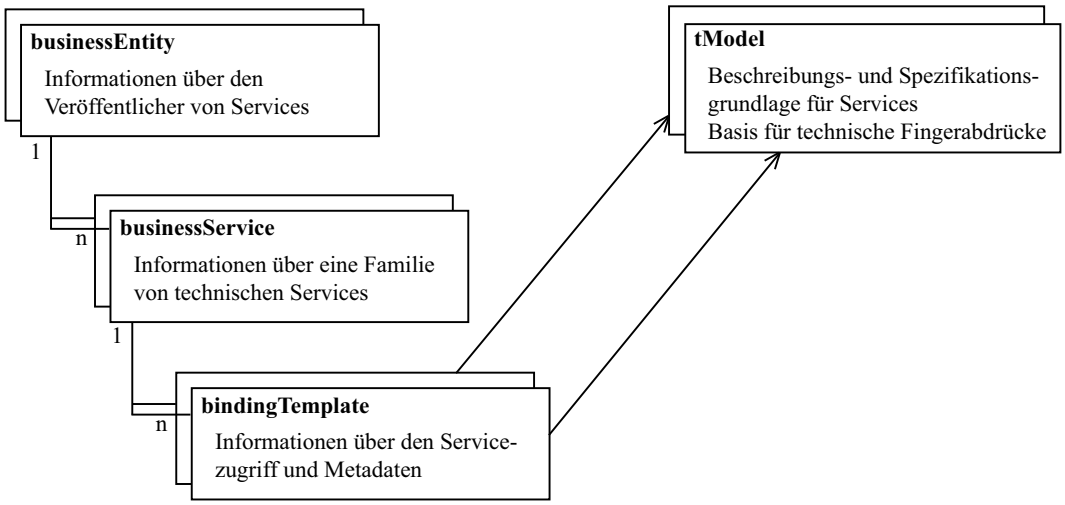


Abbildung 6: Datenstruktur eines UDDI-Registers

3.7.1.1 businessEntity - Anbieterbeschreibung

Informationen über Unternehmen, Personen etc., die in einem UDDI-Register Services veröffentlichen möchten, werden mit Hilfe der top-level Datenstruktur *businessEntity* festgehalten. Eine businessEntity enthält dafür neben der Beschreibung auch Informationen wie Name, Kontaktmöglichkeiten und Klassifikationsdaten[17] für den jeweiligen Provider. Für globale Geschäftsbeziehungen

[17]Standardisierte Daten, die genauer spezifizieren welche Dienstleistungen/Produkte ein Unternehmen anbietet (z.B. das United Nations Standard Products and Services Code System (UNSPSC)).

ist es auch möglich, relevante Daten in verschiedenen Sprachen bereitzustellen. Ein businessEntity-Element wird innerhalb eines UDDI-Registers durch das *businessKey*-Attribut (Schlüssel) eindeutig bestimmt und muss nicht gezwungenermaßen ein Geschäft im herkömmlichen Sinn darstellen. Vielmehr stellt dieses Element eine Abstraktion auf alle möglichen Anbieterformen für Services dar. Denkbar wären also auch Anbieter, wie Abteilungen, Anwendungen, Server usw..
Beschreibungen und technische Informationen über angebotene Services innerhalb eines businessEntity-Elementes werden mit Hilfe der enthaltenen businessService-Elemente (1:n-Beziehung; s.Kap. 3.7.1.2) näher beschrieben.

3.7.1.2 businessService - Services allgemein

Mithilfe des Elements *businessService* wird innerhalb des UDDI-Registers eine logisch zusammengehörige Gruppe von Webservices spezifiziert. Es werden auf diesem Level allerdings noch keinerlei technische Informationen über die enthaltenen konkreten Services gespeichert. Vielmehr wird eine Struktur zur Verfügung gestellt, welche es ermöglicht, eine Menge von Services in einer bestimmten Rubrik einzuordnen. Dazu werden wiederum Daten wie Name, Beschreibung und spezielle Klassifizierungsinformationen festgehalten, welche einen Überblick über den Zweck eines Webservices geben sollen.
Jeder businessService ist dabei genau einem businessEntity (s.Kap. 3.7.1.1) zugeordnet (n:1).

3.7.1.3 bindingTemplate - Services speziell

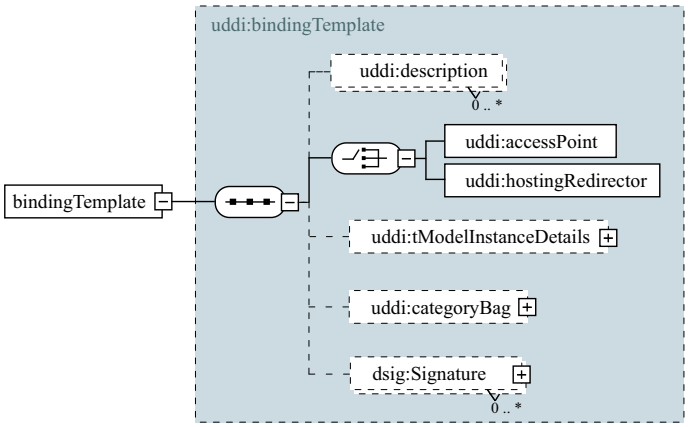


Abbildung 7: UDDI - *bindingTemplate*-Element

Ein *bindingTemplate* repräsentiert einen konkreten Webservice und enthält notwendige technische Informationen, um eine Verbindung zu einem Webservice aufbauen und mit diesem interagieren zu können. Dabei wird der Zugriff auf einen Webservice mit Hilfe der Angabe einer Netzwerkadresse (i.d.R. in Form einer URL) und eines konkreten Protokolls (HTTP, SMTP etc.) ermöglicht. Die Adresse kann dabei entweder direkt auf den Port des Webservice (*Access Point*) oder aber auf einen Mechanismus, der letztendlich zum Port führt (*Hosting Redirector*), verweisen.
Es können zusätzlich auch verschiedene Metadaten gespeichert werden, welche

ein leichtes Auffinden innerhalb eines businessService mit Hilfe von Kriterien und Parametern ermöglichen.
Die Sammlung von *tModelInstanceInfo*-Elementen innerhalb eines bindingTemplate wir auch als "technischer Fingerabdruck" des Webservices bezeichnet und erlaubt eine Überprüfung, ob ein konkreter Webservice mit einer bestimmten technischen Anforderung (ausgedrückt mit Hilfe von *tModel*-Elementen; s.Kap. 3.7.1.4) kompatibel ist.
Jedes bindingTemplate ist Kind von genau einem businessService (s.Kap. 3.7.1.2; n:1).

3.7.1.4 tModel - Daten, Konzepte, Konstrukte

tModel-Elemente existieren außerhalb der sonst üblichen Eltern-Kind-Beziehung innerhalb eines UDDI-Registers. Sie repräsentieren Daten, Konzepte und Konstrukte (z.B. verschiedene Spezifikationen, WSDL, Protokolle, Namensräume, Kontrakte, spezifische Verhaltensweisen etc.) auf einem abstrakten Level, losgelöst von den restlichen Strukturen und garantieren so Wiederverwendbarkeit und ein gewisses Maß an Standardisierung. Referenzen eines bindingTemplate auf ein tModel kennzeichnen Kompatibilität mit dem jeweiligen Konstrukt. Ein einziges tModel kann in diesem Zusammenhang beliebig oft von verschiedenen bindingTemplate-Elementen referenziert werden.
Es sei allerdings darauf hingewiesen, das es nicht üblich ist, technische Dokumente und ihre möglicherweise vorhandene Dokumentation in einem UDDI-Register direkt abzuspeichern. Vielmehr sollte ein tModel die Adresse, wo diese Dokumente gefunden werden können, sowie Metadaten und einen Entitätsschlüssel, der das jeweilige tModel eindeutig identifiziert, enthalten.
tModel-Elemente sind in der Lage, nahezu jedes beliebige Konzept innerhalb eines UDDI-Informationsmodels darzustellen. Sie können beispielsweise für folgende Zwecke eingesetzt werden:

- Informationen über das zu verwendenden Protokoll, z.B. HTTP oder SMTP
- Wert-Mengen, z.B. Identifikations- und Kategorisationssysteme, Namensräume etc.
- Strukturierte Kategorien unter Verwendung von mehreren Wert-Mengen (categorization groups)
- Adressformate
- Beliebige weitere nutzerdefinierte Konzepte

BindingTemplate-Elemente, die exakt die selbe Menge von tModel-Elementen referenzieren, werden als vom selben Typ bezeichnet und haben einen identischen "technischen Fingerabdruck". Auf diese Art und Weise können tModels dazu verwendet werden, Interoperabilität zwischen Softwaresystemen auszudrücken.

3.7.1.5 publisherAssertion - Beziehung zwischen Anbietern

Besitzen verschiedene businessEntity-Elemente Gemeinsamkeiten, welche in einem UDDI-Register zum Ausdruck gebracht werden sollen, so stellt eine *publisherAssertion* eine Möglichkeit dar, um genau diesen Sachverhalt festzuhalten.

Als Beispiel sei eine Organisation genannt, die aus mehreren Tochtergesellschaften besteht. Die Zusammengehörigkeit soll im Register zum Ausdruck kommen, jedoch besitzt ein einziges businessEntity-Element nicht die nötige Ausdrucksstärke, um eine passende Beschreibung für den Zweck aller Tochtergesellschaften zu formulieren. Die Lösung liegt hier in der Definition mehrerer businessEntity-Elemente, welche anschließend über eine publisherAssertion als zusammengehörig markiert werden.
Um zu vermeiden, dass ein businessEntity eine Zugehörigkeit zu einem beliebigen anderen businessEntity vorgeben kann, wird diese Beziehung erst sichtbar, wenn beide Seiten eine identische publisherAssertion publizieren.

3.7.1.6 operationalInfo - Metadaten über eine Veröffentlichung

Immer, wenn eine Instanz der gerade vorgestellten Datenstrukturen (mit Ausnahme der publisherAssertion) veröffentlicht bzw. modifiziert wird, wird ein zur jeweilige Operation zugehöriger Log-Eintrag angelegt: das *operationalInfo*-Element. Dieses beinhaltet das Datum und die Zeit der Operation, den Identifikator des UDDI-Knotens, auf dem die Operation ausgeführt wurde, sowie den Identifikator des jeweiligen Veröffentlichers.

3.7.1.7 Vergleich UDDIv2 und UDDIv3

Obwohl sich die Details der vorangegangene Beschreibung eines UDDI-Registers inhaltlich auf die Version 3 der Spezifikation beziehen, ist ein Vergleich zur Version 2 wichtig, da diese zur Zeit eine weite Verbreitung genießt und die Änderungen zur Version 3 deutlich mehr als nur von fehlerbehebender Natur sind.
Neben einer Verbesserung des Spezifikationsdokumentes (Eliminierung von Redundanz und Konflikten; Erweiterung um Anhänge, welche die Nutzung der wichtigsten Bestandteile erklären und beschreiben [Be02]), wurden unter anderen folgende wichtige Neuerungen und Erweiterungen eingeführt:

Multi-Registry Umgebungen :
In UDDIv2 war es schwierig, die Übergabe von Einträgen aus einem z.B. privaten Register in ein öffentliches Register durchzuführen. Die Identifikatoren wurden dabei umbenannt und erschwerten ein späteres Auffinden von bekannten Einträgen.
UDDIv3 unterstützt das Konzept der Sichtbarkeit (private, semi-private und öffentliche Register), indem es so genannte Root- und Affiliate-Register einführt. Das ROOT-Register ist dabei für die Schlüsselvergabe verantwortlich und prüft diese auf Eindeutigkeit innerhalb seines Register-Verbandes. Ein Affiliate-Register lässt seine Schlüssel also erst von seinem Root-Register prüfen, bevor es den Eintrag zulässt.

Vorgabe von Schlüsseln :
Ab UDDIv3 ist es möglich die Schlüssel für neue Einträge selbst vorzugeben. Die einzige Anforderung besteht darin, dass sie mit *"uddi:"* beginnen müssen.
In älteren Versionen wurden diese vom Register als eindeutige zufällige hexadezimale Zeichenkette vorgegeben, was den Umgang mit ihnen als eindeutigen Identifikator sehr schwierig gestaltete.

Subscription :
Mit Hilfe von "Subscription" wird es möglich, über Änderungen innerhalb eines UDDI-Registers informiert zu werden. Jeder, der das Recht besitzt, sich als Empfänger von Änderungsmitteilungen registrieren zu lassen, kann zusätzlich auch vorgeben, in welchem Format er die entsprechenden Mitteilungen erhalten möchte.
Eine wichtige Rolle kommt diesem Konzept innerhalb von Multi-Registry Umgebungen zu: ein Affiliate-Register[18] kann sich so immer über Änderungen im Root-Register auf dem Laufenden halten.

Policy :
Policies beschreiben das spezifische Verhalten eines Registers oder eines Knotens innerhalb eines Registers. In früheren Versionen vermischte die UDDI-Spezifikation Policy-Fragen mit UBR-Implementationsdetails [Be02]. UDDIv3 trennt diese und erweitert die Rolle von Policies, indem sie bestimmte Policies vorgibt, die von einem Register eingehalten werden müssen.

Sicherheit :
Das UDDI-Sicherheitsmodell wird durch eine Reihe von Register- und Knoten-Policies vorgegeben.
Ab UDDIv3 wird eine neue Vertrauensebene eingeführt, indem sämtliche UDDI-Basisdatentypen vom Veröffentlicher signiert werden können. Ein Anfragender kann somit die Authentizität erhaltener Suchanfrageergebnisse selbst überprüfen.
Die Anfrage-Sprache für ein UDDIv3-Register geht sogar soweit, dass sie bei Bedarf die Ergebnismenge so einschränkt, dass nur Einträge geliefert werden, die auch digital signiert wurden.
Auch dieses Konzept gewinnt innerhalb von Multi-Registry Umgebungen insofern an Bedeutung, dass sichergestellt werden kann, dass zwischen Root-Register und Affiliate-Register ausgetauschte Daten nicht durch Dritte unbemerkt modifiziert werden können.

Erweiterte Anfragemöglichkeiten :
In UDDIv3 wurden neue Abfragekriterien für die Suche in einem UDDI-Register (z.B. ignorieren von Groß- und Kleinschreibung, SQL-ähnliche Verwendung von Jokerzeichen, Signatur vorhanden etc.), leichte Erweiterbarkeit von Abfragekriterien durch das UDDI-Register, zusammengesetzte Anfragen (Schnittmenge, Vereinigungsmenge etc.) und eine bessere Verwaltung von großen Ergebnismengen eingeführt.

3.7.2 ebXML Register

Die Spezifikation für das ebXML-Register (electronic business eXtensible Markup Language) wird ebenfalls vom OASIS-Konsortium in Zusammenarbeit mit UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) und ca. 30 weiteren Industriepartnern entwickelt. Dabei stellt sich die Frage: Warum werden zwei verschiedene Spezifikationen mit der scheinbar gleichen Aufgabe vom selben Konsortium unterstützt und ausgearbeitet?
Bei einer eingehenden Betrachtung der Spezifikationen fällt tatsächlich eine Überschneidung

[18] Ein an einen Registerverband angegliedertes Register.

im Aufgabenbereich "Auffinden von Services" auf, jedoch ist der Primärfokus des ebXML-Registers mehr auf die Zusammenarbeit von Firmen gerichtet. Diesen Sachverhalt kann man auf zwei Ebenen betrachten: Zusammenarbeit zur Entwicklungszeit und Zusammenarbeit zur Laufzeit.
Ein ebXML-Register ermöglicht das Abspeichern und Verwalten von XML-Artefakten, welche dann von den zusammenarbeitenden Unternehmen weiterentwickelt werden können. Dies kann sowohl zur Entwicklungszeit innerhalb z.B. einer Organisation oder aber zur Laufzeit zwischen mehreren Handelspartnern geschehen. Z.B. könnte so ein XML-Artefakt von einem Nutzer veröffentlicht und von vielen anderen dann verwendet und erweitert werden. Dadurch könnten z.B. Geschäftsabläufe durch viele verschiedenen Parteien entwickelt werden, welche dann in eigenen Handelsszenarios umgesetzt werden.
Der eigentliche Einsatzbereich eines ebXML-Registers liegt nach Meinung vieler Autoren [Ch03, Ci02] "hinter" einem UDDI-Register. Aufgrund der stärkeren Verbreitung des UDDI-Registers [Ch03] für die Suche wurde sogar vom ebXML-Projekt-Team selbst ein Dokument veröffentlicht [HM+01], welches beschreibt, wie UDDI zum Auffinden eines ebXML-Registers verwendet werden kann. Hat man dann das gewünschte ebXML-Register gefunden, können mit dessen Hilfe die Handelsbeziehungen vertieft werden.
Die ebXML Registerspezifikation besteht aus zwei Teilen: dem ebRIM (ebXML Register Informations-Modell) und dem ebRS (ebXML Register Services).
Neben der Speicherung von XML-Artefakten können auch andere Artefakte verwaltet werden, welche die Zusammenarbeit von Unternehmen unterstützen. Da wären z.B.:

- *Collaboration Protocol Profile* (CPP): Beschreibt die Fähigkeiten für den Nachrichtenaustausch eines Unternehmens innerhalb einer geschäftlichen Beziehung.
- *Collaboration Protocol Agreement* (CPA): Beschreibt die Bedingungen, mit denen Unternehmen übereinstimmen müssen, bevor sie eine geschäftliche Beziehung aufbauen können.
- *Business Process Specification Schema* (BPSS): Standardisiertes Rahmenwerk durch welches Geschäftssysteme konfiguriert werden können, um Geschäftsbeziehungen, welche Geschäftstransaktionen enthalten, eingehen zu können.

Das Datenmodell für die Suche nach Services besitzt große Ähnlichkeiten zum Datenmodell eines UDDI-Registers. Es wird eine andere Terminologie verwendet, welche sich wie folgt auf die UDDI-Basisdatentypen abbilden lässt[19]:

UDDI Register	**ebXML Register**
businessEntity	Organization
businessService	Service
bindingTemplate	ServiceBinding
tModel	ExtrinsicObject

Ein interessantes Merkmal eines ebXML-Registers ist die direkt vorgesehene Möglichkeit einen hierarchischen Kategorisationsbaum aufzubauen. Kontext-Informationen, wie ein Unternehmen befindet sich in Dresden, Dresden liegt in Sachsen, Sachsen liegt in Deutschland usw. sind somit leicht und wiederverwendbar umsetzbar. Mit Hilfe der tModel-Elemente wäre dies in einem UDDI-Register auf einem Umweg auch möglich, ist aber nicht direkt

[19] Von einer exakten Äquivalenz kann dabei natürlich nicht ausgegangen werden, sondern nur von einer Ähnlichkeit.

vorgesehen.
Weiterhin verfügen alle Datentypen in einem ebXML-Register über einen erweiterten Lebenszyklus. Während in einem UDDI-Register Daten nur hinzugefügt, modifiziert und gelöscht werden können, bewegen sich ebXML-Daten zusätzlich durch folgende Zustände:

- *Submitted*: Das Datum wurde soeben zum Register hinzugefügt.
- *Approved*: Das hinzugefügte Datum ist bestätigt wurden und ist nun sichtbar.
- *Deprecated*: Das Datum ist veraltet und kann nur noch gelesen, aber nicht mehr verändert werden.
- *Undeprecated*: Ein vorher als Deprecated gekennzeichnetes Datum wird wieder für Änderungen frei gegeben.
- *Withdrawn*: Ein Datum wurde entfernt. Sein zugehöriges ExtrinsicObject existiert allerdings weiterhin.

3.8 Weitere Technologien und Begriffe

Dieses Kapitel soll einen kurzen Überblick über weitere Webservices betreffende Technologien geben, ohne einen Anspruch auf Vollständigkeit zu erheben.

In der Literatur taucht hin und wieder der Begriff GXA (Global XML Architecture) auf. Dieser stellt eine Bezeichnung dar, unter welcher Microsoft eine Menge verwandter Webservice-Standards zusammenfasst. Von anderen Organisationen wird der selbe Sachverhalt auch als "Web Services II" bzw. "WS2" oder "WS-I Advanced Profile" bezeichnet. Folgende Spezifikation werden unter diesen Namen zu einem Profil[20] zusammengefasst:

> WS-Referral, WS-Routing, WS-Coordination, WS-Transaction, WS-Attachments, WS-Inspection und DIME,

sowie folgende Spezifikationen, die sich sicherheitsrelevanten Themen widmen (s.Kap. 6):

> WS-Security, WS-Policy, WS-PolicyAttachment, WS-PolicyAssertions, WS-SecurityPolicy, WS-Trust, WS-SecureConversation und WS-Privacy.

WS-Addressing: bietet eine Host-Protokoll-unabhängige Möglichkeit den Sender und Empfänger einer SOAP-Nachricht anzugeben.
Sehr oft werden Sender und Empfänger im Rahmen der Möglichkeiten des Host-Protokolls angegeben (HTTP: URL und Return-Adresse). Dies birgt allerdings Gefahren: Adressinformationen können durch eine Unterbrechung der Verbindung (z.B. Timeout) verloren gehen oder durch einen Intermediary (z.B. Firewall) verändert werden.
Mit Hilfe von WS-Addressing wird der Sender und der Empfänger einer Nachricht in der SOAP-Nachricht selbst angegeben und ermöglicht so eine Entkoppelung von Host-Protokoll und SOAP.

WS-Routing: definiert, wie Routing-Informationen im Kopf einer SOAP-Nachricht angegeben werden müssen, um diese auf einem vorgegebenen Pfad von Intermediaries zu einem bestimmten Webservice zu schicken. Diese Routing-Informationen müssen

[20] Sammlung von Spezifikationen, welche dafür konzipiert wurden zusammen zu arbeiten

nicht immer vom Sender vorgegeben werden, sondern können auch z.B. zur Lastverteilung bei der Weiterverarbeitung einer Anfrage durch einen Intermediary angepasst bzw. eingefügt werden.

WS-MetadataExchange ermöglicht es, Webservice-Metadaten (WSDL, WS-Policy, XSD) durch eine definierte Schnittstelle zur Verfügung zu stellen. Für einen Anfragenden ist es nur notwendig, die Referenz (Port) zum gewünschten Webservice zu kennen. Die erhaltenen Daten ermöglichen die automatische Bindung an den jeweiligen Dienst.

WS-ReliableMessaging definiert einen Mechanismus, um die zuverlässige Auslieferung von Nachrichten über ein unzuverlässiges Netz (Nachrichten verschwinden, Verbindungsabbrüche etc.) zu garantieren. Dazu erhalten alle Nachrichten eindeutige Identifikatoren (z.B. eine Sequenznummer), welche es ermöglichen bei Nichterhalt gezielt neu zu senden bzw. doppelte Verarbeitung von identischen Nachrichten zu verhindern.

WS-Transaction: spezifiziert die Grundlagen, um zwei verschiedene Typen von Transaktionen in den Kontext der Webservices zu überführen: ACID-Transaction (entspr. WS-AtomicTransaction) und Long Running Transaction (entspr. WS-BusinessActivity). WS-Transaction baut dabei in großem Maße auf WS-Coordination auf, um es mehreren teilnehmenden Parteien zu ermöglichen, an einer koordinierten Transaktion teilzunehmen.

WS-AtomicTransaction: definiert ein auf WS-Coordination und WS-Transaction aufsetzendes Protokoll, um das traditionelle 2-Phasen-Transaktionsprotokoll zu implementieren.

WS-BusinessActivity: definiert ein auf WS-Coordination und WS-Transaction aufsetzendes Protokoll, um lang laufende transaktionsbasierte Geschäftsoperationen zu implementieren.

WS-Attachments: adressiert die drei Hauptprobleme, welche beim Senden von Attachments innerhalb einer SOAP-Nachricht auftreten:

1. das Einbetten von binären Attachment-Daten in ein XML-Dokument kann die XML-Struktur zerstören
2. das Einbetten von XML-Dokument-Attachments zerstört die XML-Struktur und kann das Parsen unmöglich machen
3. der Wunsch ein Attachment in einzelne Datensätze (Records) zu zerlegen und entsprechend flexiblen (z.B. random access) Zugriff darauf zu ermöglichen

WS-Referral ist ein zustandsloses Protokoll für das Einfügen, Löschen und Abfragen von Einträgen eines SOAP-Routers. Ein SOAP-Router ist ein SOAP-Knoten, welcher SOAP-Nachrichten nach einem vorgegebenen Schema weiterleitet und entweder als einzelner Webservice oder in Kombination mit anderen Services arbeitet.

WS-Inspect: erlaubt das Auffinden von angebotenen Webservices durch Dritte. Obwohl es offensichtlich Überlappungen mit der UDDI-Spezifikation geben muss, sind beide grundsätzlich für verschiedene Zwecke gedacht:

- WSIL (Web Service Inspection Language) ist für die Veröffentlichung durch den Entwickler selbst gedacht, wohingegen UDDI meist durch Drittanbieter implementiert und gestellt wird.

- WSIL-Implementationen werden oft geografisch dicht zum jeweiligen Webservice angeboten (z.B. auf dem gleichen Server)
- WSIL ist als leichtgewichtige Implementation gedacht, wohingegen UDDI ein Verzeichnis mit einer Vielzahl von Features und Möglichkeiten darstellt.
- WSIL ist für einfache Anfragen gedacht, wohingegen UDDI wesentlich komplexerer Anfragen zulässt

DIME: ist eine Spezifikation um mehrere Datensätze in einem Dokument zusammenzupacken. Es verwendet Datenköpfe mit einer festen Länge und ermöglicht randomaccess, in dem es die Datensatzlänge im Kopf fixiert. Dies stellt im Vergleich zu MIME-Multipart einen entscheideneden Vorteil dar, da dieses auf Grund seiner Separator-Struktur für den Zugriff vorher geparst werden muss.
Nachrichten im DIME-Format können per WS-Attachment an eine SOAP-Nachricht gebunden werden.

BPEL4WS: (Business Process Execution Language for Web Services) ist eine XML-basierte Sprachdefinition für komplexe Arbeitsabläufe. Es erlaubt die Erstellung von fortgeschrittenen Geschäftsprozessen (abstrakte Metaebene welche beschreibt, wie die verschiedensten Webservices interagieren müssen, um ein bestimmtes Ziel zu erreichen), welche Webservices sowohl konsumieren als auch anbieten können.

4 Grundlegende Sicherheitsbetrachtungen

Will man sich mit Sicherheitsbetrachtungen im Kontext von Computersystemen auseinandersetzen, so stellt sich zuallererst die Frage, was "Sicherheit eines Computersystems" eigentlich bedeutet. Dies ist nicht einfach zu beantworten und wenn man sich damit beschäftigt, denkt man vermutlich zuerst an Schlagworte, wie "Verschlüsselung" oder "Signaturen". Dies ist allerdings schon sehr speziell und trifft den Punkt nur teilweise. Simson Garfinkel und Gene Spafford beschreiben diesen Sachverhalt treffender in ihrem Buch *Practical UNIX & Internet Security* [GS03]:

> "*a computer is secure if you can depend on it and its software to behave as you expect*"
>
> "Ein Computer ist sicher, wenn man sich darauf verlassen kann, dass er und seine Software sich so verhalten, wie man es erwartet."

Diese Aussage weißt auf einen interessanten Punkt hin, der in der Sicherheits-Industrie für Software leider oftmals vergessen wird: Sicherheit muss im Kontext des verwendenden Systems verstanden und implementiert werden, nicht nur um seiner Selbst Willen [On02].

Es ist eine Tatsache, dass jedes Computersystem, welches in irgend einer Art mit anderen (potentiell unsicheren bzw. nicht kontrollierbaren) Systemen verbunden ist(Intranet, Internet etc.), einer permanenten Bedrohung durch Hacker, Cracker und so genannte "Script Kiddies" ausgesetzt ist.
Hacker und Cracker sind Spezialisten auf ihrem Gebiet und unterscheiden sich nur in ihren Absichten. Während ein Hacker versucht Lücken in Computersystemen zu finden, um deren Administratoren oder die Systemhersteller zu warnen (stehen daher oft auch auf der Gehaltsliste von Unternehmen), setzt ein Cracker sein Knowhow ein, um unter potentieller Missbrauchsabsicht, in meist veraltete bzw. schlecht geschützte Systeme von Unternehmen oder Regierungsstellen einzudringen.
Ein nicht zu vernachlässigendes und schnell wachsendes Angriffspotential geht von unter dem Begriff "Script Kiddies" zusammengefassten Personen aus [Ns04], welche, meist mit wenigen bis gar keinen Kenntnissen von Computersicherheit, versuchen, bekannte Sicherheitslöcher unter Verwendung fertiger Programme bzw. Skripte auszunutzen. Die Zahl dieser Personen ist aufgrund der geringen notwendigen Voraussetzungen sehr hoch.
Passenden Programme und Skripte im Internet zu finden ist leicht und führt mit den richtigen Suchmaschinen oder/und nach Konsultationen in einschlägigen Foren (z.B. IRC - Internet Relay Chat) sehr schnell zum Erfolg.

Möglichen Angriffspunkte ziehen sich durch alle Ebenen der OSI-Architektur.
Dieses Kapitel soll einen Überblick darüber geben, welche Aspekte eines Systemes zu schützen sind und mit welchen Bedrohungen in einem vernetzten System zu rechnen ist. Im letzten Teilkapitel wird aufgezeigt, wie sich die Bedrohungen in den letzten Jahren entwickelt haben und wo sie sich vermutlich hinbewegen werden.

4.1 Schutzziele

Für eine eingehende Betrachtung von Sicherheitsaspekten für Webservices ist es wichtig mit einer Aufteilung in unterschiedliche Schutzziele[21] zu beginnen.

[21] Die Anzahl der nötigen Schutzziele variiert in der Literatur, da sich einige aus abstrakter Sicht auch zu einem einzigen zusammenführen lassen (z.B. 3 in [Pf00] und 6 in [Ec02, On02]).

In den Kapiteln 4.1.1 bis 4.1.7 werden Vertraulichkeit, Integrität, Authentizität, Autorisation, Verbindlichkeit, Verfügbarkeit und Privatheit als mögliche Schutzziele vorgestellt und näher erläutert. Diese strikte Trennung soll es erleichtern in weiterem Verlauf der Arbeit entsprechende Anforderungen zu formulieren.
Der Frage, ob theoretische Ansätze, Spezifikationen und eventuell Implementationen für entsprechende Sicherheitsmechanismen im Webservice-Umfeld existieren und wie sie zu bewerten sind, wird in Kapitel 6 nachgegangen.
Auf Bedienungs-, Soft- oder Hardwarefehler bzw. transitive Trojanische Pferde[22] innerhalb der verwendeten Software wird in der nachfolgenden Unterteilung nicht explizit eingegangen, da diese potentiell allgegenwärtig sein können und bei allen Schutzzielen zu Einschränkungen in der Nutzung oder zu einer (bösartigen) Unterwanderung führen können.

4.1.1 Vertraulichkeit - *confidentiality*

Vertraulichkeit befasst sich mit der Geheimhaltung von gespeicherten oder im Transit befindlichen Daten. Dies bedeutet, dass Daten sowohl gegen Angreifer, welche in ein System einbrechen (oder sich dort legal aufhalten dürfen) und versuchen dieses direkt auszuspähen, als auch gegen das "Abhören" einer Kommunikationsbeziehung über ein verteiltes Netzwerk durch Außenstehende, Provider etc. geschützt werden müssen.
Bei lokal gespeicherten Daten werden Algorithmen der symmetrischen oder asymmetrischen Kryptografie verwendet, wobei der Schlüssel an einer geschützten Stelle (z.B. Diskette außerhalb des Laufwerks) oder/und in einem geschützten Zustand (z.B. Passwortverschlüsselt) verwahrt werden muss.

Für den Schutz der Vertraulichkeit bei Kommunikationsbeziehungen bieten sich bei einer Betrachtung des OSI-Protokollstack aus Anwendungssicht zwei Arten der Verschlüsselung an: anwendungsorientierte oder transportorientierte Verschlüsselung.
Eine anwendungsorientierte Verschlüsselung muss in Schicht 7 umgesetzt werden, wohingegen transportorientierte Verschlüsselung in einer Schicht unterhalb der Anwendungsebene stattfindet (SSL in Schicht 5, VPN (Virtual Private Network) per IPSEC in Schicht 3 etc.).
Transportorientierte Sicherheit bietet den Vorteil, dass sich eine Anwendung nicht um die Verschlüsselung ihrer Transferdaten zu sorgen braucht, sondern dies auf den unteren Schichten des OSI-Protokollstacks automatisch umgesetzt wird. Problematisch wird dies allerdings bei der Einbeziehung so genannter Intermediaries. Dies sind Kommunikationsknoten auf dem Weg der übermittelten Nachricht, welche sich nicht auf das bloße Routen von Paketen beschränken, sondern auf die (Teil-)Nachricht in der Anwendungsebene zugreifen und dort für das Protokoll oder die Nachrichtenaufgabe notwendige Operationen ausführen. Transportorientierte Sicherheit versagt an dieser Stelle, da die übermittelten Daten hier temporär unverschlüsselt und ohne jeglichen Schutz vorliegen (Punkt-zu-Punkt Verschlüsselung).
Anwendungsorientierte Verschlüsselung beseitigt diesen Umstand, indem die Anwendung entscheidet in welcher Art und Weise welche (Teil-)Daten einer Nachricht verschlüsselt werden. Dadurch kann gewährleistet werden, dass Intermediaries nur auf die Daten zugreifen können, welche für sie bestimmt sind und diese somit auch über beliebig viele Zwischenstationen ihre gewünschte Vertraulichkeit erhalten (Ende-zu-Ende Verschlüsselung).

[22] Ein Systemteil ist ein Trojanisches Pferd, wenn er unter Ausnutzung der ihm anvertrauten Daten und Rechte *mehr* als das von ihm Erwartete oder von ihm Erwartetes *falsch* oder *nicht* tut.
Ein *transitives* Trojanisches Pferd ist ein Trojanisches Pferd, welches nicht durch einen unmittelbaren Angriff in das betroffene System eingeschleust wurde, sondern sich in der transitiven Hülle eines oder mehrerer Soft- bzw. Hardware-Entwurfshilfsmittel rekursiv bis in das betroffene System ausbreiten konnte[Pf00].

4.1.2 Integrität - *integrity*

Mit Hilfe dieses Schutzzieles soll jede unbefugte Manipulation in Transit befindlicher oder abgespeicherter Daten erkannt werden. Die Veränderung der Daten an sich kann auch bei einer strikten Einhaltung dieses Schutzziel nicht verhindert werden. Jedoch das "Wissen" um eine Veränderung ermöglicht es, angemessen zu reagieren.

Der Schutz der Integrität kann ebenso wie der der Vertraulichkeit aus Anwendersicht anwendungsorientiert oder transportorientiert erfolgen.
Bei einer transportorientierten Umsetzung (z.B. SSL) können die Daten durch eventuell vorhandene Intermediaries unbemerkt manipuliert werden, da sie dort temporär ohne Schutz vorliegen (Punkt-zu-Punkt-Integrität).
Mit Hilfe einer anwendungsorientierten Integritätssicherung kann eine Ende-zu-Ende-Integrität auch über zwischengeschaltete Intermediaries erfolgen. Die Anwendung kann so genau festlegen, welche Integritätskriterien an welchem Knoten eingehalten werden sollen.

Dieses Schutzziel kann, wie bereits am Beispiel "SSL" erkenntlich, neben der Verwendung von speziellen Signaturalgorithmen auch unter Verwendung von kryptografischen Konzelationsalgorithmen umgesetzt werden.
Manipulationen an den übermittelten Daten werden dadurch erkannt, dass die Entschlüsselung "fehlschlägt". Das Erkennen eines "Fehlschlages" kann entweder am Daten-Kontext (z.B. XML-Format, spezieller Header etc.) oder besser an zusätzlich eingefügter Redundanz (z.B. Hash, Quersumme etc.) in den Ausgangstext erfolgen. Die eingefügt Redundanz sollte dabei allerdings auf anderen mathematischen Grundlagen beruhen, wie der verwendete Verschlüsselungsalgorithmus.
Wird ein bei der Übertragung manipulierter verschlüsselter Text entschlüsselt, so werden diese Kriterien durch das Entschlüsselungsergebnis mit sehr hoher Wahrscheinlichkeit nicht mehr erfüllt.

Ein Spezialfall des Schutzzieles Integrität ist die Möglichkeit eines Empfängers nachzuweisen, dass eine Nachricht auch tatsächlich vom Sender verschickt worden ist (s.Kap. 4.1.3).

4.1.3 Verbindlichkeit - *nonrepudiation*

Dieses Schutzziel garantiert neben der Integrität von Daten auch den Nachweis, wer diese Daten abgespeichert oder verschickt hat. Der Ersteller bzw. Sender der Daten darf das Abspeichern bzw. Verschicken der Daten nicht abstreiten können.

Zur Umsetzung werden asymmetrisch erstellte Signaturen (nur einer kann signieren, viele können prüfen), auch unter dem Namen "Digitale Signaturen" bekannt, verwendet. Das Prinzip der digitalen Signaturen stellt sicher, dass nur eine Entität dazu in der Lage ist, mit Hilfe ihres privaten Schlüssels korrekte Signaturen zu erstellen und potentiell viele Entitäten dazu in der Lage sind, eine vorhandene Signatur mit Hilfe des öffentlichen Schlüssels (Testschlüssel) zu prüfen.
Eine positive Überprüfung von Daten und ihrer digitalen Signatur stellt sicher, dass die Daten nachweislich vom Eigentümer des zugehörigen privaten Schlüssels erstellt und entweder abgespeichert bzw. verschickt worden sind.

4.1.4 Authentizität - *authentication*

Das Schutzziel der Authentizität stellt einen grundlegender Mechanismus zur Überprüfung der von einem Subjekt vorgegebenen Identität dar.

Digitale Signaturen stellen die wichtigsten Werkzeuge zur Authentifikation in verteilten Systemen dar. Wird ein bestimmtes Datum korrekt digital signiert, so kann man mit der Sicherheit des verwendeten kryptografischen Systems davon ausgehen, dass man es mit der zum Schlüssel bzw. Zertifikat (s.Kap. 2.7) gehörenden Entität zu tun hat. Die Zuordnung zu einer realen Person oder zu einem realen System kann mittels PKI erfolgen (s.Kap. 2.8).

Weitere Methoden zur Authentizitätsfeststellung, wie die Erfassung biometrischer Daten werden mit unterschiedlichem Erfolg eingesetzt und befinden sich in der Entwicklung. Die Sicherheit dieser Verfahren beruht bei verteilten Systemen letzten Endes ebenfalls auf kryptografischen Verfahren.

Die Authentifikation als Vorgang kann auf zwei Ebenen stattfinden:

1. **Nachrichtenebene** ("message authentication"):
   Der Empfänger einer Nachricht kann prüfen, ob sie von einem bestimmten Sender generiert und nicht modifiziert worden ist.

2. **Entitätsebene** ("entity authentication"):
   Geht weiter als eine Prüfung auf Nachrichtenebene, indem sie es den Kommunikationspartnern zusätzlich erlaubt die Identität des Gegenüber festzustellen.

Wichtig ist dabei, nicht aus dem Auge zu verlieren, dass eine erfolgreich durchgeführte Authentifikation immer mit dem Verlust eines bestimmten Grades an Privatsphäre einhergeht und damit direkt im Gegensatz zum im Kapitel 4.1.7 vorgestellten Schutzziel "Privatheit" steht.

4.1.5 Autorisation - *authorization*

Autorisation ist ein, meist auf Authentifikation basierender Mechanismus, welcher einer bestimmten Entität seine auf einem bestimmten System zustehenden Rechte zuteilt.
Während Authentifikation sich mit der Frage "Wer bist du?" auseinandersetzt, ist es Aufgabe der Autorisation die Frage "Was darfst du erlaubter Weise tun?" zu beantworten.
Einer bestimmten, möglicherweise authentifizierten Entität, werden, entsprechend festgelegter Regeln, bestimmte Zugriffsrechte erteilt.
Die eigentliche Zugriffskontolle durch das verwaltende System kann dabei auf unterschiedlichen konkreten Mechanismen beruhen: z.B. Unix-Dateifreigabe (rwx), Acces Control Lists (ACL), Role-Based Access Contotl (RBAC), proprietäre Insellösungen etc..
ACL und RBAC werden im Rahmen von XKMS in Kapitel 6.6 näher beschrieben.

4.1.6 Verfügbarkeit - *availability*

Das Schutzziel "Verfügbarkeit" beschäftigt sich damit, dass es in einem Netz allen Teilnehmern ermöglicht werden muss, gewünschte (technisch mögliche und erlaubte) Kommunikationsbeziehungen aufzubauen.
Eine gezielte Be- bzw. Verhinderung dieses Schutzziels führt zu einer Einschränkung, welcher durch die strikte Wahrung der anderen Schutzziele nicht zu begegnen ist. Ohne Kommunikation keine Vertraulichkeit, Integrität etc..
Angriffe auf die Verfügbarkeit (z.B. DoS-Attacken; s.Kap. 5.2.1) sind meist mit finanziellen oder möglicherweise sogar personellen Schäden (z.B. Software in einem Krankenhaus) verbunden.

Einen absoluten[23] Schutz der Verfügbarkeit kann es im Gegensatz zu Verschlüsselung und Integrität nicht geben, da erstens Verbindungen physikalisch beschädigt oder gar durchtrennt werden können und die Beantwortung/Ablehnung "sinnloser" bzw. bösartiger Anfragen ebenfalls Ressourcen verbraucht ("Nein!"-sagen kostet Ressourcen).

4.1.7 Privatheit - *privacy*

Dieses Schutzziel, welches in der Literatur auch als "Informationelle Selbstbestimmung" bezeichnet wird, beschäftigt sich mit den Bedürfnissen von in einem Rechnernetz kommunizierenden Teilnehmern nach Privatsphäre bzw. Anonymität.
Zur Zeit wird weltweit an Gesetzen gearbeitet, welche den Schutz von Daten betreffen (nicht zu verwechseln mit Vertraulichkeit). Zu schützenden Daten in diesem Kontext sind z.B. der momentane Standort eines Nutzers/Gerätes oder personengebundenen Daten, wie Alter, Geschlecht etc..
Einige dieser Daten sind für bestimmte im Internet angebotene Dienstleistungen notwendig, jedoch muss geklärt sein, wie damit durch die jeweiligen Dienstanbieter umgegangen werden muss.
Im heutigen Kommunikationszeitalter, wo z.B. fast jeder Zugriff auf einen Dienst im Internet zurückverfolgt werden kann und wo brisante Daten, wie z.B. die Kreditkartennummer in Formulare eingegeben werden, gewinnt dieser Aspekt immer mehr an Bedeutung. Denn was passiert, wenn Daten zwar sicher übertragen wurden, aber der Diensteanbieter die Daten zwischenspeichert und ein Angreifer in dessen System einbricht?
Für diese und weitere Fragen müssen zuerst bindende gesetzliche Grundlagen geschaffen werden, worauf dann bestimmte Anforderungen an Dienstleister gestellt und von diesen zwingende umgesetzt werden müssen.
Es existieren bereits Frameworks, die sich mit dem Abgleich von Nutzerwünschen in Bezug auf Privatheit und den Zusagen von Diensteanbietern beschäftigen (z.B. P3P, s.S. 62).

4.2 Bedrohungen

Die möglichen Bedrohungen für ein beliebiges Computersystem (und somit im Speziellen auch für Webservices) lassen sich in *beabsichtigte* und *unbeabsichtigte* Bedrohungen unterteilen.

Die *unbeabsichtigten Bedrohungen* sollen im Rahmen dieser Arbeit nur am Rande erwähnt werden, da ihre Bekämpfung nicht direkt in das Aufgabengebiet der Informatik fällt.
In diese Kategorie fallen Fehlverhalten, die zu einer Verletzung aller in Kapitel 4.1 aufgeführten Schutzziele führen können. Dazu gehören z.B. [Ho03]:

- höhere Gewalt (z.B. Naturkatastrophen)
- menschliches Fehlverhalten (z.B Fehlbedienung)
- technisches Versagen (z.B. Fehlkonstruktion, Abnutzung)
- Umwelteinflüsse (z.B. Temperatur, Nässe, mechanische Einwirkung)

Wie man aus dieser Aufzählung entnehmen kann, sind diese Einwirkungen zufällig und die Wahrscheinlichkeit ihres Auftretens kann entweder gar nicht oder nur mit hohem Aufwand

[23] "Absolut" bei Verschlüsselung und Integrität im Sinne von informationstheoretischer Sicherheit, denn zufälliges richtiges Raten kann nicht ausgeschlossen werden.

verringert werden.

Anders sieht es da bei den *beabsichtigten Bedrohungen* aus. Diese Art der Bedrohung entsteht durch ein Einwirken eines entsprechend motivierten Angreifers (i.A. reale Personen) und kann theoretisch durch entsprechende Vorsichtsmaßnahmen vermieden oder zumindest auf ein Minimum reduziert werden. In Bezug auf die Vermeidung solcher Bedrohungen beschreibt Prof. Pfitzmann in [Pf00], wie die Bedrohung von Computersysteme durch Viren, Würmer und trojanische Pferde theoretisch auf die von *transitiven* trojanischen Pferden reduziert werden kann.
Betrachtet man den Ort, an dem ein beabsichtigter Angriff ausgeübt wird und ob bzw. wie er sich auf das betroffene System auswirkt, kann man eine weitere Unterscheidung zwischen *aktiven* und *passiven* beabsichtigten Bedrohungen durchführen [Ho03].

Passive beabsichtigte Bedrohungen beschränken sich darauf, ein System und deren Informationen auszuspionieren, ohne es dabei selbst direkt zu beeinflussen. Dazu gehören unter anderem folgende:

- Auswertung der elektromagnetischen Abstrahlung von Computersystemen
- Einsatz von so genannten "Packet Sniffern"
- Informationsgewinn durch "Social Engineering" (Ausnutzung von Beziehung zu bestimmten Personen)

Eine Absicherung gegen diese Art des unberechtigten Informationsgewinns durch Angreifer kann mit Hilfe des Fachbereichs der Datensicherheit erfolgen. Dazu gehört z.B. die Verschlüsselung von Daten oder das Aufteilen von Geheimnissen in Teilgeheimnisse z.B. unter Verwendung des Schwellwertschemas [Pf00].

Bei den *aktiven beabsichtigten Bedrohungen* handelt es sich um Bedrohungen, welche in ihrer Ausführung direkten Einfluss auf das betroffene System nehmen und daher, im Gegensatz zu den passiven Bedrohungen, in aller Regel durch den Angegriffenen direkt nachgewiesen werden können (z.B. Auswertung von Logdaten über die Wirkung, Auffinden von bösartigem Code etc.). Zu diesen Bedrohungen gehören z.B.:

- (Transitive) Trojanische Pferde
- Viren
- Würmer
- "Denial of Service"-Attacken
- "Buffer Overrun"-Attacken

Die Gefahr, die von diesen Bedrohungen ausgeht, kann in einigen Fällen vermieden (z.B. Viren) [Pf00] und in anderen Fällen auf ein Minimum reduziert werden (z.B. DoS), wenn man sich konsequent an wohluntersuchte Sicherheitsrichtlinien hält. Dazu gehört neben der Verschlüsselung und Signatur von Daten auch die richtige Konzeption und Umsetzung des Zugriffsmanagements in Betriebssystemen bzw. Anwendungssoftware.

Wird eine der beabsichtigten möglichen Bedrohungen in einem Angriff erfolgreich in die Tat umgesetzt, so ergeben sich für den Angreifer verschiedene Arten der Ausnutzung. In [Mue04] wird eine Unterteilung in folgende Kategorien vorgenommen:

- Nachahmung einer fremden Identität ("Spoofing")
- Unbefugte Änderung von Daten ("Tampering")
- Abstreitbarkeit ("Repudiation")
- Informationsenthüllung ("Information Disclosure")
- Dienstverweigerung ("Denial-of-Service")
- Erhöhung der Berechtigung ("Elevation of rights")
- Profilbildung einer Entität ("Profiling")

Bringt man diese Einordnung mit den in Kapitel 4.1 vorgestellten Schutzzielen in Verbindung, ergibt sich der in Tabelle 1 dargestellte Zusammenhang zwischen den Angriffszielen

Schutzziele	Nachahmung der Identität	Unbefugte Änderung	Abstreit-barkeit	Ent-hüllung	Dienst-ver-weigerung	Erhöhung der Berechtigung	Profil-bildung
Vertraulichkeit	X	X		X			
Integrität	X	X				X	
Authentizität	X						
Verbindlichkeit	X	X	X	X			
Verfügbarkeit					X		
Autorisierung	X				X	X	
Privatheit							X

Tabelle 1: Angriffsziele und ihr Einfluss auf die Schutzziele

und ihrem Einfluss auf das jeweilige Schutzziel. Jedes Kreuz (X) stellt dabei die mögliche Unterwanderung des Schutzziels (Zeile) durch das jeweilige Angriffsziel (Spalte) dar.

Die unterschiedlichen Angriffsziele werden meist kombiniert, um einen konkreten Angriff durchzuführen. Als Beispiel könnte ein Angreifer eine fremde Identität nachahmen um dann "unbefugte" Änderungen durchführen zu können.

4.3 Statistik

Die ständig wachsende Anzahl entdeckter Sicherheitslücken in Softwareprodukten und gemeldeter Störfälle zeigen, dass auf Grund der zunehmenden Abhängigkeit von Netzwerken und der zunehmenden Komplexität und Dynamik von Software das Angriffsrepertoire und die "Spielwiese" für Cracker und Script Kiddies stetig wächst.
Die in Tabelle 2 aufgezeigte Statistik über gemeldete Schwachstellen und Störfälle im Zeit-

Jahr	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
Schwachstellen	171	345	311	262	417	1090	2437	4129	3784	3780
Störfälle	2412	2573	2134	3734	9859	21756	52658	82094	137529	???

Tabelle 2: Gemeldet Schwachstellen und Störfälle bei Software von 1995-2005 nach [Ce05]

raum von 1995-2004 beweist diesen Trend deutlich.

Zu den "Schwachstellen" gehören hauptsächlich Designfehler und Fehler bei der Implementierung, wohingegen "Störfälle" offiziell gemeldete Übergriffe auf Computersysteme darstellen.

5 Webservice - Sicherheitsbetrachtungen

Während in der Vergangenheit die überwiegende Zahl der Angriffe auf die unteren Schichten der OSI-Architektur zielten, führt die Verbreitung von Webservices mit ihren in der Applikationsebene angesiedelten Sicherheitsmechanismen, zu einer Fokusierung auch auf die oberste Ebene des OSI-Modells.
Dadurch ergibt sich die ganz neue Bedrohung, Enterprise Systeme anzugreifen, ohne dass die zur Zeit weit verbreiteten und ausgereiften Sicherheitsmechanismen der unteren OSI-Ebenen (z.B. Firewalls) Schutz bieten könnten.
Um aufzuzeigen, wo sich Angriffsmöglichkeiten gegen Webservices ergeben soll in diesem Kapitel zuerst ein Angreifermodell (s.Kap. 5.1) im Webserviceumfeld erstellt und anschließend verschiedene mögliche Angriffszenarios (s.Kap. 5.2) aufgezeigt werden.

5.1 Angreifermodell

Da ein allmächtiger Angreifer alle für ihn interessanten Daten an der Stelle ihrer Entstehung erfassen, nach Belieben unbefugt verändern und/oder das betroffene IT-System durch physische Zerstörung etc. in seiner Funktionalität beliebig beeinträchtigen könnte [Pf00], kann es vor diesem keinen Schutz geben. Alle möglichen praktisch umsetzbaren Maßnahmen führen daher nur zu einer *Annäherung an einen perfekten Schutz*.

Für die weitere Betrachtung möglicher Angriffe auf Webservices wird nun ein grobes Modell für einen Angreifer erstellt, mit dem in diesem Umfeld gerechnet werden muss und welcher für die zu entwickelnden Prototypen in Kapitel 7 und 8 angenommen wird.
Vom finanziellen Aspekt betrachtet ist mit Angreifern mit kleinem Budget (z.B. Script Kiddies, die nur herumprobieren) bis hin zu Angreifern mit sehr großem Budget (z.B. Geheimdienst) zu rechnen. Die Motivation des Angreifers hängt dabei in hohem Maße vom erhofften Nutzen aus einem erfolgreich durchgeführten Angriff ab.
Das Verhalten von Script Kiddies ist allerdings nur sehr schwer vorhersagbar, da deren Motivation oftmals in anderen Ursachen begründet liegt, als im erhofften Nutzen und sie meist über sehr viel Zeit verfügen [Gi04]. Glücklicherweise ist ihr zur Verfügung stehendes Budget und somit auch ihre Rechenkapazität begrenzt.
Die Verteilung möglicher Angreifer, die es abzuwehren gilt, ist beliebig, wobei einschränkend darauf hingewiesen werden muss, dass es nötig ist anzunehmen, dass Personen oder Maschinen mit direktem Zugriff auf den Webservice (*Insiders*) zwar auch als Angreifer in Frage kommen können, aber wegen ihrer potentiell unbegrenzten Möglichkeiten im verwendeten Model nicht als Angreifer bzw. Angriffshelfer betrachtet werden. Mit Außenstehenden wird in beliebiger Anzahl und beliebiger Verteilung gerechnet.
Die Rechenfähigkeit der erwarteten Angreifer wird als *komplexitätstheoretisch beschränkt* [Pf00] angenommen, was bedeutet, dass als schwer angenommene Probleme (z.B. in NP liegend) für diese auch *schwer* zu lösen sind.

5.2 Angriffszenarios

In diesem Kapitel soll eine Auswahl möglicher Angriffe auf Webservices vorgestellt werden. Die meisten Szenarios besitzen allerdings allgemeinen Charakter und können daher auch auf vernetzte Computersysteme allgemein angewendet werden. Dazu gehören die Angriffe auf die Verfügbarkeit (s.Kap. 5.2.1), der Angriff durch wiederholtes Senden einer abgehörten Nachricht (s.Kap. 5.2.2) und der Angriff durch Vortäuschung eines Servers (s.Kap. 5.2.3). Die Übernahme einer Sitzung (s.Kap. 5.2.4) und der Angriff, welcher durch einen "Mann in

der Mitte" durchgeführt wird (s.Kap. 5.2.5) besitzen dagegen schon einen mehr "sitzungsgebundenen" Charakter und sind somit nicht mehr auf jedes beliebige vernetzte System anwendbar.
Bei den XML-Angriffen in Kapitel 5.2.6 werden dann spezielle Angriffsmöglichkeiten, welche sich durch das XML-Format ergeben aufgezeigt. Die Falschen Parameter in Kapitel 5.2.7 können prinzipiell zwar in jeder typ- bzw. wertunsicheren Umgebung (z.B. PHP) auftreten, werden allerdings im Rahmen dieser Arbeit nur im Zusammenhang mit Webservices betrachtet.
Zum Schluss wird dann eine typische Angriffsmöglichkeit auf eine SOA vorgestellt, indem ein UDDI-Register gefälscht wird. Dieser Angriff könnte zwar prinzipiell auch unter der Vortäuschung eines Servers (s.Kap. 5.2.3) eingeordnet werden, aber auf Grund seiner Wichtigkeit für den in Kapitel 7 vorgestellten Prototypen für "authentifizierende Webservices" wird er in Kapitel 5.2.8 detailliert vorgestellt.

5.2.1 Angriff auf die Verfügbarkeit - *denial of service*

Wenn ein Webservice einen Dienst anbietet, muss der Nutzer in der Regel davon ausgehen können, dass der Dienst auch wirklich dann, wenn er benötigt wird, in einer gewissen Zeitspanne erbracht werden kann. Ist er dazu nicht in der Lage, kann in kritischen Bereichen (z.B. Echtzeitanwendungen, Medizin, Warenbestellung) ein beträchtlicher Schaden entstehen. Dieser kann von Vertrauensverlust über finanziellen Schaden bis hin zum Verlust von Menschenleben reichen.
Ein möglicher Angriff auf eine SOA könnte versuchen, genau diesen Schaden zu verursachen. Bei einer DoS-Attacke versucht der Angreifer die Erreichbarkeit eines Services zu minimieren[24] oder zumindest illegal dessen Ressourcen zu verbrauchen [Ce99]. Dabei kann er versuchen

- den Service physikalisch von der Außenwelt abzuschneiden (z.B. durchtrennen einer/mehrerer Netzwerkleitung(en), Stromleitung(en) etc.)
- eines/mehrere der vorgelagerten Netzwerke derart zu überlasten, dass der Service von einem bestimmten Teil oder von der kompletten Außenwelt abgeschnitten ist
- den Service mit Hilfe von bestimmten Eingangssignalen zu beschäftigen und so Ressourcen des Opfers zu verbrauchen (z.B. aufwendige Berechnungen, unerwartete Eingaben)
- unbemerkt ein System zweckzuentfremden (z.B. einen anonymen FTP-Server zur Lagerung/Verteilung von illegalen Software-Kopien missbrauchen).

Erfolgreiche Angriffe dieser Art in der Vergangenheit[25] zeigen, dass mit dieser Art von Attacken immer zu rechnen ist und ihre Häufigkeit auf Grund der wachsenden Vernetzung wohl eher noch zunehmen wird.
Die Art der Abwehr, die für einen Service am besten scheint (dass eine Abwehr *nicht funktioniert* erkennt man beim nächsten gelungenen Angriff; dass sie wirklich *perfekt* ist, erfährt man vielleicht nie), unterscheidet sich je nach Typ und Aufgabe des zu schützenden Webservices. In einem Fall reicht es z.B. einen gewissen Grad an Service-Redundanz hinzuzufügen (z.B. bei einem Service, der nur Suchanfragen für einen öffentlich zugänglichen

[24] Reicht von einer Unerreichbarkeit durch einzelne Benutzer bis hin zu einer völligen Abtrennung des Services von der Außenwelt.
[25] Microsoft, Ebay, Google, Mozdev, Gibson Research Center, um nur einige zu nennen.

Produktkatalog entgegen nimmt und bearbeitet) bei einem anderen Fall genügt z.B. eine vorgelagerte wohlkonfigurierte Firewall (z.B. Absicherung eines Intranet-Services gegen Angriffe von "außen").
Bei Angriffen, die einen Service oder den Server dadurch attackieren, indem sie ihn mit einer Vielzahl von Anfragen konfrontieren oder durch Eingaben versuchen einen Teil seiner Ressourcen aufzubrauchen, kann eine Ressourcen-Beschränkung oder/und eine Mustererkennung auf der Transport- bis zur Applikationsebene (OSI-Modell) Abhilfe schaffen.
In der Regel ist jedoch für eine solide Absicherung eine Kombination aus mehreren Maßnahmen notwendig.
Um einen Angriff "hinterher" korrekt analysieren zu können, ist es in jedem Fall ratsam einen permanenten Logging-Mechanismus einzusetzen. Vielleicht geben dessen Ausgaben einen Hinweis auf eine Verbesserung der eigenen Verteidigung.

5.2.2 Wiederholtes Senden einer Nachricht - *replay attack*

Dieser Angriff ist überall dort denkbar, wo in einer Kommunikationsbeziehung als Reaktion auf eine bestimmte Nachricht durch einen authorisierten Nutzer ein bestimmtes Verhalten beobachtet wurde, welches der Angreifer noch einmal auslösen möchte. Um dies zu verhindern reicht eine Sender-Authentikation nicht aus. Jede Nachricht, die abgehört und erneut gesendet wird besitzt einen korrekten Nachweis über die Authentizität des Senders. Was hier benötigt wird, ist eine Möglichkeit zusätzlich jede einzelne Nachricht auf ihre Einmaligkeit hin zu prüfen (Nachrichten-Authentikation). In der SOAP-SEC Spezifikation (Vorgänger von WS-Security) beschreibt man diesen Sachverhalt folgendermaßen:

> "*Digital signatures alone do not provide message authentication. One can record a signed message and resend it (replay attack). To prevent this type of attack, digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, such as nonces or time stamps.*"

Es wird vorgeschlagen, so genannte Nonce-Werte, Zeitstempel bzw. Zähler zu verwenden, um einen Angriff durch wiederholtes Übermitteln einer Nachricht zu vermeiden. Das Ergebnis ist eine Authentikation mit Hilfe einer digitalen Signatur über Daten, welche sich in jeder Nachricht eindeutig unterscheiden. Der Erhalt einer Nachricht mit dem identischen Nonce-Wert, Zeitstempel oder Zählerstand, wie eine vorher erhaltenen, würde diese als Replay-Attacke[26] entlarven.

5.2.3 Manipulation eines Servers - *server spoofing*

Bei dieser Art eines Angriffs werden Zugriffe, welche an einen bestimmten Server gerichtet waren auf einen Server des Angreifers umgeleitet. Dazu werden z.B. Routingtabellen oder DNS-Einträge meist in Zusammenhang mit einer Manipulation von IP-Adressen verändert. Dieser Angriff ist nur schwer zu erkennen, da der gefälschte Server in aller Regel eine exakte Kopie des Originalservers ist und nur in einige Punkten für den Angriff verändert wurde.
Ein interessanter "Anwendungsfall" ist der so genannte *DNS-Angriff*. Dieser Angriff beruht darauf, dass es für einen Menschen kaum möglich ist, sich ein bzw. sogar mehrere IP-Adressen[27] einzuprägen.
Daher wurde in den 70er Jahren zuerst beim so genannten ARPANET [Ha05] eine Datei

[26] Mit hoher Wahrscheinlichkeit, da z.B. auch ein Netzwerkfehler zum mehrmaligen Erhalt ein und der selben Nachricht führen kann.

[27] Zieladressen für Computersysteme bestehend aus 4 Ziffern zwischen 0 und 255.

mit dem Namen *Hosts.txt*, welche anfangs ein bis zweimal wöchentlich erstellt wurde und vom NIC (Network Information Center) bezogen werden konnte, auf dem lokalen Computer abgelegt, mit dessen Hilfe eine Übersetzung von Domainnamen auf die zugehörigen IP-Adressen erfolgte.
Mit der immer größer werdenden Zahl von Nutzern wurde allerdings sehr schnell klar, dass ein verbessertes System zur eindeutigen Namensraumverwaltung geschaffen werden musste. Paul Mockapetris [An05], vom USC's Information Science Institute entwickelte daher das DNS (Domain Name System), welches Domainnamen in einer hierarchischen Anordnung verwaltet. Anfragen zur Auflösung eines leicht durch einen Menschen einprägbaren Domainnamen (z.B. www.google.de) in eine computerverständliche IP-Adresse werden heute von hierarchisch angeordneten DNS-Servern bearbeitet.
Ein DNS-Angriff macht sich diesen Umstand zunutze, indem er eine Anfrage zur Auflösung eines Domainnamens manipuliert und eine falsche IP-Adresse als Ergebnis zurücksendet. Der Nutzer merkt im ungünstigsten Fall nichts von dieser Manipulation und kommuniziert fortan mit einem falschen Dienst.

5.2.4 Übernahme einer Session - *session theft*

Bei dieser Angriffsart wird eine zwischen einem Klienten und einem Server etablierte Sitzung durch einen Angreifer übernommen. Sitzungen werden vom Klienten und vom Service anhand einer bestimmten Identifikation (*SessionID*), welche mit jeder Anfrage und Antwort mitgeschickt wird, erkannt. Durch den daraus erhaltenen Sitzungskontext erfolgt eine Zuordnung zu einer bestimmten Entität, welche sich vor der Erzeugung der Sitzung in einer bestimmten Art und Weise authentifiziert hat.
Bei der Übernahme einer solchen Sitzung gelangt der Angreifer somit in die Lage, sich gegenüber der anderen Partei als der ursprüngliche Nutzer ausgeben und in dessen Namen Aktionen durchführen zu können (z.B. Überweisung von einem Online-Konto).
Zur Abwehr solcher Angriffe hat das verwendete Kommunikationssystem dafür Sorge zu tragen, dass bei sicherheitskritischen Sitzungen die Sitzungskennung nur verschlüsselt übertragen wird und dass ein Erraten der Sitzungskennung nahezu unmöglich ist. Weiterhin können zur Absicherung der Sitzung zusätzliche Informationen, wie z.B. die IP des Computersystems, welches im Auftrag des Nutzers operiert, abgespeichert werden. Diese zusätzlichen Daten können allerdings die Flexibilität des Kommunikationssystems einschränken und sind je nach Einsatzfall neu zu erarbeiten und zu bewerten.

5.2.5 Mann in der Mitte - *man in the middle*

Wenn es einem Angreifer gelingt, einen fremden Kommunikationskanal soweit unter seine Kontrolle zu bringen, das die "Abgehörten" nicht feststellen können, ob sie tatsächlich miteinander oder mit dem Angreifer kommunizieren [Mue04], bezeichnet man das als einen *Mann-in-der-Mitte-Angriff*. Der Angreifer wird dadurch in die Lage versetzt, sich unberechtigten Zugang zu Informationen zu verschaffen, sowie komplette Datenverbindungen beliebig zu manipulieren (*connection hijacking*).
Die beste Möglichkeit für die Durchführung eines solchen Angriffs bietet sich bei der Verwendung von sicheren Kanälen in der Phase des Schlüsselaustauschs. Gelingt es dabei einem Angreifer mit den Kommunikationspartnern Schlüssel derart auszutauschen, dass diese glauben dies mit dem jeweils anderen Partner getan zu haben, so ist der Angreifer in der Lage unbemerkt jede ausgetauschte Information im Klartext abzuhören. Dies erreicht er dadurch, indem er die verschlüsselten Daten mit dem in die jeweilige Beziehung

eingeschleusten Schlüssel entschlüsselt und unter Verwendung des in die andere Beziehung eingeschleusten Schlüssels wieder verschlüsselt an den Empfänger weiterschickt.

5.2.6 XML-Angriffe - *XML attacks*

Jede Kommunikation unter Verwendung von XML-Nachrichten (z.B. SOAP) besitzt Schwächen, über die jeder Entwickler auf diesem Gebiet Bescheid wissen sollte.

Der wichtigste Punkt bei der Überprüfung von XML-Dokumenten ist das Wissen darum, dass sein Inhalt nahezu beliebig formatiert werden kann. Dadurch wird die Überprüfung durch einfachen Zeichenkettenabgleich (*string matching*) erschwert, wenn nicht sogar unmöglich gemacht.
Eine verbreitete Angriffsart ist das Übermitteln von XML-Dokument mit extremer Größe bzw. Tiefe (z.B. <param><param><param>...</param></param></param>). Der verarbeitende Service kann dadurch, falls er z.B. einen DOM-Parser (komplettes Objektmodel im Speicher) verwendet, ein Problem mit seinem Ressourcenverbrauch bekommen, was unter Umständen in einem Totalausfall (DoS-Angriff; s.Kap. 5.2.1) enden kann.
Diese Art des Angriffs wird dadurch begünstigt, dass es wesentlich einfacher ist ein so geartetes Angriffsdokument zu erstellen, als es zu verarbeiten.

5.2.7 Falsche Eingaben/Parameter - *defective parameter*

Bei diesem Angriff geht der Angreifer davon aus, dass ein Webservice eventuell nicht mit falschen[28] oder falsch formatierten Eingaben rechnet.
Ziel eines solchen Angriffs könnte es sein, den Webservice zu "verwirren" und so möglicherweise ein nicht vorgesehenes, evtl. sogar bösartiges Verhalten hervorzurufen oder den Service einfach "nur" zu deaktivieren (siehe DoS-Angriff in Kap. 5.2.1).
Insbesondere Webservices müssen mit dieser Attacke rechnen, da sie (1.) ein besonders "lohnendes" Ziel darstellen, um in ein Enterprise-System einzubrechen (schließlich ist ein Ziel der SOA die Abwicklung von Geschäften inkl. ihrer Bezahlung und es ist daher sehr wahrscheinlich, dass ein solcher Webservice weitreichende "Vollmachten" besitzt) und (2.) Webservices Feinheiten ihrer Schnittstellen per Definition in so genannten Deskriptor-Dateien (WSDL, siehe Kap. 3.4) zur Verfügung stellen[29].
Diese Datei beschreibt genau, mit welchen Eingangsdaten der Webservice bei einer Anfrage rechnet. Diese Tatsache lädt mögliche Angreifer geradewegs dazu ein, inkorrekte Daten zu schicken und "zu schauen, was passiert".
Eine WSDL-Datei könnte z.B. folgende Zeile beinhalten:

```
<xsd:element name="title" type="string"/>
```

Dadurch wird angegeben, dass einer der erwarteten Parameter (*title*) vom Typ *string* ist. Der Angreifer könnte nun z.B. versuchen ein abnormales Verhalten hervorzurufen, indem er *NULL*, einen die Maximallänge des Typs überschreitenden Text, Sonderzeichen (Wildcards, Escape-Zeichen etc.), Programmcode (SQL-Anweisungen, Skripten etc.) usw. oder eine Mischung daraus an den Service schickt.
Aus diesem Grund ist es wichtig eine Überprüfung der "Vernünftigkeit" (*sanity checks*) von an einen Service gerichteten Daten durchzuführen. Dies könnte z.B. vor der Weitergabe der

[28] Die Einstufung einer Eingabe als *falsch* kann natürlich nur im Rahmen des zugehörigen definierten Typ- bzw. Wertebereichs erfolgen.

[29] Die Sichtbarkeit der WSDL-Datei kann dabei natürlich von *öffentlich* bis *privat* variieren. Dies spielt aber an dieser Stelle keinerlei Rolle, da ein Angriff sowohl von "außen" als auch von "innen" erfolgen kann.

Daten an den Service durch eine Validierung gegen ein XML-Schema (Vorteil: Trennung von Definition und Verwendung; aber: *prozessorintensiv*, da der XML-Request geparst werden muss) oder im Webservice selbst vor einer Weiterverarbeitung erfolgen.

5.2.8 Gefälschtes UDDI-Register - *fake-UDDI*

Wie bereits in Kapitel 3.7.1 beschrieben, stellt ein UDDI-Register einen Platz dar, wo Informationen über Webservices durchsuchbar abgelegt werden. Nutzer, die einen bestimmten Dienst in Anspruch nehmen möchten, wenden sich in aller Regel an einen solchen Service. Durch diese zentrale Stellung innerhalb einer SOA werden diese Register zu einem sehr interessanten Angriffspunkt und bedürfen einer speziellen Sicherheitsbetrachtung.

In der Welt des UDDI existieren zwei maßgebliche Gruppen von Nutzern, die über die Indirektion des UDDI-Registers miteinander kommunizieren. Da wären auf der einen Seite die *Suchenden*, die einen bestimmten Dienst auffinden möchten und auf der anderen Seite die *Veröffentlicher*, deren Aufgabe es ist, Services zu publizieren. Sie kennen sich in der Regel nicht persönlich und müssen sich darauf verlassen, was ihnen durch das UDDI an Informationen zur Verfügung gestellt wird.
Ein Angreifer könnte diesen Umstand dazu verwenden, dem Suchenden Daten zu präsentieren, die durch ihn geschickt manipuliert worden sind. Bis zur UDDI Version 2 war es einem Nutzer nicht möglich zu prüfen, ob ein Ergebnis seiner Anfrage genau die Daten enthält, die durch den Veröffentlicher vorgegeben wurden.
Denkbar wären Szenarios, in denen der Veröffentlicher oder der Suchende mit einem "falschen" UDDI-Register kommunizieren oder das korrekte Register mutwillig oder zufällig falsche Daten herausgibt. Als Folge dessen könnten z.B. Passwörter, Kreditkartendaten usw. in falsche Hände gelangen, obgleich diese den eigenen Kontrollbereich niemals im Klartext verlassen sollten und dieser Angriff in der realen Welt nur einen Teil eines größeren Angriffs darstellen sollte.
UDDI Version 3 adressiert dieses Problem, indem es Signaturen für die Basis-Datentypen *businessEntity*, *businessService*, *bindingTemplate* und *tModel* einführt. Diese ermöglichen es dem Endnutzer eine Validierung/Authentifizierung von Anfrageergebnissen durchzuführen. Damit verliert das beschriebene Angriffsszenario an Brisanz, obgleich die immer noch weite Verbreitung des UDDI Version 2 Angriffe diesen Typs zulassen.

6 Webservice - Sicherheitsmechanismen

In diesem Kapitel werden bereits existierende Sicherheitsmechanismen für Webservices vorgestellt, welche in den meisten Fällen unter der Aufsicht von Oasis [OASIS] bzw. dem W3C [W3C] entwickelt werden.
Dazu gehören neben der Erstellung von Signaturen (XML-Signature, s.Kap. 6.1) und des Vorgangs der Verschlüsselung im XML-Format (XML-Encryption, s.Kap. 6.2) auch die Übertragung von Authentifikationsannahmen (SAML, s.Kap. 6.3) und die Möglichkeiten deren Einbettung in eine SOAP-Nachricht (WS-Security, s.Kap. 6.4).
Weiterhin wird für eine systemunabhängige Darstellung von Zugriffsregeln für Ressource die Beschreibungssprache XACML (s.Kap. 6.5) vorgestellt.
Eine erleichterte Verwendung von PKI-Infrastruktur wird durch das in Kapitel 6.6 vorgestellte Framework XKMS vorgestellt.
WS-SecureConversation (s.Kap. 6.7) ermöglicht die Erstellung von sitzungsgebundenen Sicherheitskontexten und verringert damit den Aufwand für die Authentifizierung und die Verschlüsselung von Daten über die Dauer einer Sitzung hinweg.
Das in Kapitel 6.8 vorgestellte Szenario für das Zusammenspiel der vorgestellten Sicherheitstechnologien soll deutlich machen, dass diese zwar in kleinen Szenarios prinzipiell auch einzeln einsetzbar sind, aber die große Stärke für komplexe Szenarios in ihrer Verbindung liegt.
Anschließend werden in Kapitel 6.9 verschiedene weitere Sicherheitstechnologien im Webservice-Umfeld kurz vorgestellt, welche für diese Arbeit nur eine untergeordnete Bedeutung besitzen.

6.1 Signaturen im XML-Format - *XML-Signature*

Der XML-Signature Standard [XSig] war der erste XML-Sicherheitsstandard, der den "Recommendation-Status" (fertiger Standard un für die Verwendung empfohlen) erreicht hat. Die aktuellste Version wurde vom W3C [W3C] in Zusammenarbeit mit dem IETF (Internet Engineering Task Force) [IETF] im Februar 2002 freigegeben und beschreibt, wie digitale Signaturen im XML-Datenformat dargestellt werden können. Dadurch wird die Integrität der übermittelten Daten auf Anwendungsebene gesichert.
Auf anderen OSI-Ebenen existieren eine Vielzahl verschiedener Protokolle (z.B. SSL, IPSec), welche die Integrität auf dem Kommunikationskanal sichern. Ein großer Nachteil dieser Sicherheitsmechanismen ist aber der fehlende Integritätsschutz bei persistenten Daten (Festplatte, Datenbank etc.) und der Umstand, dass die Daten bei einer Kommunikation über mehrere Intermediaries dort temporär ohne jeglichen Schutz vorliegen und eine Ende-zu-Ende-Integritätswahrung somit nicht umsetzbar ist.
An dieser Stelle setzt *XML-Signature* an, indem es einen Mechanismus für den persistenten bzw. Ende-zu-Ende-Integritätsschutz bietet. Weiterhin ermöglicht XML-Signature in Verbindung mit einer identitätsbasierten Sicherheits-Infrastruktur (z.B. PKI; s.Kap. 2.8) auch die Sicherheitsziele Authentikation/Autorisation und Verbindlichkeit zu realisieren.
XML-Signature definiert dabei keinerlei neue Signaturalgorithmen, sondern verwendet Standard-Algorithmen, welche auch in vielen anderen Technologien Anwendung finden (z.B. SHA-1 für Hashs und RSA oder DSA für Signaturen).
Die Signaturmöglichkeiten sind nicht, wie man fälschlicherweise annehmen könnte, auf bloße XML-Inhalte beschränkt. Vielmehr können beliebige Daten digital signiert und die entstandene Signatur dann mit Hilfe von XML dargestellt werden.
Bereits im Vorfeld dieser Spezifikation war es möglich, XML-Dokumente im Ganzen zu

signieren (z.B. PKCS#7) und die entstandene Signatur an das Dokument zu binden (z.B. S/MIME). Jedoch konnte mit dieser Methode die Signatur als solche nicht im XML-Format dargestellt und keine XML-Teildokumente signiert werden.
Genau an dieser Stelle liegt die große Relevanz von XML-Signature für Webservices: verbindungsüberdauernde Integrität mit variabler Granularität.
Es existieren drei Möglichkeiten der Einbettung einer Signatur in ein XML-Dokument:

enveloped signature: Die Signatur befindet sich innerhalb des zu signierenden XML-Dokuments bzw. -Elements:

```
<IntegritaetErhaltendesElement>
    <Datum_1/>
    <Datum_2/>
    <ds:Signature/>
</IntegritaetErhaltendesElement>
```

Vor einer Überprüfung der Signatur durch den Empfänger ist es allerdings notwendig diese zu entfernen, da die Validierung sonst, auf Grund des bei der Erzeugung nicht vorhandenen Signatur-Elements, offensichtlich fehlschlagen würde. Dies kann durch eine Angabe der Algorithmusdefinition *http://www.w3.org/2000/09/xmldsig#enveloped-signature* im *Transform*-Element für den Empfänger am einfachsten ausgewiesen werden.
Es besteht allerdings auch die Möglichkeit die notwendige Transformation mit Hilfe eines etwas umfangreicheren XPath-Statements anzugeben [On02]:

```
<XPath xmlns:ds="&ds;">
    count(ancestor-or-self::ds:Signature |
    here()/ancestor::ds:Signature[1]) >
    count (ancestor-or-self::ds:Signature)
</XPath>
```

enveloping signature: Die Signatur befindet sich ebenfalls im selben XML-Dokument, wie der zu signierende Inhalt, jedoch *außerhalb* des zu signierenden Elements:

```
<ds:Signature>
    <ds:SignedInfo>
        <...>
        <ds:Reference URI="#SigId1"/>
    </ds:SignedInfo>
    <...>
</ds:Signature>
<ds:Object Id="SigId1">
    <IntegriteatErhaltendesElement>
        <Datum_1/>
        <Datum_2/>
    </IntegriteatErhaltendesElement>
</ds:Object>
```

Das *Object*-Element umfasst dabei den Inhalt, auf den sich die Signatur bezieht (*URI*-Attribut des *Reference*-Elements).

detached signature: Die Signatur referenziert ein beliebiges externes Dokument:

```
<ds:Signature>
    <ds:SignedInfo>
        <...>
        <ds:Reference URI="http://www.w3.org/TR/xml-stylesheet"/>
    </ds:SignedInfo>
    <...>
</ds:Signature>
```

In diesem Beispiel wurde die W3C XSL-Spezifikation signiert.

Der Aufbau eines *Signatur*-Elements sieht wie in Abb. 8 dargestellt aus. Die aktuellste

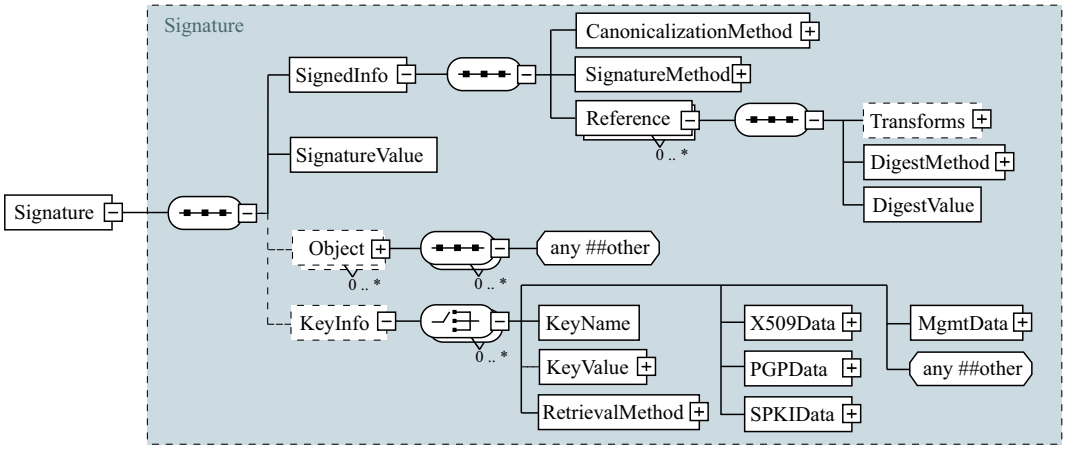


Abbildung 8: Ausschnitt aus der Struktur des *Signatur*-Elements einer XML-Signatur

Schema-Definition für diesen Aufbau ist auf den Webseiten vom [W3C] zu finden.
Das standardmäßig und auch im weiteren Verlauf dieser Arbeit verwendetet Namespace-Prefix für diese XSD lautet "*ds*" (digital signature).
Das *Signature*-Element stellt das äußerste Element einer Signatur-Definition dar. Das untergeordnete *SignatureValue*-Element enthält die errechnete Signatur und innerhalb des *Object*-Elements befindet sich bei einer "enveloping signature" das zu signierende Element, welches völlig beliebig strukturiert sein kann.
Innerhalb des *KeyInfo*-Elements befinden sich alle Informationen, die für eine Überprüfung der digitalen Signatur mit dem öffentlichen Schlüssel notwendig sind (z.B. RSA-Modulus, RSA-Exponent, X.509-Zertifikat, X.509-Aussteller-Informationen usw.).
Zum Auffinden des Testschlüssels kann es auch ausreichend sein, mit Hilfe des *KeyName*-Elements eine Phrase anzugeben, welche es dem Empfänger ermöglicht, den richtigen Schlüssel auszuwählen bzw. zu finden (z.B. der Name für eine XKMS-Anfrage oder einfach nur der Text "Schlüssel von unserem letzten Treffen").
Das *SignedInfo*-Element enthält alle wichtigen Informationen darüber, welcher Signatur-Algorithmus verwendet wurde (*SignaturMethod*-Element), was signiert wurde (*Reference*-Elemente) und welche Kanonisations-Methode (s.Kap. 2.6) angewendet wurde (*CanonicalizationMethod*).

Der Vorgang der Signatur-Erzeugung läuft folgendermaßen ab:

1. mit Hilfe des *Reference*-Elements festlegen, *was* signiert werden soll (über *URI*-Attribut oder indirekt über den Signatur-Kontext bei "enveloped signature").

2. sich auf eine Signatur-, Hash- und Kanonisationsmethode festlegen (die Signatur wird, wie bereits in Kap. 2.3.5 beschrieben, nicht über der Nachricht selbst, sondern über ihrem Hash berechnet) und diese über standardisierte URIs auf Algorithmussynonyme in den *Algorithm*-Attributen der *DigestMethod*-Elemente, des *SignatureMethod*-Elements bzw. des *CanonicalizationMethod*-Elements festhalten.

3. vorzunehmende Transformationen im *Transforms*-Element festhalten. Dabei bestehen folgende Möglichkeiten:

   (a) Daten wurden nach dem Signieren verschlüsselt.

   (b) Daten wurden nach dem Signieren komprimiert.

   (c) Daten wurden selektiv signiert. So können z.B. nur "alle *Person*-Elemente" eines XML-(Teil)Dokuments oder "alles außer *Log*-Elemente" signiert werden. Zur Spezifikation solcher Element-Auswahlen wird XPath [XPath] verwendet.

   Diese Transformationen müssen offensichtlich *vor* einer Überprüfung der Signatur entsprechend umgekehrt bzw. im Falle der Selektion einzelner Elemente ebenfalls getroffen werden.

4. den berechneten Hash über die jeweilige Information im zugehörigen *DigestValue*-Element festhalten

5. die berechnete Signatur über den Hash-Wert des vorher kanonisierten *SignedInfo*-Elements im *SignatureValue*-Element festhalten. Dadurch werden automatische sämtliche *Reference*-Elemente, also eine *beliebige* Anzahl von Nachrichten-Hashs, mit einer einzigen Signatur versehen.

Die Überprüfung der Signatur beim Empfänger erfordert einen ähnlichen Ablauf:

1. mit Hilfe des *SignatureMethod*-Elements feststellen, welcher Signatur- und Hashalgorithmus für die Erstellung der Signatur verwendet wurde.

2. den Hash über das vorher kanonisierte *SignedInfo*-Element mit dem entsprechenden Hashalgorithmus berechnen.

3. mit Hilfe des öffentlichen Schlüssels vom Unterzeichner aus der Signatur im *SignatureValue*-Element den originalen Hashwert über das *SignedInfo*-Element bestimmen und mit dem in Schritt 2 berechneten Hashwert vergleichen.

4. alle nunmehr auf Integrität geprüften *Reference*-Elemente mit Hilfe ihrer enthaltenen *DigestMethod*- und *DigestValue*-Elemente gegen die Original-Dokumente bzw. -Element prüfen.

Wurden alle angegebenen Schritt erfolgreich und übereinstimmend durchgeführt, so kann entsprechende der Sicherheit der verwendeten Hash- und Signaturalgorithmen eine Aussage darüber gemacht werden, mit welcher Wahrscheinlichkeit keine Modifikation an den signierten Daten durchgeführt worden ist.
XML-Signature erlaubt auch die Verwendung mehrerer Signatur-Ebenen, was bedeutet, dass für den selben zu signierenden Inhalt unterschiedlich Signatur-Semantiken möglich sind. Z.B. können bestimmte Daten von mehreren Personen signiert, mitunterzeichnet (Signaturen über Signaturen), bezeugt, notariell beglaubigt etc. werden.

6.2 Verschlüsselung im XML-Format - *XML-Encryption*

XML-Encryption ist eine Spezifikation des W3C, welche sich seit Dezember 2002 in ihrer aktuellsten Version befindet.
Mit ihrer Hilfe wird festgelegt, wie verschlüsselte Daten in einem XML-Dokument standardisiert dargestellt werden können. Es werden allerdings auch hier keine neuen Algorithmen oder Verschlüsselungstechniken definiert.
Es können nicht nur XML-Inhalte[30], sondern prinzipiell beliebige digitale Daten verschlüsselt werden.
Wie bereits bei XML-Signature erwähnt, existieren auch für Vertraulichkeit in anderen OSI-Schichten Modelle, welche diese für in Punkt-zu-Punkt-Transit befindliche Daten garantieren (SSL, IPSec etc.). Jedoch existiert auch hier das Problem, dass die Daten diesen Schutz in der Anwendungsschicht verlieren und somit auf dem Ausgangs- bzw. Zielrechner ausgespäht werden können. Die Verwendung von Intermediaries führt dazu, dass die Daten temporär unverschlüsselt vorliegen und eine Ende-zu-Ende-Verschlüsselung damit nicht umsetzbar ist.
XML-Encryption beseitigt diesen Umstand, indem es die verschlüsselten Daten direkt in ein XML-Dokument (z.B. eine SOAP-Nachricht) einbettet. Dadurch wird der Verschlüsselungsschutz für Daten im lokalen bzw. persistenten Zustand und Ende-zu-Ende-Verschlüsselung über mehrere Intermediaries hinweg garantiert.
Bereits im Vorfeld von XML-Encryption war es möglich ein gesamtes XML-Dokument zu verschlüsseln (z.B. Emails per S/MIME). Jedoch wird es mit dieser Spezifikation ermöglicht, auch einzelne Teile eines XML-Dokuments zu verschlüsseln und den gewonnenen Schlüsseltext im XML-Format darzustellen.

Es existieren drei mögliche Verschlüsselungsmodi:

Verschlüsselung eines XML-Elements: In diesem Modus wird ein beliebiges XML-Element verschlüsselt und durch das entstandene *EncryptedData*-Element ersetzt:

```
<BezahlungsInformation>
    <Name>Max Muster</Name>
    <EncryptedData/>
</BezahlungsInformation>
```

In diesem Beispiel wurden Kreditkarten-Daten verschlüsselt. Diese Art der Verschlüsselung wird mit Hilfe des Wertes *http://www.w3.org/2001/04/xmlenc#Element* im *Type*-Attribut des *EncryptedData*-Elements bekannt gemacht.

Verschlüsselung des Inhalts eines XML-Elements: Falls nicht das Element selbst, sondern nur dessen Inhalt verschlüsselt werden soll, Greift man auf diesen Verschlüsselungsmodus zurück:

```
<BezahlungsInformation>
    <Name>Max Muster</Name>
    <KreditKarte Waehrung="EUR">
        <EncryptedData/>
    </KreditKarte>
</BezahlungsInformation>
```

[30] Die Unicode-Zeichen eines XML-Dokuments müssen vor einer Verschlüsselung in UTF-8 (Unicode Transformation Format) umgewandelt werden. UTF-8 bildet die Unicode-Zeichen auf Oktetts (bytes) ab.

In diesem Beispiel wurde nicht das gesamte *KreditKarte*-Element verschlüsselt, wie im vorangegangenen Modus, sondern nur die enthaltenen Elemente (z.B. *KartenNummer*, *Aussteller* etc.). Mit Hilfe des Wertes *http://www.w3.org/2001/04/xmlenc#Content* im *Type*-Attribut des *EncryptedData*-Elements wird dieser Modus gewählt.

Verschlüsseln beliebiger Daten: Werden beliebige Daten (z.B. XML- oder PDF-Dokumente, Bilder etc.) verschlüsselt, so sollte das *Type*-Attribut des *EncryptedData*-Elements eine Definition des verschlüsselten Datentyps enthalten. Standardmäßig werden dafür Spezifikationen der IANA (Internet Assigned Number Authority) unter *http://www.isi.edu/in-notes/iana/assignments/media-types* verwendet. Dies ist allerdings nicht zwingend notwendig. Die Typ-Definition kann auch beliebig anders definiert (z.B. eigenes Typsystem) oder gar weggelassen werden. Voraussetzung für Letzteres wäre dann aber eine anderweitige Erkennungsmöglichkeit (z.B. aus dem Kontext oder an Hand der ersten Binär-Bytes).

Der Aufbau eines *EncryptedData*-Elements sieht für eine verschlüsseltes Datum wie in Abb. 9 dargestellt aus. Die aktuellste Schema-Definition für diesen Aufbau ist auf den Webseiten

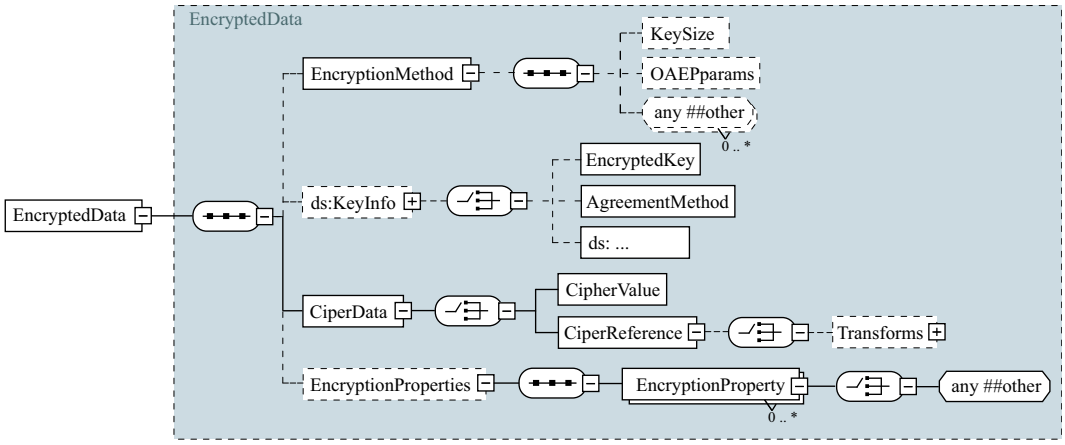


Abbildung 9: Ausschnitt aus der Struktur des *EncryptedData*-Elements bei einer XML-Verschlüsselung

vom [W3C] zu finden.
Das *EncryptedData*-Element, als äußerstes Element eines verschlüsselten Datums, kann mit Hilfe des *Type*-Attributs die Art der Verschlüsselung (siehe oben), mit dem *MimeType*-Attribut den MIME-Type des verschlüsselten Datums (z.B. image/png) und mit dem *Encoding*-Attribut eine spezielle Codierung (z.B. base64) zum Ausdruck bringen.
Mit Hilfe des *EncryptionMethod*-Elements wird spezifiziert, welcher Verschlüsselungsalgorithmus verwendet wurde (z.B. RSA für die Verschlüsselung eines symmetrischen Schlüssels, AES für die Verschlüsselung der Daten und CBC (Cipher Block Chaining) als Verkettung der einzelnen Blöcke), welche Schlüssellänge festgelegt wurde und welche zusätzlichen Parameter relevant sind.
Das *ds:KeyInfo*-Element ist, wie der verwendete Namensraum verrät, aus der Spezifikation der *XML-Signature* (s.Kap. 6.1) übernommen und kann somit sämtliche Informationen für den Transport eines öffentlichen Schlüssels bzw. Zertifikates übernehmen. Da dies bei einer Verschlüsselung allerdings nicht üblich ist, ist dieses Element optional.
Weiterhin wurde zur Unterstützung hybrider Verschlüsselungmechanismen das Unterele-

ment *EncryptedKey* eingeführt, welcher dann den asymmetrisch verschlüsselten symmetrischen Schlüssel enthält. Damit der Empfänger den passenden privaten Schlüssel zur Entschlüsselung auswählen kann, enthält dieses Element auch entsprechende Informationen im enthaltenen *KeyInfo*-Element.
Das *AgreementMethod*-Element enthält Informationen darüber, wie die Verständigung auf den verschlüsselten Schlüssel erfolgte (z.B. Diffie-Hellmann [Pf00]).
Das *CipherData*-Element ist das einzige Element, welches immer vorhanden sein muss. Es enthält entweder die Darstellung der verschlüsselten Information selbst (*CipherValue*-Element) oder für den Fall, das z.B. die zu verschlüsselten Daten sehr groß sind und deshalb besser in einer eigenen Datei abgespeichert sein sollten, eine Referenz (URI) auf diese Ressource (*CipherReference*-Element). Innerhalb des *CipherReference*-Elements können auch Transformationen angegeben werden. Im Vergleich zu XML-Signature existieren dabei allerdings Unterschiede. Wurden bei XML-Encryption die Transformationsschritte im Großen und Ganzen vom Signierer und bei der Prüfung in der selben Reihenfolge abgearbeitet, so werden sie bei XML-Encryption vom Ver- und Entschlüsseler in entgegengesetzter Reihenfolge durchgeführt. Z.B. könnte ein Video vom Verschlüsseler verschlüsselt, base64 kodiert und in einem Dateisystem abgespeichert werden. Der Entschlüsseler würde demzufolge die Daten aus dem Dateisystem holen, den Oktett-Strom base64 dekodieren und anschließend entschlüsseln.
Das *EncryptionProperties*-Element bietet eine Möglichkeit, Eigenschaften, welche keine vorgegebenen Deklarationsmöglichkeit innerhalb des *EncryptedData*-Elements besitzen, darzustellen.
Mit Hilfe der XML-Encryption-Spezifikation ist es auch möglich, bereits verschlüsselte Inhalte (*EncryptedData*-Elemente) beliebig oft wieder zu verschlüsseln. Dadurch entstehen so genannte "mehrfach verschlüsselte Dokumente" (*super-encrypted*). Dabei ist allerdings darauf zu achten, dass laut Spezifikation keine separate Verschlüsselung einzelner Unterelemente von *EncryptedData* durchgeführt werden darf. Einem *EncryptedData*-Element ist es demzufolge nicht erlaubt, Kindelement eines anderen *EncryptedData*-Elements zu sein.

6.3 Vertrauen auf Reisen - *SAML*

SAML (Security Assertion Markup Language)[31] ist eine Spezifikation der OASIS [OASIS] und befindet sich seit September 2003 mit der Version 1.1 im "Recommendation-Status". Die Version 2.0 ist zur Zeit in Arbeit und hat den "Commitee-Status" erreicht, befindet sich also noch in der Entwicklung und ist noch nicht für eine Verwendung empfohlen.
Mit Hilfe von SAML wird es ermöglicht, Vertrauensbehauptungen (*trust-assertions*) eines Systems im XML-Format zu formulieren und für den Benutzer meist unsichtbar im Hintergrund an andere Systeme weiterzugeben (*portable trust*). Entsprechend formulierte Assertions (standardmäßiges XML-Prefix: "saml") können Aussagen über eine in der Vergangenheit stattgefundene Authentikation bzw. Autorisation, sowie verschiedene Eigenschaften einzelner Entitäten (z.B. Personen oder Computer-Systeme) enthalten. Die Glaubwürdigkeit dieser Angaben basiert natürlich auf dem gegenseitig entgegengebrachten Vertrauen der Partei, welche Behauptungen über eine Entität aufstellt und übermittelt und der Partei, welche diese Behauptung verarbeitet und in Zusammenwirken mit eigenen Policies (Regelwerk) eine entsprechende Einstufung der Entität im eigenen Wirkungsbereich vornimmt.
Innerhalb einer SOA, in der sich weitreichende B2B-Netzwerke entwickeln sollen, kommt der Assertion-Übermittlung ein hoher Stellenwert zu, da es hier sehr wichtig ist, für alle

[31] Entstanden aus dem Zusammenschluss von S2ML und AuthML.

Mitarbeiter aller beteiligten Unternehmen mit nur einer einzigen Authentikation Zugriff auf sämtliche für diese Entität freigegebenenen Services zu ermöglichen (*single sign-on*). Da sich dies auch auf Services anderer Organisationen innerhalb des entstandenen B2B-Netzwerks bezieht, entsteht dadurch ein gemeinsames *virtuelles System* mit vertrauensbasierter, verteilter Authentikation und Autorisation.
Zwei wichtige Architektur-Vorteile von SAML in diesem Anwendungsgebiet sind, dass ersten keine zusätzliche zentrale Verwaltungsstelle für Authentikation und Autorisation eingerichtet werden muss und dass zweitens keiner der am Netzwerk beteiligten Organisationen gezwungen ist, die Nutzer der anderen Organisationen komplett mitzuverwalten. Gegen diese beiden alternativen Lösungsmöglichkeiten sprächen ohnehin die Fragen nach der Finanzierung, dem steigenden adminstratorischen Aufwand für Datenabgleich und Verwaltung sowie das wichtige Problem der Geheimhaltung von Daten. Eine Bank z.B. wäre wohl nur widerwillig dazu bereit, die wichtigsten Grundlagen ihrer Existenz, die Daten ihrer Kunden, zur Verwaltung bzw. sogar zur kompletten Verwahrung an Dritte herauszugeben.

Für SAML gilt, wie auch für alle anderen Webservice-Protokoll-Standards, dass keine statische Bindung an ein konkretes Transportprotokoll vorgegeben ist. Im Kontext der Webservices wird dies allerdings in aller Regel SOAP (s.Kap. 3.2) sein.
Die SAML-Spezifikation definiert auch ein Client/Server-Protokoll, welches es ermöglicht nach dem Request/Response-Verfahren XML-Nachrichten untereinander auszutauschen. Aufgabe dieses Protokolls (standardmäßiges XML-Prefix: "samlp") ist die Anforderung benötigter Aussagen über konkrete Entitäten durch einen sogenannten PEP (Policy Enforcement Point) an einen PDP (Policy Decision Point) innerhalb einer zusammengehörigen Authentikations-/Autorisationseinheit bzw. durch einen Service an einen anderen (s.Abb. 10). Durch dieses Protokoll sind beliebig strukturierte und verteilte Geschäftsabläufe denkbar, in denen an jedem Authentikationspunkt keine direkte Re-Authentifizierung des Nutzers, sondern eine Anfrage an einen PDP bzw. einen externen Service durchgeführt wird

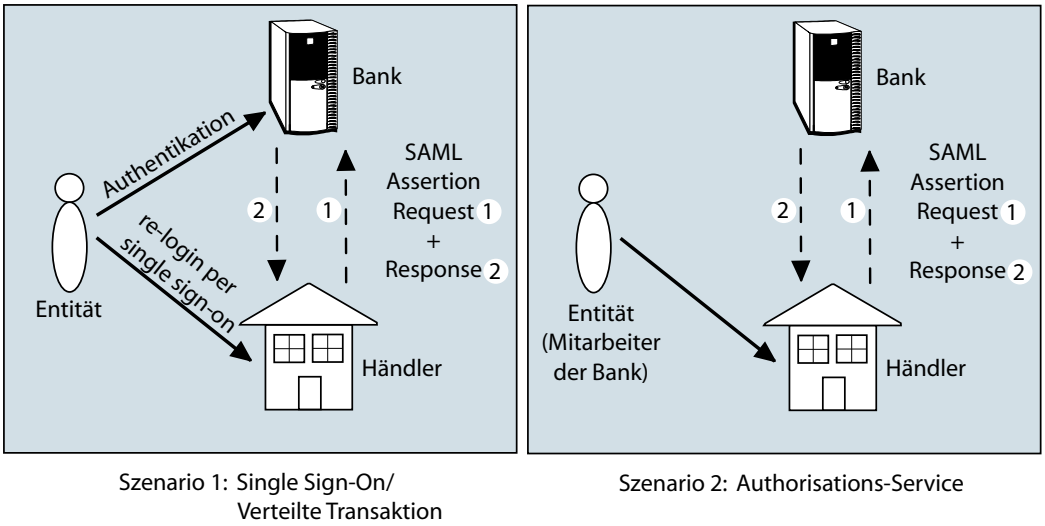


Abbildung 10: SAML Einsatz-Szenarios

(s.Abb. 10, Szenario 2).
Wie bereits Eingangs erwähnt sind Assertions nicht auf bloße Authentikation beschränkt. Vielmehr können diese verschiedenste Attribute und Autorisationsdaten enthalten. Da-

durch werden Anwendungsszenarien, wie z.B:

> Eine Bank authentifiziert eine Entität anhand ihrer präsentierten Berechtigungsnachweise (Login, Passwort etc.) und formuliert eine entsprechende Assertion. Weiterhin formuliert sie unter Verwendung der SAML-Attribut- und Autorisations-Fähigkeiten eine Assertion, welche beinhalted, dass dieser Nutzer kreditwürdig ist und über ein Limit von 2000,- Euro verfügt.
> Daraufhin möchte diese Person einen Einkauf bei einem Händler (mit entsprechendem Vertrauensverhältnis zur Bank) tätigen. Dieser fordert unter Verwendung des Request/Response-Protokolls die oben formulierten Assertions von der Bank an. Auf Grund dieser (unbedingt Integrität wahrenden) Informationen stellt er die Identität (bzw. Pseudonym) dieser Entität fest und weiß, dass sie auch in der Lage ist, bis zu einem Betrag von 2000,- Euro zu zahlen. Somit kann der Händler, basierend auf der Aussage der Bank, bei einer Bestellung entscheiden, ob er diese entgegen nimmt oder ob er sie ablehnt (s.Abb. 10, Szenario 1).

ermöglicht.

Eine Menge von Assertions wird auch als "Profil" bezeichnet und kann aus Assertions unterschiedlicher Organisationen bestehen. Den einzelnen Inhalten eines solchen Profils kann entsprechend nur soviel Vertrauen entgegen gebracht werden, wie man dem ausstellenden System vertraut. An dieser Stelle wird auch die Wichtigkeit der Kryptografie für SAML offensichtlich: bei einer Übertragung darf eine Assertion auf keinem Fall modifiziert werden (Integrität) und muss unter Umständen auch geheim gehalten werden (Vertraulichkeit). Hierbei spielen XML-Encryption und XML-Signature eine wichtige Rolle.
Folgende Assertion-Typen existieren in SAML:

Authentication Assertion: Eine Authentikationsautorität verarbeitet vorgelegte Berechtigungsnachweise einer Entität in Zusammenarbeit mit ihrem zugrunde liegenden Regelwerk (Policies). Bei einer positiven Bewertung dieser Nachweise können entsprechende *Authentication Assertions* in Bezug auf die Identität der Entität und des Assertion-Ausstellers sowie über Attribute, wie Zeitpunkt bzw. Art der Identitätsfeststellung, Zeitspanne, für welche die Authentifikation maximal gültig ist etc. formuliert werden. Durch diese Assertions wird eindeutig zum Ausdruck gebracht, dass die Identität einer Entität von einem bestimmten System festgestellt wurde. Im selben Vertrauensnetzwerk befindliche Systeme können nun mit Hilfe der Abfrage dieser Assertions und in Zusammenarbeit mit ihrem eigenen zugrunde liegenden Regelwerk entsprechende eigene Einstufungen der Entität in Bezug auf Authentikation und Autorisation vornehmen.

Attribute Assertion: Erhält eine Attributautorität eine *Authentication Assertion* und stuft diese selbst als vertrauenswürdig ein, so kann diese z.B. anhand eines eigenen Regelwerks Privilegien der identifizierten Entität feststellen und mit Hilfe von *Attribute Assertions* in SAML formulieren. Diese können dann z.B. in einem PDP für die Authentikation eingesetzt werden.

Authorization Decision Assertion: Erhält ein PDP von einem PEP eine Anfrage, ob eine bestimmte Entität Zugriff auf eine Ressource erhalten darf oder nicht, so entscheidet die PDP anhand der vorliegenden *Authentication Assertions* und *Attribute Assertions* und formuliert ihr Ergebnis mit Hilfe einer *Authorization Decision Assertion*. Das Ergebnis einer solchen Anfrage kann eine Freigabe (Permit), eine Ablehnung

(Deny) oder die Feststellung einer fehlenden Entscheidungsfähigkeit (Indeterminate) sein.
SAML ist neben diesen groben ja/nein/weiß nicht-Aussagen auch in der Lage feingranularere Aussagen über Zugriffsrechte zu formulieren. Dies hat den entscheidenden Vorteil, dass Anwendungsprogrammierer nicht dazu gezwungen sind, eine spezielle Umgebung für die Zugriffskontrolle zu entwickeln. Zu diesen Aussagen gehören z.B. Read, Write, Execute, Delete, Control (Entität besitzt erweiterte bzw. administrative Rechte) allgemein, Get, Head, Put, Post (lesen bzw. schreiben vom/auf Server) im HTTP-Umfeld oder Unix-Zugriffsrechte im "rwx"-Stil.

6.4 SOAP-Nachrichten sichern - *WS-Security*

Die in Kapitel 6.1 und 6.2 vorgestellten Kryptografie-Standards XML-Signature und XML-Encryption machen es möglich, beliebige Teile eines XML-Dokuments gegen Veränderung bzw. Abhören zu schützen und mit Hilfe von SAML (s.Kap 6.3) können so genannte *Sicherheitstokens* in XML-Notation dargestellt werden.
Da Webservice-Nachrichten unter Verwendung von SOAP übermittelt werden und dieses eine fest vorgegebene Struktur besitzt kann nicht einfach an beliebigen Stellen signiert, verschlüsselt oder eine sicherheitskritische Information eingefügt werden. Dies könnte die Struktur einer SOAP-Nachricht derart verzerren, dass kein Intermediary mehr dazu in der Lage wäre, die ankommenden Bytes als standardisierte SOAP-Nachricht zu identifizieren oder weiterzuverarbeiten.
Aus diesem Grund begann Microsoft im Oktober 2001 an der Arbeit einer entsprechenden Spezifikation, welche dann im Juni 2002 in Zusammenarbeit mit IBM und Verisign als WS-Security-Vorschlag an OASIS übergeben wurde, um einen Standard zu formulieren. WS-Security stellte den ersten Schritt in Richtung von abgesicherten Geschäftsbeziehung im Webservice-Kontext dar und bildete die Grundlage für spätere Spezifikationen, wie WS-Trust, WS-Policy (s.Kap. 6.9) oder WS-SecureConversation (s.Kap. 6.7).
WS-Security beschreibt die Vorgehensweise, wie Sicherheitstokens in eine SOAP-Nachricht eingebettet werden sollen und wie diese oder auch beliebige Teile der SOAP-Nachricht anschließend signiert bzw. verschlüsselt werden. Tokens werden dabei in so genannte Claims (dt.: Recht oder Anspruch) unterteilt, welche bestimmte Privilegien einer Identität widerspiegeln. Es werden durch WS-Security auch Szenarios betrachtet, in denen Nachrichtensicherheit beim Passieren von Intermediaries erhalten werden muss oder diese selbst zusätzlich Sicherheitsfunktionen auf die passierenden Nachrichten anwenden müssen.
Sämtliche WS-Security-kompatiblen Sicherheitsinformationen innerhalb einer SOAP-Nachricht sind in einzelne Blöcke unterteilt und werden an entsprechend vorgesehenen Stellen in den SOAP-Header eingefügt. Dabei werden die unterschiedlichsten Sicherheitsmodelle (Nutzername+Passwort, Zertifikate etc.), Sicherheitstechnologien (Kerberos, PKI, SAML etc.) und Sicherheitstokens (Kerberos-Ticket, X.509, SAML Assertions etc.) sowie Verschlüsselung und Signaturen allgemein unterstützt. Daran lässt sich erkennen, dass WS-Security auf der einen Seite so flexibel ist, sich nicht auf einen speziellen Mechanismus zu versteifen, auf der anderen Seite sich aber fest auf die Adressierung von sicherheitsrelevanten Problemen beschränkt.
Abbildung 11 stellt eine beispielhafte SOAP-Nachricht dar, in welcher ein Sicherheitstoken, ein verschlüsselter symmetrischer Schlüssel und eine Signatur under Verwendung der WS-Security Spezifikation eingebettet wurden. Das Element *wsse:Security* ist das umfassende Element aller WS-Security konformen Einbettung in den Kopf einer SOAP-Nachricht (das XML-Prefix "wsse" wird standardmäßig für WS-Security verwendet).

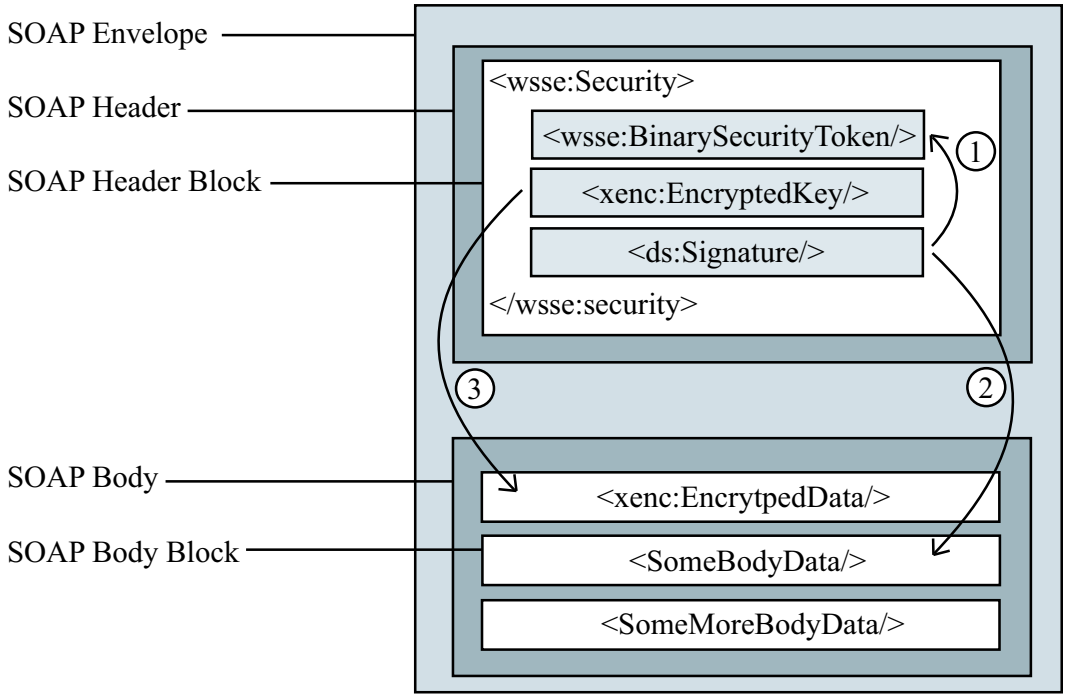


Abbildung 11: Einbettung von Sicherheitsinformation mit Hilfe von WS-Security in eine SOAP-Nachricht

Die Referenz (1) hält fest, dass die Signatur über das *SomeBodyData*-Element (2) mit Hilfe des im *wsse:BinarySecurityToken*-Element enthaltenen binären Tokens überprüft werden kann. Der *BinarySecurityToken* ist ein einfach strukturiertes Element, durch welches es ermöglicht wird, alle Arten von Tokens, welche sich nicht direkt als XML-Daten darstellen lassen (z.B. X.509 Zertifikate, Kerberos Tickets etc.) an eine SOAP-Nachricht zu binden. Er enthält die binären Daten für den Token, eine Definition des Token-Typs (z.B. wsse:X509v3) im *ValueType*-Attribut, die Codierung für die Darstellung der binären Daten im XML-Dokument (z.B. wsse:Base64Binary) im *EncodingType*-Attribut und eine eindeutige ID für die Referenzierung durch andere Strukturen im *Id*-Attribut.
Neben dem *BinarySecurityToken* existiert noch die Möglichkeit des *UsernameToken*. Dabei wird der Nutzername und ein Passwort bzw. dessen Hash an eine SOAP-Nachricht gebunden. Zusätzlich beinhaltet er weitere wichtige Elemente wie das *Nonce*-Element zur Verhinderung von Replay-Angriffen (s.Kap. 5.2.2) oder Informationen, wann der Token angelegt wurde etc.. Dieser Art der Tokenübermittlung erfordert in den meisten Fällen weitere Schutzmaßnahmen, wie XML-Encryption oder/und XML-Signature.

Eine weitaus bessere Möglichkeit als die Übermittlung eines *UsernameToken* ist die Einbettung einer *SAML Authentication Assertion*. Obwohl dies im Beispiel nicht verwendet wird, ist dies durch Einbettung des *saml:Assertion*-Elements in das *wsse:Security*-Element möglich. Hierbei muss allerdings auch darauf geachtet werden, dass enthaltenen Informationen bei der Übermittlung auf Integrität und/oder Vertraulichkeit hin geschützt werden. Die im Beispiel dargestellt Referenz (3) verweist von einem im WS-Security-Block enthaltenen symmetrischen Schlüssel (per *Id*-Attribut) auf einen Datenblock, welcher mit diesem verschlüsselt wurde. Die Information, wie der verschlüsselte symmetrische *EncryptedKey* zu entschlüsseln ist, wird im enthaltenen *KeyInfo*-Element festgehalten.

Treten bei einer WS-Security-Operation Fehler auf, so sieht die WS-Security Spezifikation dafür standardisierte Rückgabeinformationen als SOAP-Fehler vor. Dies sind unter anderem folgende Fehler-Codes: *wsse:UnsupportedSecurityToken*, *wsse:UnsupportedAlgorithm*, *wsse:InvalidSecurity* etc..

6.5 Zugriffsregeln definieren - *XACML*

Die XACML[32] (eXtensible Access Control Markup Language) ist ebenfalls ein OASIS-Standard (seit Februar 2005 in der Version 2.0 verfügbar), welcher es ermöglicht, in einer standardisierten Art und Weise Zugriffsregeln im XML-Format zu definieren. Dazu gehören neben der Festlegung für ein Regelformat auch Vorgaben für deren Kombination und wie diese abgearbeitet werden.
Das zugrundeliegende fest definierte XML-Vokabular ermöglicht außerdem einen Austausch der entstandenen Regeln zwischen Enterprise-Systemen, auch wenn sie vor einer lokalen Verwendung erst in ein anderes Format überführt werden müssen. Dadurch werden Regeln nicht mehr für jede neue Anwendung lokal erneut definiert, sondern können, einmal definiert, verbreitete Verwendung finden (z.B. Dokumente, Datenbanken, Applikationsserver etc.).
XACML-Zugriffsregeln stellen die Grundlage für komplexe Szenarios dar, in welchen über Zugriffe durch Entitäten auf Ressourcen entschieden wird. Diese Entscheidungen können dabei basierend auf der Charakteristik des Anfragenden ("auf Dokument X haben nur Nutzer der Gruppe Y lesenden Zugriff"), des verwendeten Anfrageprotokolls ("es muss SSL verwendet werden, um auf Dokument X lesend zugreifen zu können"), des Authentikationskontextes ("Authentikation für das Dokument X nur nur mit Hilfe eines digitalen Zertifikats") oder einer beliebigen Kombination daraus gefällt werden.
Die in Kapitel 6.3 vorgestellte Spezifikation "SAML", welche sehr eng mit XACML verknüpft ist, ermöglicht die Übertragung von Tokens, welche Authentikations- und Autorisationsinformationen sowie verschiedene Attribute enthalten. Die Regeln für eine Entscheidung (am PDP), ob die Autorisation für eine bestimmte Aktion erteilt wird oder nicht, können mit Hilfe von XACML dargestellt werden. Die verfügbaren SAML-Daten werden dazu mit Hilfe von XSL [XSL] und entsprechenden per Spezifikation vordefinierten Transformationsregeln an die entsprechenden Stellen in einem XACML-Dokument zur Evaluation abgebildet.
Einige XACML-Regeln benötigen für eine korrekte Ausführung zusätzlich bestimmte kontextgebundene Attribute (z.B. die Rolle eines Nutzers oder die aktuelle Zeit). Die Fähigkeit einer Regel, diese Informationen anzufordern (z.B. in Form einer SAML Attribute Assertion) bezeichnet man als *Prädikat* einer Regel. Dieses muss für einen konkreten Einsatzfall erst errechnet und zugeteilt werden.
Die XML-Darstellung einer XACML-Regel besitzt folgende grobe Struktur:

```
<Rule RuleId="..."
      Effect="[Permit | Deny]"
      xmlns="..." xmlns:...="..." ...>
      <Description/>
      <Target/>
      <Condition>
</Rule>
```

[32] "XACML" ausgesprochen: "zac-mull"

Das *RuleId*-Attribut enthält eine eindeutige Kennzeichnung für diese Regel und mit Hilfe des *Effect*-Attributs wird angegeben, ob der gewünschte Zugriff bei einer erfolgreichen Anwendung dieser Regel gestattet ("Permit") oder verweigert ("Deny") werden soll. Die *xmlns*-Attribute definieren die verschiedenen im Dokument verwendeten Namensräume.
Die Unterelemente besitzen folgende Bedeutung:

Beschreibung (*description*): Eine kurze, für den Menschen lesbare textuelle Beschreibung des Sinn und Zweck dieser Regel.

Ziel (*target*): Definiert eine mögliche Aktion (auf Dokumenten, Diensten etc.), für welche eine anfragende Entität eine Berechtigung erlangen möchte. Anhand dieses Zieles wird eine passende Regel für eine konkret auszuführende Aktion ausgewählt und abgearbeitet.

Bedingungen (*conditions*): In vielen Fällen ist es notwendig eine Entscheidung nicht nur auf Grund einer starren Festlegung, sondern mit einer bestimmten Dynamik zu treffen. Diese Dynamik kann dabei durch Attribute der teilnehmenden Parteien (Aufrufender, Ressource etc.), der gewünschten Operation oder durch bestimmte Umgebungsbedingungen (z.B. Uhrzeit) gesteuert werden. Zur Auswertung dieser Bedingungen kann auf Standardoperationen, wie Gleichheit bzw. Ungleichheit oder auf eine Vielzahl erweiterter Funktionen (Addition, Subtraktion, Rundung, Datumsvergleiche, Größer/Kleiner als etc.) zurückgegriffen werden.

Anhand dieses ausdrucksstarken Regelwerks können selbst für komplexe Szenarios Aussagen über Rechtmäßigkeit/Unrechtmäßigkeit eines Zugriffs getroffen werden.
Um das Verständnis für die Umsetzung der errechneten Zugriffsrechte am PEP zu erhöhen sollen hier die wohl bekanntesten Konzepte für das Zugriffsmanagement kurz vorgestellt werden:

Access Control List (ACL): Bei dieser Methode der Zugriffskontrolle werden dem Nutzer unter Zuhilfenahme einer *Zugriffsmatrix* bestimmte Rechte zugewiesen. Anhand dieser wird dann geprüft, ob die gewünschte Zugriffsart auf eine Ressource gestattet wird oder nicht. Beispiele dafür sind die Dateisysteme der Betriebssysteme MS Windows (NTFS) und Unix.

Role-Based Access Control (RBAC): Diese Art der Zugriffskontrolle ermöglicht eine Mitbetrachtung des Nutzerkontextes. Ein Kontext beinhalted z.B. die Mitgliedschaft in einer bestimmten Gruppe oder das Ausfüllen einer bestimmten Rolle (z.B. Manager, Programmierer etc.). Eine Gruppen- und Rolleneinteilung erlaubt es, die Mitgliederstruktur einer Organisation widerzuspiegeln.
Da in diesem Fall eine Rolle[33] bestimmte Rechte besitzt und nicht der Nutzer (mehrere Nutzer können die selbe Rolle ausüben) und da Rollen auseinander abgeleitet werden können, ergibt sich für RBAC eine geringere Komplexität als für das Konzept der ACLs.
Die Zahl der zu verwaltenden Regeln wächst bei RBAC proportional zur Zahl der Rollen/Gruppen und nicht zur Zahl der registrierten Nutzer. Es wird daher angenommen, dass RBAC besser skaliert als ACL [On02].

Das Zusammenfügen mehrerer Regeln zu einer so genannten "Policy" stellt einen der wichtigsten Aspekte der XACML-Spezifikation dar. Dadurch wird es ermöglicht, ein für einen

[33] Eine Nutzer muss nicht statisch an eine Rolle gebunden sein. Es kann durch verschiedene Umgebungseinflüsse, wie z.B. die Uhrzeit, ein Veränderung der Rollenzugehörigkeit hervorgerufen werden.

Bereich oder eine gesamte Organisation gültiges Regelwerk für den Ressourcenzugriff zu formulieren. Die XML-Darstellung einer solchen Policy besitzt folgende grobe Struktur:

```
<Policy PolicyId="..."
        RuleCombiningAlgId="..."
        xmlns="..." xmlns:...="..." ...>
    <Description/>
    <Target/>
    (<Rule/>)*
    <Obligations/>
</Policy>
```

Das *PolicyId*-Attribut enthält eine eindeutige Identifikation für diese Policy und mit dem *RuleCombiningAlgId*-Attribut wird standardisiert festgelegt, wie die errechneten Effekte bei einer Abarbeitung mehrerer Regeln miteinander kombiniert werden (z.B. einmal "Deny" überschreibt alle möglichen "Permit"-Entscheidungen). Das *Description*-Element enthält einen durch Menschen lesbare Beschreibung des Policy-Zwecks. Unter Verwendung des *Target*-Elements wird es analog zu den *Rule*-Elementen ermöglicht, zu einer konkreten Anfrage die anzuwendenden Policies herauszusuchen. Wenn alle enthaltenen *Rule*-Elemente allerdings eigene Zielfestlegungen über ihre *Target*-Elemente getroffen haben, ist es nicht notwendig, zusätzliche Zielfestlegungen auf *Policy*-Ebene zu definieren. Das Ziel der Policy wird dann nach bestimmten Kombinationsregeln aus den Zielen der enthaltenen Regeln errechnet (Vereinigungs- bzw. Schnittmenge der Definitionen für die Ressource, die auszuführende Aktion und die aufrufende Entität).
Die enthaltenen *Rule*-Elemente spezifizieren entweder die Regeldefinitionen selbst oder Referenzen auf die anderswo definierten zu verwenden Regeln.
Das *Obligations*-Element enthält eine beliebige Anzahl von *Obligation*-Elementen, welche bestimmte einzuhaltende Verpflichtungen definieren, denen bei einer positiven Zugriffsentscheidung nachzukommen ist. Dies können z.B. Benachrichtigungen per Email, SMS etc. an bestimmte Personen/Systeme oder auch die Eintragung in ein Zugriffsregister sein.
Mehrere Policies können hierarchisch in einer *Metapolicy* zusammengefasst werden. Bei Kombinationen von Ressourcen, Aktionen und Aufrufenden, welche auf mehrere Policies abgebildet werden können, wird so, ähnlich wie bei der Konfliktbereinigung von unterschiedlichen Regelentscheidungen innerhalb einer Policy, das Vorgehen in Konfliktsituationen definiert (z.B. "Deny" überschreibt alles).

6.6 Kommunikationsbrücke für PKI - *XKMS*

XKMS (XML Key Management Specification) ist eine W3C-Spezifikation und befindet sich seit April 2004 als "Recommendation Candidate" in der Version 2.0. Mit Hilfe dieser Spezifikation wird ein auf XML-Nachrichten basierender Mechanismus definiert, der den Zugriff einer Anwendung auf einen PKI-Service wesentlich vereinfacht. Es wird eine Entkoppelung von den unterschiedlichen PKI-Implementationen und ihrem Nachrichtenprotokoll, sowie eine starke Reduzierung der klientenseitigen Anwendungskomplexität erreicht. In Klienten-Richtung wird eine standardisierte und in PKI-Richtung eine flexible und umfangreiche Kommunikationsschnittstelle (z.B. unabhängig vom verwendeten Zertifikat-Standard) geboten. Durch sein XML-basiertes Nachrichtenprotokoll fügt sich diese Spezifikation bestens in die Reihe existierender Webservice-Standards und -Protokolle ein.
Der ursprüngliche Gedanke und Antrieb hinter der Entwicklung von XKMS war es, Kleingeräten, wie z.B. Handhelds, Pocket PCs oder Handys, trotz ihrer begrenzten Ressourcen,

eine Zugriff auf PKI und somit asymmetrische Kryptografie zu ermöglichen. Selbst eine rudimentäre Implementation der PKIX-Spezifikation[34] benötigt ein Vielfaches an Ressourcen der Implementation einer XKMS-Klientenschnittstelle [On02].
XKMS stellt eine Zwischenschicht für den Zugriff auf eine PKI dar und bildet im Webservice-Umfeld eine wichtige und kompatible Grundlage für die Verwendung von XML-Signature und XML-Encryption. Allerdings muss festgehalten werden, dass es *nicht* dafür gedacht ist, die eigentliche Komplexität von PKI zu reduzieren.
In der XKMS-Spezifikation werden zwei zentrale Webservices definiert:

XML Key Registration Service Specification (XKRSS): Definiert ein Lebenszyklus-Protokoll für die Registration (*register*), Ungültigerklärung (*revoke*), Wiederaktivierung (*reissue*) und die Wiederherstellung[35] (*recovery*) von öffentlichen Schlüsseln..

XML Key Information Service Specification (XKISS): Definiert ein Protokoll für das Auffinden (*locate*) und Überprüfen (*validate*) der Gültigkeit von öffentlichen Schlüsseln.

Ein XKMS-Service ist frei in der Entscheidung, ob er XKRSS, XKISS oder beides unterstützt.

6.7 Sichere Zwiesprache - *WS-SecureConversation*

Der WS-Security-Standard spezifiziert Sicherheitstokens, die in SOAP-Nachrichten eingebettet und mit einer entsprechenden Signatur bzw. Verschlüsselung versehen dazu verwendet werden, bestimmte in diesem Token enthaltene Ansprüche authentisch auszudrücken. Dazu wird der Sicherheitstoken jeder eingehenden SOAP-Nachricht gegen definierte Sicherheitsrichtlinien geprüft. Ein Problem dabei liegt darin, das dies für wirklich jede ankommende SOAP-Nachricht durchgeführt werden muss. Diese Überprüfung ist aufwendig, da immer wieder asymmetrische Signaturen geprüft und gegebenenfalls asymmetrische Chiffriertexte entschlüsselt werden müssen.
WS-SecureConversation führt das Prinzip der Sitzung für eine Gruppe von SOAP-Nachrichten ein. Dabei wird ein so genannter "authentifizierter Sicherheitskontext" erstellt, indem eine einzige SOAP-Nachricht zur Authentikation gegen die Sicherheitsrichtlinien verwendet wird. Im Rahmen dieses Sicherheitskontextes werden die Folgenachrichten auf der Basis von Sitzungsschlüsseln, abgeleiteten Schlüsseln (z.B. Pseudo-One-Time-Pad [Pf00]) oder ausgehandelten Schlüsseln für einzelne Nachrichten verschlüsselt. Bei jeder dieser Nachrichten ist auf Grund des umgebenden Sicherheitskontextes eine implizite Authentikation des Kommunikationspartners durchführbar.
Genau wie bei SSL werden auch bei WS-SecureConversation Folgenachrichten mit symmetrischen Algorithmen verschlüsselt, um deren Geschwindigkeitsvorteil bei der Berechnung gegenüber asymmetrischen Algorithmen ausnutzen zu können. Ein einmal (asymmetrisch) ausgetauschter symmetrischer Schlüssel kann für die Verschlüsselung mehrerer folgender Nachrichten verwendet werden.
Auf Grund der Ähnlichkeit zu SSL beim Protokollablauf wird WS-SecureConversation auch oft als "SSL auf SOAP-Level" bezeichnet.

[34] PKIX beschreibt, wie X.509-Zertifikate mit SSL, S/MIME oder IPSEC verwendet werden.

[35] Sehr umstrittene Funktion, welche die Kenntnis des privaten Schlüssels durch den PKI-Provider voraussetzt und von deren Verwendung in der Regel abzuraten ist. Als Alternative könnte z.B. das Hinterlegen durch ein Schwellwertschema [Pf00] erzeugter Teilgeheimnisse bei Bekannten dienen.

6.8 Zusammenspiel

Die soeben vorgestellten Technologien für die Sicherheit von Web Services können zwar alle prinzipiell völlig alleinstehend eingesetzt werden, jedoch ist dies nicht immer sinnvoll. So ist z.B. eine SAML-Assertion ohne einen Integritätsschutz völlig wertlos, da digitale Daten von jedem erzeugt und beliebig kopiert werden können.
In diesem Kapitel soll deshalb anhand eines selbst gewählten Beispiels ein Einsatzszenario vorgestellt werden, bei dem möglichst viele der vorgestellten Sicherheitstechnologien zusammenhängend angewendet werden, um eine bestimmte Aufgabe zu erfüllen.

In Abbildung 12 ist Tom ein Angestellter der "Wisch&Weg Reinigung" und führt in ih-

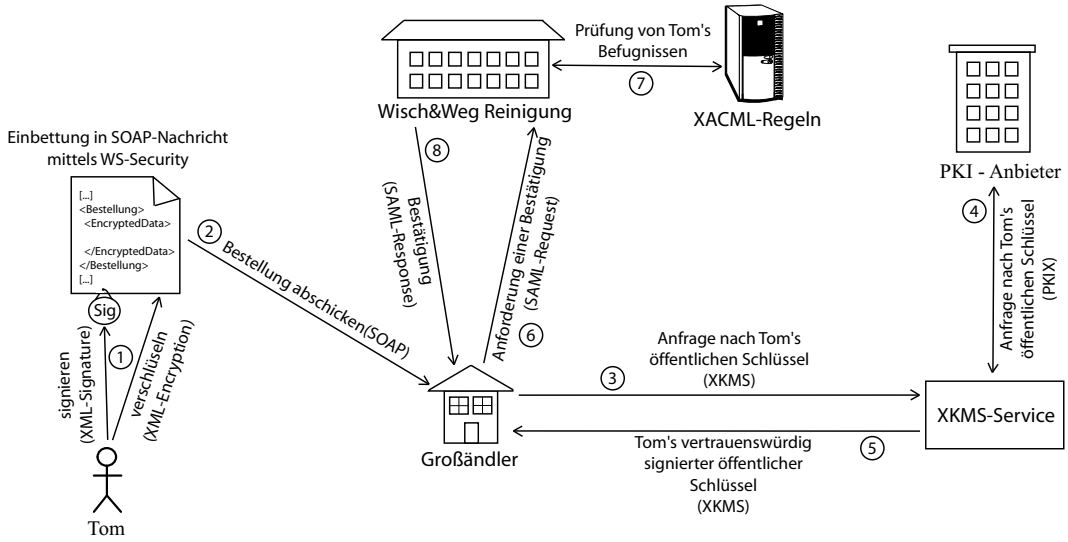


Abbildung 12: Beispiel für das Zusammenspiel der Webservice-Sicherheitsmechanismen

rem Auftrag Einkäufe bei einem Großhändler durch. Dazu formuliert er eine Bestellung und signiert diese mit seinem privaten Schlüssel unter Verwendung von *XML-Signature* (1). Weiterhin besteht auch die Möglichkeit, die Bestellung mit *XML-Encryption* zu verschlüsseln, wenn deren Inhalt vertraulich übertragen werden soll (1). Den notwendigen Schlüssel dazu bezieht er z.B. als Zertifikat von der Internetseite des Großhändlers (asymmetrisch) oder er hat bereits auf Grund mehrerer abzuwickelnder Bestellungen oder einer über mehrere SOAP-Nachrichten dauernden Bestellung mit dem Großhändler unter Verwendung von WS-SecureConversation einen symmetrischen Sitzungsschlüssel vereinbart.
Die Einbettung der Signatur bzw. des Schlüsseltextes in die SOAP-Nachricht erfolgt unter Verwendung von WS-Security.
Die so erstellte, auf Vertraulichkeit und Integrität hin geschützte SOAP-Nachricht übermittelt Tom in Schritt (2) an den Großhändler. Dieser fordert für die Prüfung der Signatur unter Verwendung eines XKMS-Clients Tom's öffentlichen Schlüssel bei einem XKMS-Service an (3). Dieser verwaltet Tom's öffentlichen Schlüssel nicht selbst und fordert diesen deshalb von einem PKI-Anbieter an (4) und sendet ihn zurück zum Großhändler (5).
Der Großhändler "weiß" nun nach positiver Prüfung der Signatur, dass tatsächlich Tom die Anfrage gestellt hat und vorgibt dies im Auftrag der Wisch&Weg Reinigungsgesellschaft zu tun. Die Bestellung wird der Großhändler allerdings noch nicht auslösen, da er nicht weiß, ob Tom tatsächlich befugt ist, für die Reinigungsgesellschaft Einkäufe mit dem

gewünschten Gesamtpreis zu tätigen.
Der Großhändler formuliert eine SAML-Anfrage mit dem Inhalt:
"Ist die authentifizierte Entität Tom [... genaue Angaben ...] befugt im Auftrag der Wisch&-Weg Reinigung einen Auftrag in Höhe von XXX,XX Euro auszulösen?"
und schickt diese als SAML-Request (6) an die Reinigungsgesellschaft. Die Reinigungsgesellschaft bearbeitet diese Anfrage, indem sie einen internen Server, welcher sämtliche Befugnisse von Mitarbeitern in Form von XACML-Regeln verwaltet, "befragt" (7).
Je nachdem, ob die Überprüfung vom Tom's Befugnissen positiv oder negativ ausfällt, formuliert die Reinigungsgesellschaft in Form von SAML-Assertions einen SAML-Response und sendet diesen an den Großhändler zurück (7). Dieser entscheidet nun anhand dieser Antwort, ob er die Bestellung entsprechend annimmt oder ablehnt.

6.9 Weitere Technologien

Wie auch bei den Basistechnologien gibt es bei den Webservice-Sicherheitstechnologien eine Vielzahl von weiteren Technologien, welche in dieser Arbeit zwar nicht im Detail vorgestellt aber zumindest mit einer kurzen Beschreibung erwähnt werden sollen. Auch bei dieser Auflistung wird kein Anspruch auf Vollständigkeit erhoben.

WS-Trust: definiert ein erweiterbares Modell für den Aufbau von Vertrauensbeziehungen. Diese Beziehungen können entweder direkt oder über einen so genannten "Vertrauensproxy" aufgebaut werden.
Da der direkte Aufbau einer Vertrauensbeziehung bei global operierenden Dienstleistern nur in den seltensten Fällen möglich ist, liegt das Schlüsselkonzept von WS-Trust in den so genannten Sicherheitstoken-Services (STS).
Ein STS ist ein Dienst, welcher Sicherheitstokens ausstellt, austauscht und überprüft. WS-Trust ermöglicht es Webservices entsprechende Aussagen zu treffen, welchen STS sie vertrauen. Das dem STS entgegengebrachte Vertrauen wird dann auch "seinen" Sicherheitstokens und seinen Beurteilungen entgegengebracht.
Der Versand der Tokens (vom STS selbst erstellt oder durch ihn vertrauenswürdig anderweitig bezogen) erfolgt dann wieder unter Verwendung der bereits vorgestellten WS-Security-Mechanismen.

WS-Federation: beschreibt, wie Vertrauensbündnisse unter Verwendung von WS-Security, WS-Policy, WS-Trust und WS-SecureConversation zwischen kooperierenden Parteien aufgebaut werden können. Dabei wird ein besonderes Augenmerk auf die "Übersetzung" zwischen verschiedenen Sicherheitsspezifikationen gelegt. Z.B. könnte die eine Partei Kerberos anwenden, wohingegen die andere Partei X.509 Zertifikate verwendet. Die Aufgabe von WS-Federation würde hier darin bestehen, eine Nutzer, welcher bei der einen Partei eingeloggt ist den Zugang zu den Web Services der anderen Partei zu ermöglichen.
WS-Federation ist somit als eine Art "Beziehungsmanager" oberhalb von WS-Trust und WS-Policy, welche dafür verantwortlich sind, welche Tokens verwendet und wie sie bezogen werden können, angesiedelt.

WS-SecurityPolicy: aufbauend auf der WS-Security Spezifikation wird festgelegt, wie Richtlinien (policies) bei der Verwendung von WS-Security ausgedrückt werden. Es wird vom Konsortium (IBM, Microsoft, RSA Security Inc. und VeriSign Inc.) selbst als Nachtrag für die WS-Security Spezifikation gesehen.

WS-Authorization: ist eine Spezifikation mit sehr großer Ähnlichkeit zu XACML (s.Kap. 6.5). Sie beschreibt, wie Zugriffsregeln für einen Web Service angegeben und verwaltet werden.
WS-Authorization ist sehr flexibel in Bezug auf das Autorisationsformat und die jeweilige zu verwendende Autorisationssprache. Sie unterstützt sowohl ACL- als auch RBAC-basierte Autorisationsmechanismen.

WS-Privacy: beschreibt, wie sicherheitsrelevante Anforderungen an die kommunizierten Informationen formuliert werden können. Entsprechende Festlegungen können in eine WS-Policy-Beschreibung (s.Kap. 3.5) integriert werden.
Um das Ziel von WS-Privacy umsetzen zu können, wird zusätzlich zu WS-Policy eine Kombination aus WS-Security und WS-Trust verwendet. WS-Security stellt sicher, dass übermittelte Sicherheitstokens, welche Informationen enthalten, die durch eine entsprechende Privacy-Regel vorgeschriebenermaßen enthalten sein müssen, überprüft werden können (im Sinne von Integrität und evtl. Vertraulichkeit), wohingegen WS-Trust die enthaltenen Informationen (claims) gegen nutzerspezifische Vorlieben und durch die betreibende Organisation formulierte Regelwerke auswertet.

P3P: (Platform for Privacy Preferences) beschreibt mit Hilfe serviceseitiger PrivacyPolicies und clientseitiger Preferences Festlegungen und Wünsche im Umgang mit privaten Daten. Dazu gibt der Service an, ob und warum personenbezogene Daten erhoben und gespeichert werden, welche Zugiffsmöglichkeiten existieren, wo Streitfälle geschlichtet werden u.s.w.. Diese Informationen können dann vor einer Verwendung eines Services mit den Einstellungen des Nutzers verglichen werden.

7 Prototyp - Authentifizierender Webservice

Obwohl der Fortschritt von UDDI Version 2 zur Version 3 eine Vielzahl von Sicherheitsproblemen eliminiert hat (s.Kap. 3.7.1.7), besteht immer noch ein Gefahrenpotential, welches es zu beseitigen gilt:
Erhält der Klient von einem UDDI-Register einen signierten und als vertrauenswürdig eingestuften Eintrag als Ergebnis einer Anfrage, so kann er bei einem Aufbau der Verbindung zu diesem Service immer noch (z.B. durch einen DNS-Angriff; Kap. 5.2.3), ohne das er etwas davon bemerkt, an einen "falschen" Webservice geraten.
Um dies zu verhindern benötigt der Klient Informationen, die es ihm ermöglichen, den Webservice[36] eindeutig als den gewünschten zu identifizieren[37].
Mit Hilfe der Kryptografie (s.Kap. 2.3) werden Mechanismen zur Verfügung gestellt, mit deren Hilfe diese Identifizierung vorgenommen werden kann. Dabei gilt es zu unterscheiden, ob nur eine einzelnen Nachricht (mit oder ohne Antwort) zum Server geschickt, oder ob ein sicherer Kommunikationskanal (s.Kap. 2.4) zum Webservice aufgebaut werden soll. Weiterhin gilt es zu betrachten, ob der *Inhalt* der Nachricht zu schützen ist, oder nicht.

Eine Nachricht zum Service ohne Bestätigung :

Sinn dieser Art der Kommunikation, im Kontext der authentifizierenden Webservices, kann es nur sein, dass die Nachricht einzig und allein vom Zielservice entschlüsselt werden kann. Eine Verschlüsselung der Nachricht mit dem öffentlichen Schlüssel des Zielservices (asymmetrisches Verschlüsselungssystem; s.Kap. 2.3.2) stellt hierfür die beste Möglichkeit dar.
Ein Versenden der ungeschützten Nachricht käme dem Versenden einer Nachricht, ohne etwas über den Zielservice zu wissen, gleich, da in diesem Modus keine Antwort vom Service zurückgeschickt wird, welche den Erhalt durch den richtigen Service bestätigen könnte. Die Nachricht kann also verloren gehen (ein nachrichtenorientiertes Kommunikationssystem toleriert dies) oder/und von einem beliebigen falschen Service empfangen werden.

Eine Nachricht zum Server mit Bestätigung :

Bei dieser Kommunikation wird ebenfalls eine einzelne Nachricht an den Service geschickt. Die Bestätigungsnachricht durch den Service an den Klienten soll es diesem ermöglichen, Gewissheit darüber zu erlangen, dass seine Anfrage vom richtigen Service bearbeitet wurde. Hierbei gilt es die oben genannten Fallunterscheidung zu führen, ob der Nachrichteninhalt geschützt werden soll.
Ist dies der Fall, so kommt wieder nur eine Verschlüsselung der gesamten Nachricht (oder auch eines Teiles davon) mit Hilfe des öffentlichen Schlüssel des Zielservices in Frage. Als Beweis, dass der Zielservice über den zugehörigen privaten Schlüssel verfügt und somit die Nachricht auch tatsächlich vom gewünschte Kommunikationspartner erhalten wurde, fügt dieser in die Antwort eine in der Anfrage enthaltene, nicht zu erratende Information (z.Bsp. eine Zufallszahl [Pf00]) ein und verschlüsselt/signiert diese mit einem ausgetauschten Schlüssel oder, falls es das verwendet asymmetrische

[36] Denkbar wären auch Gruppen von Webservices, welche die gleiche Identifikationsinformation verwenden. Dies verringert jedoch das Vertrauen in die Sicherheit und bietet eine größere Fläche für potentielle Angriffe.

[37] Es soll hier davon ausgegangen werden, dass es den beteiligten Parteien nicht möglich ist, einen Austausch von Geheimnissen außerhalb des Kommunikationsnetzes durchzuführen.

System unterstützt, mit seinem zum öffentlichen Schlüssel des Klienten zugehörigen privaten Schlüssel. Ob der Inhalt der Antwort/Bestätigung zu schützen ist oder nicht, hängt vom jeweiligen Anwendungsfall ab und entscheidet über ihren Aufbau.
Muss der Nachrichteninhalt der Anfrage nicht geschützt werden, so ist es ausreichen eine geheime Information mit dem öffentlichen Schlüssel des Zielservice zu verschlüsseln und in der Anfrage mitzusenden.
Der Aufbau der Antwort/Bestätigung des Services entspricht dem des vorangegangen Falls.

Aufbau eines sicheren Kommunikationskanals :

Soll zwischen dem Klienten und dem Service ein sicherer Kanal aufgebaut werden, so sendet der Klient eine Kanalaufbaunachricht (z.Bsp. einen symmetrischen Schlüssel, verschlüsselt mit dem öffentlichen Schlüssel des Zielservices, für die performante Ver- und Entschlüsselungen der folgenden zu übermittelnden Daten) an den Service und initiiert somit den Kanal für den sicheren Informationsaustausch.

Alle diese Varianten habe eine Gemeinsamkeit: es wird der öffentliche Schlüssel des Zielservice benötigt, um sicher zu stellen, dass mit dem richtigen Service kommuniziert wird. Es muss also eine Möglichkeit geschaffen werden, diesen Schlüssel vertrauenswürdig zu beziehen.
Eine Möglichkeit wäre es, anhand des Suchergebnisses aus dem UDDI-Register (mit Hilfe der Signatur des Veröffentlichers als vertrauenswürdig eingestuft) ein zugehöriges vertrauenswürdig signiertes (z.Bsp. durch den Veröffentlicher oder den Betreiber einer vertrauenswürdigen PKI; s.Kap. 2.8) Zertifikat (s.Kap. 2.7) mit dem öffentlichen Schlüssel des Zielservices zu beziehen. Dabei spielt es keine Rolle, ob ein öffentliches Verzeichnis befragt oder das Zertifikat von einer anderen Stelle bezogen wird, vorausgesetzt die Signatur einer vertrauenswürdigen Instanz beweist dessen Authentizität.
Nachteile dieser Herangehensweisen wären auf der einen Seite unter Umständen eine steigende Abhängigkeit (finanziell, organisatorisch, Vertrauensverhältnis etc.) von PKI-Anbietern und auf der anderen Seite die Tatsache, dass eine zusätzliche Stelle zur Verfügung gestellt werden müsste, von der die zugehörigen Zertifikate bezogen werden können. Für Anbieter mit einer Vielzahl von öffentlichen Webservices und einer großen Nutzergruppe wäre dieser Ort zudem stark frequentiert und würde dadurch Ressourcen (Bandbreite, Rechenleistung, Speicherplatz) konsumieren.
Die folgende Beschreibung des im Rahmen dieser Arbeit erstellten Lösungsansatzes geht einen etwas anderen Weg.

7.1 Theoretische Umsetzung

Die Spezifikation eines UDDIv3-Registers [OA04] stellt eine flexible Datenstruktur mit einer Vielzahl von Erweiterungsmöglichkeiten (z.Bsp. die Datenstruktur *tModel*; s.Kap. 3.7.1.4) zur Verfügung.
Wenn eine Suchanfrage für einen bestimmten Service sowieso an ein zentrales UDDI-Register gestellt wird, was liegt da näher, als die benötigten Informationen über den Zielservice (z.Bsp. ein Zertifikat) ebenfalls dort abzuspeichern?

Der eine oder andere Leser wird sich jetzt sicher fragen: Wo liegt der Vorteil gegenüber der Abspeicherung eines Zertifikats in einem zentralen Register oder dem Bezug vom Zielservice selbst?

Da man ja sowieso mit dem UDDI-Register kommuniziert, um den gewünschten Service zu finden, kann man sich den für den Bezug des Zertifikats aus einem öffentlichen Register nötigen Kommunikationsschritt sparen, indem man das Zertifikat gleich mit aus dem Register bezieht.
Kommuniziert man mit dem Zielservice unverschlüsselt so kann man das Zertifikat zur Prüfung der Signatur in der Antwortnachricht vom Service als *BinarySecurityToken* (s.Kap. 6.4) in diese einbetten und weiß somit, dass die Anfrage vom richtigen Webservice bearbeitet worden ist.
Will man allerdings Daten an den Dienst verschlüsselt übermitteln, so wäre eine zusätzliche Kommunikation nötig um dessen Zertifikat und somit dessen öffentlichen Schlüssel eines asymmetrischen Konzelationssystems zu beziehen.
Durch den Erhalt des Zertifikats aus dem UDDI-Register wird der nötige Kommunikationsaufwand, um sicherzustellen, dass man mit dem richtigen Webservice kommuniziert, auf den ohnehin nötigen Kommunikationschritt für den Bezug der Information des Services aus dem UDDI-Register beschränkt.

7.1.1 Identifikation einer Speicherposition für Zertifikate im UDDI-Register

Wie in Kapitel 3.7.1 und den folgenden Unterkapiteln beschrieben, entspricht in der UDDI-Terminologie ein *businessService* einem logischen Service und beinhaltet die technischen Details für konkret bereitgestellte Dienste in den untergeordneten *bindingTemplate*-Elementen. Der richtige Platz, für das Ablegen des zugehörigen Zertifikats befindet sich auf dieser Ebene, da ein logischer Service eine Vielzahl unterschiedlicher Bindings enthalten kann.
Bei der Betrachtung dessen Struktur fallen zwei Konstrukte ins Auge, welche einen Erweiterungspunkt für zusätzliche Informationen darstellen könnten: *tModelInstanceDetails* und *categoryBag*.
Das Element *categoryBag* (s.Abb. 13) ermöglicht eine flexible Kategorisierung des jeweiligen umfassenden *bindingTemplate*-Elements und würde es mit Hilfe seiner *keyedReference*-Elemente (möglicherweise auch durch *keyedReferenceGroup*-Elemente gruppiert) ermöglichen, zusätzliche Eigenschaften zum *bindingTemplate* hinzuzufügen. Die Attribute des *keyedReference*-Elements würden dabei wie folgt verwendet: *tModelKey* würde auf einen Typ verweisen, welcher diese *keyedReference* z.B. als ein Zertifikat ausweist, *keyName* würde eine beschreibende Bezeichnung für diese *keyedReference* enthalten (z.B. "Zertifikatstruktur") und *keyValue* den Wert bzw. die Werte (Zertifizierungbaum) des/der Zertifkate enthalten.

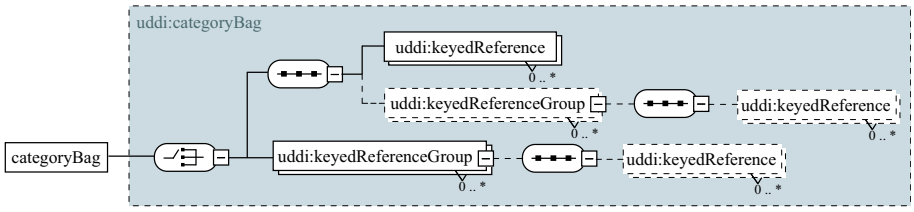


Abbildung 13: UDDI - *categoryBag*-Element

Obwohl das *categoryBag*-Element prinzipiell in dieser Art und Weise eingesetzt werden könnte, sprechen folgende Argumente dagegen:

- laut UDDI-Spezifikation wird der Verwendungszweck wie folgt beschrieben:

  > *"The optional categoryBag element allows businessEntity structures to be categorized according to published categorization systems. For example, a*

> *businessEntity might contain UNSPSC product and service categorizations [...]"*
>
> "Das optionale *categoryBag*-Element erlaubt es *businessEntity*-Strukturen anhand öffentlicher Kategorisierungssysteme einzuordnen. Z.B. könnte ein *businessEntity* eine Einordnung von Produkten und Dienstleistungen anhand der UNSPSC enthalten [...]"

Abstrahiert man diesen Sachverhalt auf *bindingTemplate*-Elemente muss man zu dem Schluss kommen, dass eine Verwendung als Ablagefläche für Zertifikate nicht im Sinne der UDDI-Spezifikation sein kann.

- die Anzahl der Zeichen im *keyValue*-Attribut ist per Spezifikation auf 255 Zeichen begrenzt und somit nur sehr bedingt für Zertifikate geeignet (Zertifikatgröße kann möglicherweise größer als 1kb (1024 Zeichen) sein).

Untersucht man die andere potentielle Zielstruktur, das *tModelInstanceDetails*-Element (s.Abb. 14), genauer, so setzt sich dies aus einer beliebigen Anzahl von *tModelInstanceIn-*

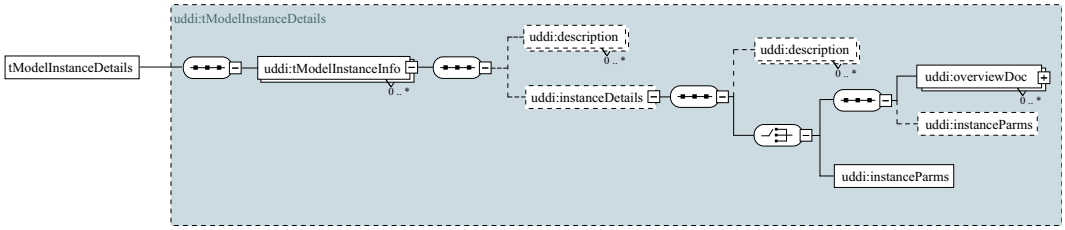


Abbildung 14: UDDI - *tModelInstanceDetails*-Element

fo-Elementen zusammen, welche über das notwendige Attribut *tModelKey* ein *tModel* referenzieren, mit dem das umgebende *bindingTemplate* dann als "kompatibel" (technischer Fingerabdruck!) gekennzeichnet wird (z.B "besitzt ein Zertifikat"). Ein *tModelInstanceInfo*-Element unterteilt sich wiederum in die Unterelemente *description* und *instanceDetails*. Jedes (optionale) *description*-Element beschreibt dabei in durch Menschen verständlicher Art und Weise, welche Rolle dieses *tModelInstanceInfo*-Element innerhalb der Service-Beschreibung spielt. Das *instanceDetails*-Element enthält neben einer Beschreibung (z.B. Verwendungszweck) im *description*-Element die Möglichkeit, entfernte Dokumente, welche zusätzliche beschreibende Informationen über die Verwendung des *tModels* und seiner *instanceParms* enthalten, zu referenzieren.
Das *instanceParms*-Element ist für die Lösung dieser Teilaufgabe interessant, da es hier ermöglicht wird, 8192 Zeichen (byte) lange Parameter, Einstellungen oder Konfigurationsinformationen zu hinterlegen. In diesen Kontext würde die Abspeicherung eines oder durchaus auch mehrerer Zertifikate im UDDI, sowohl vom verfügbaren Platz als auch von der durch die Spezifikation vorgeschlagenen Nutzungsempfehlungen her betrachtet, sehr gut hineinpassen.
Sämtliche *bindingTemplate*-Strukturen, welche Zertifikatsdaten enthalten und somit in Bezug auf Sicherheitsanforderung als kompatibel einzustufen sind, können dann anhand ihrer *tModelInstanceDetails*-Elemente identifiziert werden.

7.1.2 Zertifizierungsstruktur

Nachdem im vorangegangen Abschnitt eine Position innerhalb der aktuellen UDDI-Spezifikation (UDDIv3), in welcher die Zertifikatsdaten potentiell abgespeichert werden können, ausfindig gemacht und für das weitere Vorgehen festgelegt wurde, geht es in diesem Abschnitt darum, genau zu definieren welche Daten wie abgelegt werden müssen.
Um es einem Nutzer zu ermöglichen, einen Webservice tatsächlich als den gewünschten zu identifizieren, benötigt er, wie Eingangs erwähnt, dessen authentisches Zertifikat für die Verschlüsselung von Daten in einer Art und Weise, dass nachweislich *nur* der Ziel-Web Service in der Lage ist, diese zu entschlüsseln. Weiterhin könnte ein Zertifikat für den Rückweg einen Signaturmechanismus unterstützen, welcher es dem Nutzer ermöglicht eindeutig zu erkennen, dass die Anfrage tatsächlich vom Ziel-Webservice bearbeitet worden ist.
Am einfachsten wäre es also, ein Zertifikat abzulegen, welches von einer dem Nutzer gegenüber als vertrauenswürdig eingestuften CA direkt signiert wurde und als Zertifikatsgegenstand (Subject) exakt die Position des zu verwendenden Service enthält. Da dies allerdings erstens kostenaufwendig ist (die CA lässt sich jede Signatur bezahlen) und zweitens sich der "Anschluss" (URL, Port und eventuell zusätzliche Kriterien) eines Webservice unter Umständen oft verändert (besonders in der Entwicklungs-, Test- und Einführungsphase), muss eine andere Lösung für dieses Problem gefunden werden.
Wie bereits im Kapitel 2.8 über PKI erwähnt, ist es prinzipiell möglich, mit jedem Zertifikat eine Zertifikatsanfrage (kann jeder erstellen) zu unterschreiben und somit ein Zertifikat zu erstellen[38]. Also könnte man als Alternative zur ersten Herangehensweise z.B. nur ein Zertifikat pro Veröffentlicher (z.B. "IBM, Abteilung Microcode") von einer aus Nutzersicht vertrauenswürdigen CA ausstellen lassen. Da man dem Veröffentlicher in Bezug auf seine angebotenen Webservices sowieso vertraut (sonst würde man sie ja nicht nutzen wollen), kann man natürlich auch allen von ihm signierten Zertifikate für *seine* Webservices vertrauen.
Der Veröffentlicher kann mit seinem Zertifikat (nur einmal bei der CA dafür bezahlt) nun unabhängig und beliebig viele weitere Zertifikate für seine Web Services erstellen. Dazu können neben Zertifikaten für Konzelation auch Zertifikate für Signaturen oder kombinierte Zertifikate (z.B. RSA) erstellt werden.
Alles, was ein Nutzer jetzt noch benötigt, um einen Webservice als authentisch zu identifizieren, ist das Zertifikat der CA, das des Veröffentlichers und das des Webservices. Das Zertifikat der CA kann man sich aus einem öffentlichen Verzeichnis oder von deren Internetseiten (z.B. VeriSign, Thawte etc.) herunterladen bzw. aus sogar aus seinem installierten Browser entnehmen. Wenn das Zertifikat des Veröffentlichers nicht schon vorhanden ist, wäre es sinnvoll dieses neben den Zertifikat des Webservices aus dem UDDI-Register zu beziehen.
Leider gibt es auf dem *bindingTemplate*-Element übergeordneten Strukturen *businessEntity* und *businessService* keine Möglichkeit das Zertifikat des Veröffentliches abzuspeichern. Zweckentfremdende Möglichkeiten innerhalb eines *categoryBag*- bzw. eines *identifierBag*-Elements (bestehend aus beliebig vielen *keyedReference*-Elementen) oder sogar innerhalb eines extra anzulegenden *tModel*-Elements für *jedes* Veröffentlicher-Zertifikat zu speichern, werden dabei nicht betrachtet, zumal die Nutzdaten auf Grund der 255 Zeichen Längenbeschränkung auch noch aufgeteilt werden müssten.
Also bleibt als einzige Alternative alle Zertifikate, sowohl das des Veröffentlichers als auch das/die des Webservices innerhalb des *instanceParams*-Elements abzulegen.
Da die für den Prototypen verwendeten Zertifikate (RSA mit einer Schlüssellänge von

[38] Mit der Vertrauens- und der Glaubwürdigkeit des Besitzers des ausstellenden Zertifikats.

1024bit) inklusive sämtlicher Informationen und einer Signatur (MD5 mit RSA) eine Größe von 1536byte nicht überschreiten, ist es potentiell möglich, bis zu fünf Zertifikate innerhalb eines *instanceParms*-Elements abzulegen.

7.1.3 Probleme beim Lebenszyklus eines Zertifikats

Betrachtet man den Lebenszyklus eines Zertifikats, so besteht dieser im Wesentlichen aus der Erzeugung, der Nutzung und dem Ablauf nach dem bei der Erstellung festgelegten Zeitraum. Zusätzliche Komplexität entsteht allerdings durch die Möglichkeit, ein Zertifikat durch den Besitzer für ungültig zu erklären (revoke), wenn dieser z.B. den Verdacht hegt, dass das zugrunde liegende Geheimnis oder zumindest ein Teil davon kompromittiert worden sein könnte. Ein Widerruf eines Zertifikats kann auch durch den Aussteller im Falle der Verletzung von Geschäftsgrundsätzen etc. durch den Nutzer erfolgen.
Dieses "Problem" führt in der Praxis dazu, dass man die momentane Gültigkeit eines Zertifikats prinzipiell anzweifeln und entsprechende aktuelle Informationen beim Aussteller einholen sollte. Der entwickelte Prototyp betrachtet diesen Sachverhalt aus zwei wichtigen Gründen *nicht*:

1. Die Ablage eines Zertifikats in irgendeiner Art von Verzeichnis an einer anderen Stelle, als direkt beim Aussteller selbst, wäre somit hinfällig (man müsste sich immer nach der momentanen Gültigkeit eines Zertifikats erkundigen und könnte dabei das Zertifikat auch gleich direkt vom Aussteller beziehen).
   Ein anderer Ansatz wäre die komplette Verwaltung der Zertifikate und ihres Lebenszyklusses direkt im UDDI-Register, was allerdings aus oben beschriebenen Platzgründen nicht möglich ist.

2. Für den Prototypen wird davon ausgegangen, dass sich Nutzer eines Services "regelmäßig"[39] aktuelle Zertifikate aus dem UDDI herunterladen und der Veröffentlicher dort abgelegt Zertifikate für den Fall eines Widerrufs aktualisiert.
   Eine Aktualisierung bedeutet dabei, dass das widerrufene Zertifikat entfernt und im UDDI-Register als auch beim Ziel-Webservice ersetzt wird.

7.2 Implementation

Für die Implementation des beschriebenen Lösungsvorschlags für authentifizierende Webservices wurde die UDDI-Register-Implementation (*Systinet Registry*) von Systinet [Systinet] in der Version 5.0 Evaluation verwendet.
In den folgenden Unterkapiteln wird das Anlegen der nötigen Datenstrukturen in diesem Register detailliert beschrieben.

7.2.1 Zertifikat - tModel

7.3 Validierung

Für die Validierung des vorgestellten Lösungsansatzes wird die Implementation in die beispielhafte Umsetzung des "Zugriffszähler für Webservice" (s.Kap 8) integriert.

[39] Die Sicherheit dieses Vorgehens kann mit Recht angezweifelt werden. Es kann keine Festlegung geben, wie oft "regelmäßig" bedeutet. Für Anwendungen mit einem sehr hohen Anspruch an Sicherheit, sollte die Gültigkeit *immer* beim Aussteller geprüft werden, wobei durch die Losgelöstheit der Zertifikate von der Verwaltung durch den Aussteller *nie* völlige Sicherheit erreicht werden kann (z.B. Widerruf eines Zertifikats im Zeitraum zwischen Prüfung und Verwendung).

Der dort benötigte Zeugenservice wird zusammen mit seinen für die Verschlüsselung notwendigen Zertifikate aus dem UDDI-Register bezogen.

8 Prototyp - Zugriffszähler für Webservices

In einer modernen serviceorientierten Architektur mit öffentlich oder zumindest teilweise öffentlichen zugänglichen Services wird sich in Zukunft immer häufiger die Frage stellen, wie man geleistete Dienste abrechnen kann. Eine Abrechnung kann dabei z.B. durch einen finanziellen Ausgleich, eine abgesprochene Begrenzung des Nutzungskontingentes o.ä. erfolgen.
Dazu ist es notwendig eine Möglichkeit zur Verfügung zu stellen, mit deren Hilfe eine verbindungsorientierte Zugriffszählung erfolgen kann. Jeder Dienst soll dadurch in die Lage versetzt werden können, einen Nachweis darüber zu erbringen, wer wie oft auf diesen zugegriffen hat. Jeder Betrugsversuch eines der Teilnehmer soll mit sehr hoher Wahrscheinlichkeit aufgedeckt und vor einer schlichtenden Instanz (z.B. einem Gericht) nachgewiesen werden können.
Ziel dieses Prototyps soll es nun sein, für den Bereich "Webservices" einen solchen Mechanismus zu entwickeln und umzusetzen.
Bereits in der Vergangenheit wurden Projekte bearbeitet, welche sich erfolgreich mit sehr ähnlichen Thematiken auseinandergesetzt haben. Als einen der wichtigsten Vertreter möchte ich das Projekt SEMPER in Kapitel 8.1 vorstellen. Im Rahmen dieses Projektes wurde von 1995 bis 1998 eine Sicherheitsarchitktur für den elektronischen Handel im Internet entwickelt und prototypisch umgesetzt.
Im Anschluss daran sollen in Kapitel 8.2 die Teilnehmergruppen einer entsprechenden Protokollumsetzung im Rahmen dieser Arbeit vorgestellt und grundlegende Fragen, wie Nutzeranonymität, Abrechnungsabläufe und Schadensregulierung beantwortet werden.
Bei der Beschreibung der theoretischen Umsetzung in Kapitel 8.3 werden zuerst verschiedene mögliche Protokollansätze vorgestellt, aus denen sich dann anhand einer Pro- und Contra-Bewertung der Protokollablauf herauskristallisieren soll, der im weiteren Verlauf dieser Arbeit verwendet werden wird. Anschließend wird dieses Protokoll im Detail vorgestellt und auf mögliche Angriffe und deren Abwehr untersucht.
Kapitel 8.4 stellt im Anschluss daran die konkrete Implementation für die Webservice-Technologie in der Ziel-Programmiersprache Java vor. Dabei wird besonders Wert darauf gelegt, bestehende Standards, welche einen Beitrag für die Realisierung des Prototyps leisten können, zu identifizieren und in die Implementation mit einzubeziehen.

8.1 Das Projekt SEMPER

Das Projekt "ACTS SEMPER" (Secure Electronic Marketplace for Europe; Förderungszeitraum 1995-1998) [SEMPER] am Lehrstuhl für Kryptografie und Sicherheit (Fachbereich für Informatik) der Universität des Saarlandes verfolgte das Ziel, eine flexible und offene Sicherheitsarchitektur für den sicheren elektronischen Handel im Internet zu entwickeln und in repräsentativen Feldversuchen zu evaluieren.
Dieser, auch unter dem Begriff "Fair Exchange" bekannte Vorgang bezieht sich auf typische elektronische Handelsszenarien, wie z.B. den Versandhandel oder Online-Vertragsaushandlungen und -unterzeichnungen bei denen Geschäftsobjekte, wie z.B. Informationen (Bilder, Videodaten etc.), Dokumente (Verträge, Beglaubigungen etc.) oder Geld (digital cash, Überweisungen etc.) ausgetauscht werden. Ein solcher Austausch muss *fair* vonstatten gehen, was bedeutet, dass beide Geschäftsobjekte (z.B. Ware gegen Geld) simultan versendet werden müssen. Jeder Partner muss dabei die Zusicherung haben, dass sein Geschäftsobjekt nur dann versendet wird, wenn er auch garantiert das erwartete Handelsobjekt vom Handelspartner empfängt.

Bei der Umsetzung wird in SEMPER zwischen *optimistischen* und *nicht-optimistischen* Abläufen unterschieden. Bei beiden Ablaufarten kommt ein Schiedsrichter zum Einsatz, welcher beim nicht-optimistischen Ablauf grundsätzlich und beim optimistischen Ablauf nur im Fehlerfall zum Einsatz kommt.
Bei einem optimistischen Protokoll (von den Autoren auf Grund seines geringeren Aufwandes bei korrektem Verhalten der Teilnehmer favorisiert [PS98, Sc00]) wird vor dem Beginn einer Transaktion eine Festlegung auf die auszutauschenden Objekte durchgeführt. Anschließend wird zuerst das Objekt, welches nicht vom Schiedsrichter ersetzt werden kann (z.B. eine Nachricht) übertragen. Dann wird das Geschäftsobjekt übertragen, welches der Schiedsrichter ersetzen könnte (z.B. ein Quittung). Wird dieses Geschäftsobjekt korrekt empfangen, endet das Protokoll ohne Beteiligung des Schiedsrichters. Falls jedoch das erste Geschäftsobjekt gesendet wurde, ohne dass der Sender das erwartete Objekt im Tausch erhalten hat, wir der Schiedsrichter einbezogen, um das zweite Objekt zu ersetzen. Dieser optimistische Fall macht es notwendig, dass jedes zuerst versendete Geschäftsobjekt auf seine Korrektheit (z.B. korrekte Bilddaten) geprüft werden muss, bevor das zweite Geschäftsobjekt (z.B. Bezahlung) ausgetauscht wird. Dies kann bei einem elektronischen Warenaustausch meist nur von einem Menschen erledigt werden.
Im Falle eines nicht-optimistischen Protokolls werden alle ausgetauschten (digitalen) Geschäftsobjekte grundsätzlich in einer bestimmten Art und Weise über den Schiedsrichter ausgetauscht. In SEMPER ist dies nur dann vorgesehen, wenn eine der beteiligten Parteien den Versuch unternimmt zu betrügen. In diesem Fall werden die kompletten Daten über den Schiedsrichter an den Empfänger gesendet.

Für den zu entwickelnden Prototyp zur Zugriffszählung bei Webservices geben die vorgestellten Ansätze von SEMPER eine wichtige Richtung vor, jedoch können sie nicht exakt in der Art verwendet werden, wie sie durch SEMPER beschrieben wurden.
Der optimistische Ansatz ist im Kontext dieses Themas nicht umsetzbar, da:

- eine Beurteilung der Ware (= Antwort eines Service) im Allgemeinen nicht durch einen Menschen durchführbar ist. Der Grund dafür liegt darin, dass sich komplexe Webservice-basierte Geschäftsabläufe aus einer Vielzahl von Anfrage-Antwort-Kombinationen zusammensetzen können, welche durch einen Menschen nicht mehr überschaubar sind bzw. einfach auf Grund der bloßen Anzahl nicht einzeln durchgesehen werden können.

- eine Antwort in starkem Maße von der jeweiligen Anfrage abhängt. Bei einem Streit, ob eine Antwort von einem Service korrekt geliefert wurde, muss der Schiedsrichter im schlimmsten Fall die nachweislich zu dieser Antwort gehörende Anfrage einsehen können.

- sich ein syntaktisch korrektes aber semantisch falsches oder sogar agressives Ergebnis (z.B. Einsparung von Rechenaufwand durch Senden einer "Standardantwort", logische Bomben, Viren etc.) einer Anfrage möglicherweise erst viel später offenbart und dann trotzdem noch die Möglichkeit bestehen muss, den Hergang zu rekonstruieren, obwohl bereits eine Quittung über den Erhalt ausgestellt wurde. Dazu gehören sowohl der Nachweis über den Inhalt der Anfrage als auch der Inhalt der Antwort, sowie evtl. der Inhalt von vor dem jeweiligen Kommunikationsvorgang durchgeführten Kommunikationen zur Bestimmung des Kontextes eines bestimmten Ablaufes.

Dies genannten Nachteile des optimistischen Ansatzes legen nahe, grundsätzlich eine Variante des nicht-optimistischen Ansatzes zu verwenden. Dabei muss allerdings auf Grund

der Vielzahl von potentiell notwendigen Kommunikationen zur Abwicklung eines einzigen Vorgangs im Webservice-Umfeld darauf geachtet werden, den zusätzlichen Kommunikationsaufwand mit dem unabhängigen Dritten (und evtl. auch bei den beiden Parteien untereinander) so gering wie möglich zu halten.

8.2 Protokollgrundlagen

Das zugrundeliegende Protokoll für diesen Zugriffszähler muss sicherstellen, dass die Bedürfnisse der Teilnehmer gewahrt, Verletzungen in Form von Betrugsversuchen aufgedeckt sowie Probleme auf Netzwerkebene kompensiert und korrekt behandelt werden.
Eine der wichtigsten Aufgaben ist es, sicherzustellen, dass Beweise an Teilnehmer so zugestellt werden, dass es keinem möglich ist, durch zeitlich günstigeren Erhalt daraus einen Nutzen zu ziehen. Als Beispiel sei dabei genannt, dass K das Ergebnis erhält und eine Bestätigung für dessen Erhalt an S verweigert. Dies hätte zur Folge, dass es S unter Umständen nicht mehr möglich ist, zu beweisen, dass er die Anfrage korrekt beantwortet und gesendet hat, wohingegen K sich über eine ungezählte Dienstleistung freuen könnte.
In den folgenden Abschnitten soll daher auf die Bedürfnisse der einzelnen Teilnehmer einer Kommunikation, die Notwendigkeit eines Zeugen und verschiedene grundlegende Betrachtungen für den Protokollablauf näher eingegangen werden.

8.2.1 Klient

Der Klient (im Folgenden auch mit K bezeichnet) stellt in diesem Szenario die Partei dar, welche einen bestimmten Service in Anspruch nehmen möchte. Ein Klient hat dabei folgende Ansprüche an das System ($A_K i$ kennzeichnet dabei den i-ten Anspruch des Klienten K an das System):

$A_K 1$ Anfragen, auf welche keine(n) Antwort(en) ankommen, dürfen nicht abgerechnet werden.

$A_K 2$ Eine gestellte Anfrage soll *korrekt* bearbeitet werden. "Korrekt" in dem Sinn, dass eine offensichtlich falsche oder bösartige Antwort (Viren etc.) nachgewiesen werden kann (z.B. vor einem Gericht).

$A_K 3$ Die Anfrage muss vom *gewünschten Service* bearbeitet werden.

Der Klient benötigt zur Wahrung dieser seiner Ansprüche für jede abgearbeitete Anfrage also folgende Nachweise/Beweise ($B_K i$ kennzeichnet dabei den i-ten notwendigen Beweis für den Klienten K):

($B_K 1$) Die gestellte Anfrage selbst (kann jederzeit wieder erzeugt werden und erhält somit als Beweis vor Dritten keinerlei Bedeutung; sollte allerdings für eventuelle spätere Untersuchungen archiviert werden).

$B_K 2$ Die zugehörige Antwort vom Service mit einem Nachweis darüber, dass diese von diesem erstellt wurde und dass sie mit der Anfrage verkettet werden kann (untermauert die Ansprüche $A_K 1$, $A_K 2$ und $A_K 3$).

Unterstellt man dem Klienten böswillige Absichten, ergeben sich unter anderem folgende Angriffsmöglichkeiten ($M_K i$ kennzeichnet dabei die i-te böswillige Motivation durch den Klienten K):

$M_K 1$ Einen Dienst in Anspruch nehmen, ohne vom Zählsystem registriert zu werden.

M_K2 Viele Dienste bzw. einen Dienst mehrmals in Anspruch nehmen und nur einmal vom Zählsystem erfasst werden.

8.2.2 Service

Als Service (im Folgenden auch mit S bezeichnet) wird in diesem Szenario die Partei bezeichnet, deren Aufgabe es ist, eine Anfrage zu bearbeiten. Folgende Ansprüche werden aus dessen Sicht an das System gestellt (A_Si kennzeichnet dabei den i-ten Anspruch des Service S an das System):

A_S1 Ein korrekt geleisteter Dienst muss auch korrekt zu Gunsten des Service gezählt werden. Die Inhalt und damit auch der Nachweis für die Bearbeitung einer Anfrage muss auch vor Dritten stichfest gezeigt werden können.

A_S2 Der Ersteller[40] einer Anfrage muss eindeutig feststellbar und vor Dritten nachweisbar sein. Dadurch soll vermieden werden, dass der Service z.B. im Rahmen einer DoS-Attacke durch legitime Teilnehmer "beschäftigt"[41] wird.

Zur Einhaltung dieser Anforderungen werden vom Service folgende Beweise/Nachweise benötigt (B_Si kennzeichnet dabei den i-ten notwendigen Beweis für den Service S):

B_S1 Die gestellte Anfrage und den Nachweis, dass diese von einem bestimmten Teilnehmer verfasst wurde (untermauert A_S2). Im Streitfall soll auch der Inhalt der Anfrage bewiesen werden können.

(B_S2) Die zugehörige Antwort mit einem Nachweise darüber, dass sie mit der gestellten Anfrage verkettbar ist (kann jederzeit wieder erzeugt werden und erhält somit als Beweis vor Dritten keine Bedeutung; sollte allerdings für eventuelle spätere Untersuchungen archiviert werden).

B_S3 Eine Bestätigung des Anfragenden darüber, dass er die Antwort erhalten hat. Ein Zusammenhang zwischen Bestätigung und Inhalt der Antwort muss nachweisbar sein (untermauert A_S1).

Für einen böswillig motivierten Service ergeben sich folgende Angriffsmöglichkeiten (M_Si kennzeichnet dabei die i-te böswillige Motivation durch den Service S):

M_S1 Abrechnen ohne die gewünschte Dienstleistung zu erbringen.

M_S2 Versuchen eine Dienstleistung mehrfach abzurechnen.

8.2.3 Außenstehende

Als Außenstehende werden alle Personen oder Systeme betrachtet, welche keine legitimen Teilnehmer an einer Kommunikationsbeziehung sind. Klient, Service oder Zeuge können dabei potentiell ebenso als Außenstehende eingestuft werden (z.B. ein Angriff in Zusammenarbeit mit einer nicht am Protokoll teilnehmenden Entität), wie alle anderen existierenden Entitäten.

[40] Ob sich dieser durch ein anonymes Pseudonym, welches im Aufdeckungsfall ein bestimmtes Protokoll erfordert, oder durch eine öffentliche Kennzeichnung ausweist, sei dahingestellt und wird nicht weiter betrachtet (siehe dazu [Pf00]).

[41] Dabei wird allerdings nicht weiter darauf eingegangen, dass selbst die Prüfung der Rechtmäßigkeit einer Nachricht und im negativen Fall die notwendige Antwort "Nein" für eine DoS-Attacke genügen können.

Für einen Außenstehenden gibt es aus Sicht des Entwicklers natürlich keine Ansprüche an das System und damit auch keine notwendigen Beweise, welche diesen zugeführt werden müssten.
Außenstehende können aber durchaus böswillig motiviert handeln und durch bestimmte Arten von Einflüssen Teilnehmer zur Kooperation zwingen oder, wie oben bereits erwähnt, selbst ein legitimer Teilnehmer sein, welcher über einen "anderen Port" kommuniziert.
Folgende Angriffe sind somit für alle existierenden kommunikationsfähigen Entitäten denkbar:

M_A1 Direkte Manipulation des Zählsystems zu seinen Gunsten unter Vortäuschung falscher Identitäten, Kommunikationsbeziehungen etc..

M_A2 Anfragen unter Verwendung der Identität eines anderen Teilnehmers an eine Service senden und so entweder auf dessen Kosten eine Antwort erhalten oder diesen Teilnehmer zumindest derart Schaden zufügen, dass er für eine Anfrage, welche er nie gesendet hat eine für ihn "nutzlose" Antwort erhält und dafür vom Zählsystem belangt wird.

M_A3 Sich als Zielservice ausgeben, obwohl man gar nicht der gewünschte Service ist und somit die Anfrage unrechtmäßig bearbeitet und die Gutschrift vom Zählsystem erhält.

8.2.4 Notwendigkeit eines Zeugen

Das beschriebene Szenario der Zugriffszählung schließt eine Kommunikationsbeziehung damit ab, dass ein Recht (im einfachsten Fall das Recht, einen Zähler zu erhöhen) vom Klienten zum Service und ein Ergebnis (kann ebenfalls als Recht betrachtet werden) vom Service zum Klienten übertragen wird. Überträgt einer der beiden das jeweilige Recht zuerst direkt an den anderen, so könnte die andere Partei die Übertragung des Rechts ihrerseits verweigern und sich so unrechtmäßig bereichern.
Über herkömmliche Netzwerke ist ein erzwungenes Abschicken der Rechte zur gleichen Zeit nicht möglich. Selbst wenn dies unter großem Aufwand gelänge (denkbar z.B. mit Hilfe eines DC-Netzes [Pf00]), so kann niemand sicherstellen, dass der tatsächliche Inhalt der abgeschickten Nachricht dann auch dem entspräche, was gewünscht ist oder dass eine Nachricht nicht durch einen zufälligen oder auch herbeigeführten Netzwerkfehler verändert würde. Eine schlichtende Instanz könnte im Anschluss daran auch nur sehr schwer entscheiden, wer von beiden betrogen hat, da Aussage gegen Aussage steht und jeder jede beliebige Information zurückhalten bzw. jederzeit selbst erzeugen kann.
Die Lösung dieses Dilemmas ist die Einführung einer dritten kontrollierenden Instanz: dem Zeugen (im Projekt SEMPER in Kapitel 8.1 auch als "Schiedsrichter bezeichnet").
Dieser ist dafür verantwortlich, im Falle eines Verstoßes gegen das Protokoll, zusammen mit der betrogenen Partei Beweise zu liefern, die es einer rechtsprechenden Instanz ermöglichen, den Betrüger eindeutig zu entlarven. Dabei darf allerdings nicht außer Acht gelassen werden, dass auch ein Zeuge Interesse an einem Betrug haben kann. In solch einem Fall muss es den beiden anderen Parteien möglich sein, dies nachzuweisen.

8.2.5 Zeuge

Obwohl über die Notwendigkeit eines Zeugen (s.Kap. 8.2.4; im Folgenden auch kurz mit Z bezeichnet) diskutiert werden kann, so stellt er im Falle einer Verwendung eine Partei innerhalb einer Kommunikationsbeziehung dar. Folgende Anforderungen werden durch einen

Zeugen an das System gestellt ($A_Z i$ kennzeichnet dabei den i-ten Anspruch des Klienten Z an das System):

$A_Z 1$ Eine Zusammenarbeit von Service und Klient darf nicht dazu führen, dass der Zeuge für ein "vermeintlich falsch" abgelegtes Zeugnis belangt werden kann.

Dafür muss der Zeuge in den Besitz folgender Beweise/Nachweise kommen:

$B_Z 1$ Die Anfrage mit dem Nachweis, von welchem konkreten Klienten sie gestellt wurde.

$B_Z 2$ Die Antwort mit dem Nachweis, von welchem Service sie geleistet wurde und zu welcher Anfrage sie gehört.

Anstelle der kompletten Anfrage bzw. Antwort würde auch ein von Service und Klient unterzeichneter Hashwert ausreichen. Dieser ermöglicht es im Streitfall, zusammen mit der betrogenen Partei den tatsächlichen Inhalt der Nachricht vor einer Dritten Instanz zu beweisen. Ein Zusammenschluss von Service und Klient gegen den Zeugen funktioniert in diesem Fall ebenfalls nicht, da der Zeuge die *unterzeichneten* Hashwerte nachweisen kann.

Wird ein Zeuge aus irgend einem Grund zu einer böswilligen Handlung motiviert, so ergeben sich folgende Angriffsmöglichkeiten ($M_Z i$ kennzeichnet dabei die i-te böswillige Motivation durch den Zeugen Z):

$M_Z 1$ Die Zusammenarbeit mit einem Klienten führt dazu, dass ein Service ohne Abrechnung benutzt werden kann.

$M_Z 2$ Die Zusammenarbeit mit einem Service führt dazu, dass Dienste zu Lasten eines Klienten abgerechnet werden, ohne dass ein Dienst erbracht wurde.

$M_Z 3$ Es wird Zeugnis derart abgelegt, dass der Tatbestand einer Kommunikation verzerrt wird, ohne selbst belangt werden zu können.

8.2.6 Anonymität

8.2.7 Abrechnung

Die Abrechnung[42] von erbrachten Leistungen kann in der realen Welt unter Verwendung unterschiedlichster Modelle erfolgen. Direkte Überweisungen von Geldbeträgen, Zugriffsbeschränkungen für bestimmte Zeiträume, Gegenrechnung von Zugriffen auf eigene Dienstleistungen durch andere mit eigenen Zugriffen auf deren Dienstleistungen stellen nur einen Teil möglicher Ansätze dar.
Die Zählung im zu entwickelnden Prototypen erfolgt dadurch, dass der Zeuge interne "Konten" in Form von Zählerständen verwaltet und für Zugriffe entsprechende Überträge realisiert.
Diese Konten könnten unter Verwendung von entsprechenden Authorisations- und Identifikationsmechanismen in Zusammenarbeit mit realen Banken auf tatsächliche Überweisungen abgebildet werden, da sowohl der Zählerstand beim Zeugen als auch der Kontostand bei der Bank auf abstrakter Ebene als "Anhäufung von Rechten" verstanden werden kann.
Auch ein "Aufladen" der Zählerstunde wäre so in umgekehrter Richtung denkbar.

[42] Der Zahlvorgang selbst wird im Folgenden als eine Übertragung von Rechten gesehen und auch entsprechend bezeichnet.

8.2.8 Schadensregulierung

Wird innerhalb des Protokolls ein Betrug festgestellt, so kann anhand der Beweislage eine schlichtende Instanz eingeschaltet werden (z.B. ein Gericht), welche dann ein verbindliches Urteil, unter Verwendung der unterbreiteten Beweise, fällen kann.
Das zu entwickelnde prototypische Protokoll soll entsprechende Beweise derart zuteilen, dass ein Angriff eines oder zweier veründeter Teilnehmer gegen andere Teilnehmer nachgewiesen werden kann.
Die verwendeten Grundlagen (Signatur-, Verschlüsselungs- und Hashalgorithmen) für Beweise beruhen auf kryptografischen Annahmen (s.Kap. 2.3) und sind zumindest aus Sicht des heutigen Kenntnisstands der Informatik ausreichend.

8.3 Theoretische Umsetzung

Im Folgenden soll Schritt für Schritt eine Lösung für einen Zugriffszähler erarbeitet werden, welcher in seiner endgültigen Form den Ansprüchen aller ober vorgestellten (möglicherweise) teilnehmenden Parteien entspricht.
Der Vollständigkeit halber soll im ersten Ansatz eine Lösung ohne Zeuge vorgestellt und anhand dessen gezeigt werden, dass dieser eine notwendige Einrichtung zur Vorbeugung vor Betrügereien darstellt (s.Kap. 8.3.1).
Daran anschließend wird eine Lösung vorgestellt, die einen Zeugen beinhaltet, der allerdings nur beim Senden der Antwort in die Kommunikationsbeziehung mit einbezogen wird (s.Kap. 8.3.2). Die nahe liegende Annahme einer gegebenen Praxistauglichkeit, da z.B. eine digitale Unterschrift (s.Kap. 2.3.5) unter der Anfrage ja für den Beweis B_S1 ausreichend ist und somit aus einem direkten Kommunikationsschritt zwischen Klient und Service auch ohne Zeuge zu entnehmen wäre, soll durch das Aufzeigen anderer Protokoll-Schwächen widerlegt werden.
Kapitel 8.3.3 beschäftigt sich mit dem Szenario, in dem sowohl Anfrage als auch Antwort im vollen Umfang über den Zeugen ausgetauscht werden.
Eine weitere Verbesserung zur Reduktion der übermittelten Datenmenge an den Zeugen wird in Kapitel 8.3.4 vorgestellt.
Um den Leser in den folgenden 4 Abschnitten nicht zu verwirren, wird auf die Angabe von notwendigen zu leistenden Signaturen und evtl. Verschlüsselung nicht im Detail eingegangen. Es sollen bereits an Hand des Ablaufs der Kommunikationsschritte Anfrage, Antwort, Bestätigung und des Kommunikationsumfangs Protokoll-Schwächen aufgezeigt werden, welche eine Weiterverwendung ausschließen.

8.3.1 Ohne Zeuge

In diesem Szenario (s.Abb. 15) schickt der Klient seine Anfrage direkt an den Service (1). Dadurch erhält der Service seinen ersten Beweis (B_S1) darüber, dass ein konkreter Klient eine Anfrage gestellt hat. Bricht er an dieser Stelle die Kommunikation ab (a) und verschickt keine Antwort, könnte er später behaupten, diese verschickt zu haben (in der gezeigten Abbildung mit dem gewählten Beispiel-Protokoll tritt allerdings der Fall ein, dass er ohne die Bestätigung aus Schritt (3) nicht an den notwendigen Beweis B_S3 käme).
Noch gravierender wäre das Protokoll-Versagen in dem Fall, dass der Service eine (korrekte) Antwort sendet und der Klient diese auch erhält, aber eine Bestätigung verweigert. Der Klient könnte dann behaupten nie eine Antwort erhalten zu haben (b). Dem Service würde dann der Beweis B_S3 fehlen, um einen Betrug seitens des Klienten nachzuweisen.
Dieses Beispiel verdeutlicht, dass ein vertrauenswürdiges Protokoll für einen Zugriffszähler

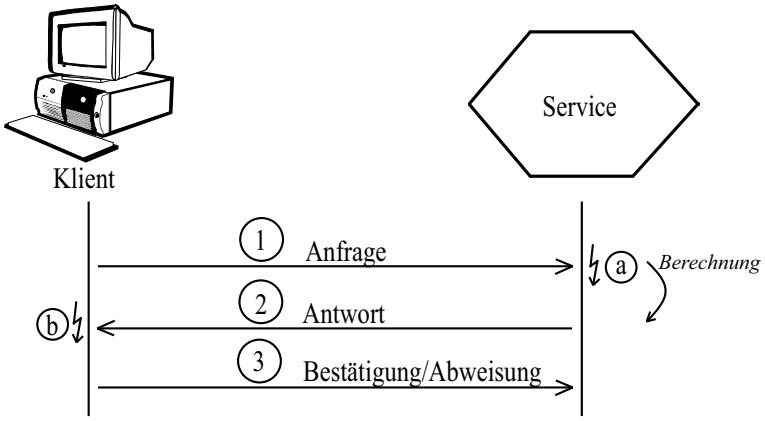


Abbildung 15: Zugriffszähler-Protokoll ohne Zeuge

nur unter Verwendung der Parteien Klient und Service nicht umsetzbar ist. Der Abbruch der Kommunikation durch einen Partner zu einem günstigen Zeitpunkt kann diesem genügend Vorteile verschaffen, einen Betrug ungestraft durchzuführen.
Zusammengefasst ergeben sich folgende Vor- (+) und Nachteile (-):

+ Kein Zeuge notwendig.
+ Wenige Kommunikationsschritte.
- Klienten(/Services) können leicht betrügen.

8.3.2 Mit Zeuge nur beim Senden der Antwort

Wird der Zeuge nur beim Senden der Antwort (und der Bestätigung) in die Kommunikationsbeziehung einbezogen, so ergibt sich ein Protokoll-Ablauf, wie in Abbildung 16 gezeigt. Der Klient stellt seine Anfrage direkt an den Service (1). Dieser erhält dadurch seinen Be-

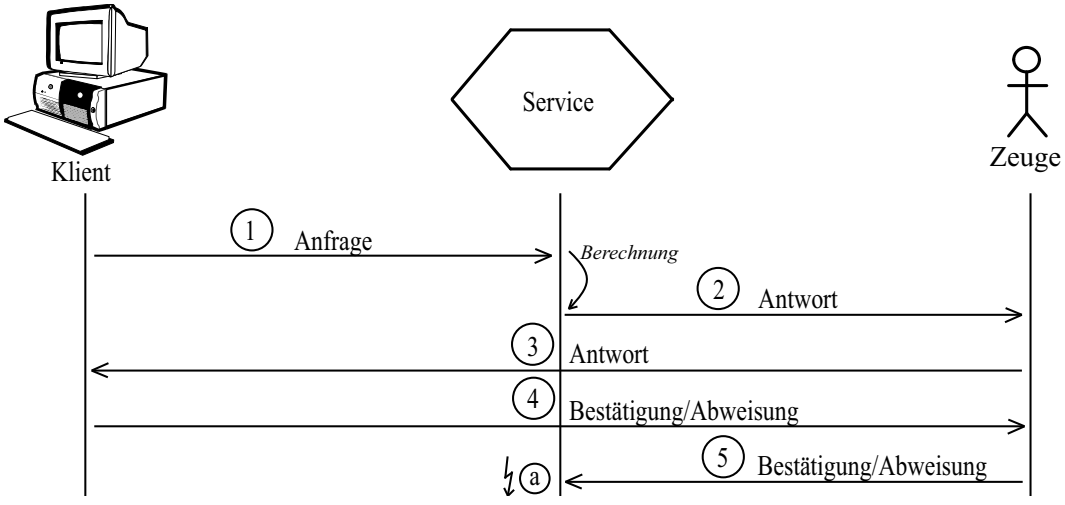


Abbildung 16: Zugriffszähler-Protokoll mit Zeuge bei Antwort

weis B_S1. Nach der Abarbeitung der Anfrage schickt dieser seine Antwort an den Zeugen

(2), welcher dadurch den Beweis B_Z2 erlangt. Im Schritt (3) erhält der Klient die Antwort vom Zeugen (BK_2). Der Klient ist nun Dank des Zeugen nicht mehr in der Lage willkürlich seine Bestätigung für den Erhalt zu unterlassen und sendet diese an den Zeugen (4), der diese dann seinerseits an den Service weiterleitet (5). Dieser gelangt dadurch in den Besitz des Beweises B_S3.
Dadurch dass der Zeuge für die Schlichtung eines Streits den Inhalt der Anfrage nicht "gesehen"[43] hat, könnte der Service, falls er eine inkorrekte Antwort geschickt hat, behaupten, er habe eine andere Anfrage erhalten, welche allerdings aus seiner Speicherung (B_S1!) verloren gegangen ist[44]. Daraufhin könnte der Klient zwar behaupten, er habe nie eine Anfrage geschickt, um so eine Bezahlung der Dienstleistung zu vermeiden bzw. rückgängig zu machen, jedoch könnte der eventuell durch die Antwort des Service angerichtete Schaden so nicht mehr zu dessen Lasten gelegt werden, da niemand mehr prüfen kann, ob die Antwort (z.B. in signiertem Zustand beim Kienten vorhanden) nun korrekt oder falsch war.
Weiterhin würde eine Zusammenarbeit von Klient und Zeuge die Sicherheit dieses Protokolls auf die Sicherheit des in Kap. 8.3.1 vorgestellten Protokolls ohne Zeugen reduzieren (Zeuge behauptet, die Antwort nie erhalten zu haben, obwohl er sie an den Klienten weitergegeben hat)
Zusammengefasst ergeben sich folgende Vor- (+) und Nachteile (-):

+ Einsparung des Kommunikationsschrittes mit dem Zeugen bei der Anfrage.
+ Klient kann den Empfang der Antwort nicht direkt leugnen.
+ Zeuge kennt übermittelten Inhalt der Antwort.
+ Relativ wenige Kommunikationsschritte.
- Zeuge hat keinerlei Informationen über den Inhalt der Anfrage (Lösen von Streitigkeiten!).
- Zeuge nimmt aktiv an der Kommunikation teil (Antwort).
- Zusammenarbeit von Klient und Zeuge ermöglicht Angriff.

8.3.3 Mit Zeuge

In diesem Fall laufen sowohl die Anfragen vom Klient als auch die Antworten durch den Service über den Zeugen (s.Abb. 17) Der Klient stellt sein Anfrage an den Zeugen (1, B_Z1), welcher diese an den Service weiterleitet (2, B_S1). Nach Bearbeitung der Anfrage wird die Antwort an den Zeugen (3, B_Z2) geschickt, welcher diese wiederum an den Klient weiterleitet (4, B_K2). Eine Bestätigung wird dann vom Klient über den Zeugen (5) an den Service geschickt (6, B_S3).
Bei der Betrachtung dieses Protokoll-Ablaufs fällt auf, dass alle Parteien in den Besitz ihrer oben beschriebenen notwendigen Beweise (B_K1 und B_S2 implizit) für eine mögliche Konfliktlösung kommen.
Zusammengefasst ergeben sich folgende Vor- (+) und Nachteile (-):

+ Klient kann den Empfang der Antwort nicht leugnen.

[43] Mit "gesehen" wird an dieser Stelle gemeint, dass der Binärstrom vom Klienten zum Service abgespeichert werden konnte (ob verschlüsselt oder unverschlüsselt sei dahingestellt und obliegt einer gesonderten Betrachtung).
[44] Plausible Gründe für solch einen Verlust kann es viele geben, z.B. physikalische Schäden, Softwareschäden, Einbruch etc..

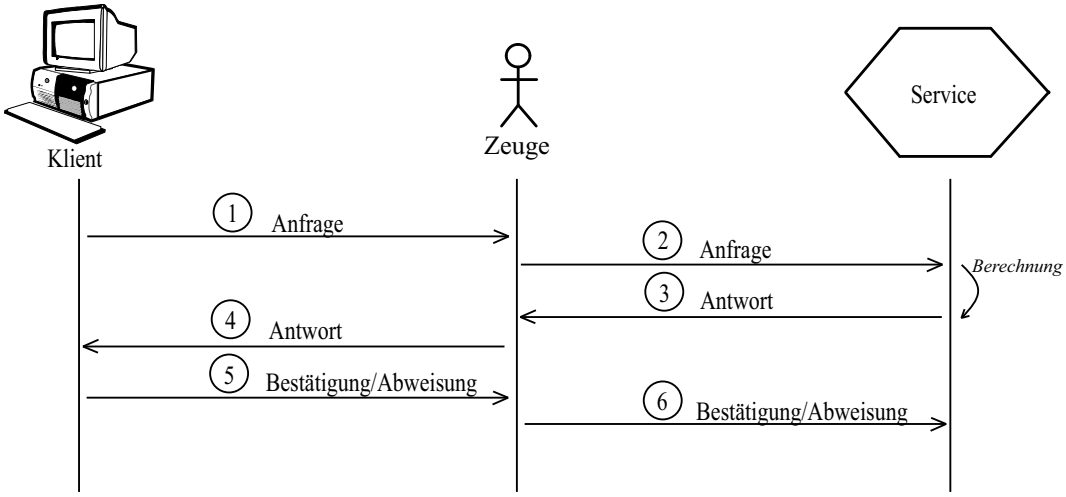


Abbildung 17: Zugriffszähler-Protokoll mit Zeuge bei Anfrage und Antwort

+ Klient kann den Inhalt der Anfrage nicht leugnen.

+ Server kann den Inhalt der Anfrage nicht leugnen.

+ Server kann den Inhalt seiner Antwort nicht leugnen.

+ Zeuge kennt den Inhalt der übermittelten Anfrage und der übermittelten Antwort.

- Viele Kommunikationschritte.

- Anfrage und Antwort werden in vollem Umfang über den Zeugen geschickt.

8.3.4 Große Datenmengen

Bei der Betrachtung der in Kap. 8.3.3 vorgestellten Lösung mit Zeuge fällt negativ auf, dass selbst bei einer Einhaltung des Protokolls durch Klient und Server grundsätzlich alle Kommunikationdaten über den Zeugen "umgeleitet" werden müssen. Besonders bei großen Datenmengen und der zusätzlichen Annahme, dass die Mehrheit der Teilnehmer nicht betrügen will, kann sich dieser Sachverhalt schnell als sehr enger "Flaschenhals" herausstellen. Bei einem Zeugen, welcher bestimmte vertrauenskritische Voraussetzungen erfüllen muss, ist es zudem sehr wahrscheinlich, dass er seine Aufgabe nicht nur bei einer einzigen Kommunikationsbeziehung wahrnimmt und somit seine verfügbare Bandbreite entsprechend aufteilen muss. Es wäre daher von großem Vorteil, das erarbeitete Protokoll derart abzuändern, dass der Zeuge nur dann eine Nachricht komplett erhalten muss, wenn möglicherweise ein Betrugsversuch vorliegt.
In Kapitel 2.3.6 wurde das Verfahren zur Erstellung von so genannten "Hashs" vorgestellt. Mit deren Hilfe wird es ermöglicht, eine Art Fingerabdruck einer Nachricht zu erstellen, welche erstens unabhängig von der Nachrichtenlänge eine konstante Länge aufweist und zweitens schwer zu fälschen ist. "Schwer zu fälschen" bezieht sich dabei auf das Finden einer Ursprungsnachricht, welche bei Anwendung der selben Hash-Funktion den selben Hash erzeugt (mal abgesehen von der zusätzlichen Schwierigkeit, auch noch eine Nachricht finden zu müssen, welche durch ihren Kontext bestimmt Vorgaben für Korrektheit erhält, z.B. ein bestimmtes Format oder fest vorgegebene Teildaten). Jeweils übereinstimmende Hashs der

Anfrage- und Antwortdaten von Klient und Server könnten somit die Grundlage für die Beweise B_Z1 und B_Z2 liefern und die Notwendigkeit für den Erhalt kompletter Nachrichten auf den Fall eines Betrugs reduzieren. In Abbildung 18 wird der Ablauf dieses Protokolls grob dargestellt. Der Klient sendet seine Anfrage direkt an den Service (1; B_K1 und B_S1)

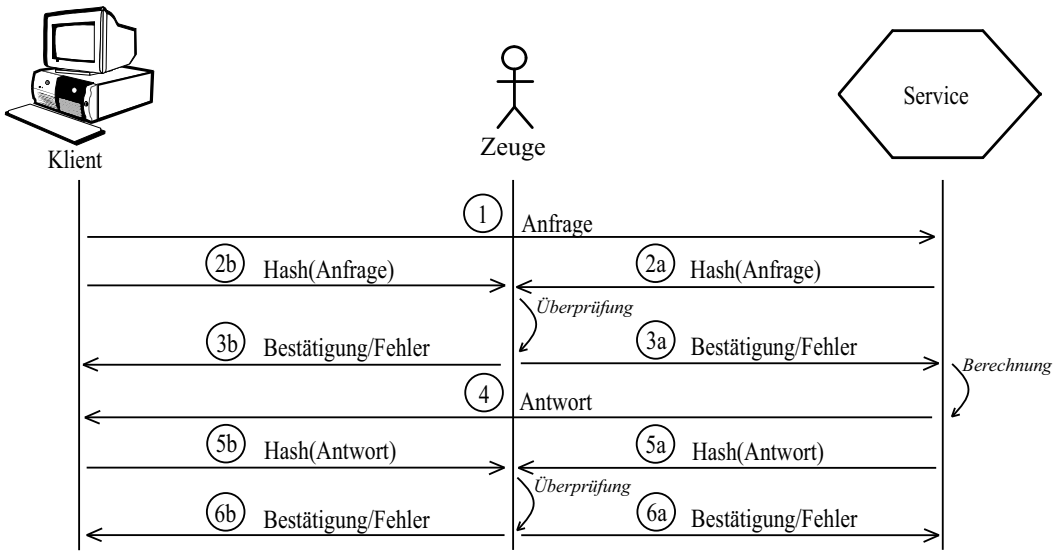


Abbildung 18: Zugriffszähler-Protokoll mit Hash-Informationen für Zeuge

und den zugehörigen Nachrichtenhash (2b) an den Zeugen. Dieser wartet auf den Erhalt des Nachrichtenhashs (2a) über die Anfrage vom Server und sendet bei Übereinstimmung der Hashs (entspricht B_Z1) eine entsprechende Bestätigung an den Server (3a; Start der Berechnung) und den Klient (3b; weiß nun, dass die Anfrage protokollkonform übermittelt wurde). Im Falle einer fehlenden Übereinstimmung der beiden Hashs sendet der Zeuge eine Fehlermeldung an die Teilnehmer (3a, 3b) und fordert so, abhängig vom jeweiligen Fall, entweder eine erneute Übertragung der Daten vom Klient zum Server (Annahme: physikalischer Fehler bei Übertragung), eine Übertragung der Daten vom Klient zum Zeugen und dann weiter zum Server (Annahme: Betrugsversuch) oder einen Abbruch (Annahme: massive Protokollverletzungen oder irreparable Netzwerkschäden). Die konkrete Umsetzung im Rahmen dieser Arbeit wird in den folgenden Kapitel spezifiziert.
Der Server bearbeitet nun die Anfrage, sendet die Antwort an den Klient (4; B_S2 und B_K2) und den errechneten Antworthash an den Zeugen (5a). Dieser wartet für die Prüfung auf den Erhalt des errechneten Antworthashs des Klienten (5b).
Liefert die Überprüfung auf Hash-Gleichheit beim Zeugen ein positives Ergebnis, so sendet der Zeuge eine Bestätigung an beide Teilnehmer (6a, 6b; entspricht B_Z2 und B_S3). Ist dies nicht der Fall, wo wird durch Senden einer Fehlermeldung (6a, 6b), wie auch bei 3a und 3b eine Forderung übermittelt, entweder direkt bzw. über den Zeugen neu zu senden oder die Kommunikation abzubrechen.
Das Protokoll ist damit beendet und führt bei einem positiven Abschluss zu einer Anrechnung der Service-Benutzung durch den Klienten. Später festgestellt Mängel z.B. an der Qualität einer Antwort sind nicht Gegenstand des Protokolls und werden mit Hilfe der entstandenen Beweislage notfalls vor Gericht geklärt.
Abschließend sei noch einmal darauf hingewiesen, dass sämtliche vorgestellten Kommunikationsschritte sowohl aus Klient-, Server- und Zeugensicht miteinander verkettbar sein

müssen. Es bietet sich daher an, wenn der Klient als Initiator der Kommunikation vor Schritt 1 mit dem Zeugen als Überwacher eine entsprechende Kennzeichnung aushandelt. Zusammengefasst ergeben sich für diesen Ansatz folgende Vor- (+) und Nachteile (-):

+ Klient kann den Empfang der Antwort nicht leugnen.

+ Klient kann den Inhalt der Anfrage nicht leugnen.

+ Server kann den Inhalt der Anfrage nicht leugnen.

+ Server kann den Inhalt seiner Antwort nicht leugnen.

+ Es werden nur die relativ zur vollständigen Nachricht kleinen Prüfsummen an den Zeugen geschickt.

+/- Zeuge kennt die Prüfsummen der übermittelten Anfrage und ihrere zugehörigen Antwort (Nachweis des konkreten Inhalts in Zusammenarbeit mit Klient oder Service möglich).

- Sehr viele Kommunikationschritte.

- Verwaltung eines relative komplexen Vorgangs durch den Zeugen als Koordinator.

8.3.5 Weitere notwendige Protokollfähigkeiten

Da es nicht sinnvoll ist, bei Dienstleistungen, welche über mehrere Kommunikationsschritte ablaufen (z.B. Reservierung mit anschließender Buchung beim Ticketverkauf) bzw. bei Kommunikationen mit mehreren Web Services, welche zusammen eine (abrechenbare) Dienstleistung erbringen, für *jede* Kommunikation einen neuen Kontext zu erzeugen, muss das erarbeitete Protokoll entsprechend erweitert werden.
Durch diese Erweiterung gewinnt eine weitere Fragestellung an Bedeutung: um wieviel wird unser "Zähler" bei der Verwendung mehrerer Web Services bzw. mehrere Kommunikationsschritte (z.B. eine Anfrage mit evtl. automatischer Reservierung im Erfolgsfall wird mit "+1" und eine anschließende Buchung mit "+2" auf den Zählerstand berechnet) erhöht und welcher Zählerstand wird bei der (z.B. transaktionalen) Verwendung mehrerer Services erhöht (hat jeder seinen eigenen Zähler oder berechnen mehrere/alle Services anhand eines gemeinsamen Zählers?).
Diese Fragen werden in die Erarbeitung des detaillierten Kommunikationsablaufs in Kapitel 8.3.7 mit einbezogen.

8.3.6 Anforderungen an das Protokoll

Um die Funktionsfähigkeit und die Sicherheit eines Protokolls zu zeigen, ist es eine gute Herangehensweise, wenn man sich vor der detailierten Definition der einzelnen Kommunikationsschritte Gedanken darüber macht, welche konkreten auflistbaren Anforderungen an das Protokoll bestehen. Diese beruhen im weitesten Sinne auf den in Kapitel 8.2.1 bis Kapitel 8.2.3 vorgestellten Anforderungen der einzelnen Teilnehmer und wurden an den konkreten Aufbau des in Kapitel 8.3.4 vorgestellten Protokolls angepasst.
Es bestehen folgende konkrete Anforderungen an die Umsetzung des Protokolls:

1. **Allgemein:**

   PA_1: Jeder Teilnehmer muss nachweisen können, dass der Vorgang von K initiiert worden ist.

PA_2: Jeder Teilnehmer muss nachweisen können, dass der Vorgang von Z bezeugt worden ist.

2. **Klient:**

   (a) Nachricht aus (3b) garantiert, dass zusammen mit dem Zeugen folgendes nachgewiesen werden kann:

   PA_3: Gewünschter Service S hat Anfrage unverändert erhalten.

   PA_4: Inhalt der Anfrage.

   PA_5: Service und Klient sind sich über den "Preis" der Leistung einig.

   PA_6: Abhängigkeiten zu beliebigen, vorher im Rahmen dieses Vorgangs ausgetauschten Nachrichten.

   (b) Nachricht aus (6b) garantiert, dass zusammen mit dem Zeugen folgendes nachgewiesen werden kann:

   PA_7: Antwort aus (4) stammt vom Service S.

   PA_8: Inhalt der Antwort.

   PA_9: Abhängigkeiten zu beliebigen, vorher im Rahmen dieses Vorgangs ausgetauschten Nachrichten. Insbesondere zur zugehörigen Anfrage.

3. **Service:**

   (a) Nachricht aus (3a) garantiert, dass zusammen mit dem Zeugen folgendes nachgewiesen werden kann:

   PA_{10}: Anfrage aus (1) stammt vom Klient K.

   PA_{11}: Inhalt der Anfrage.

   PA_{12}: Service und Klient sind sich über den "Preis" der Leistung einig.

   PA_{13}: Klient verfügt über notwendiges "Guthaben", um die anfallenden Kosten für die Leistung zu bezahlen.

   (b) Nachricht aus (6a) garantiert, dass zusammen mit dem Zeugen folgendes nachgewiesen werden kann:

   PA_{14}: Klient K hat Antwort korrekt erhalten.

   PA_{15}: Inhalt der Antwort.

   PA_{16}: Der Service hat die ausgehandelte Gutschrift erhalten.

   PA_{17}: Abhängigkeit der Antwort zur zugehörigen Anfrage.

4. **Zeuge:**

PA_{18}: Identifizierbarkeit von Klient und Server für die Abrechnung.

PA_{19}: Zuordenbarkeit von erhaltenen Nachrichten zu Entitäten und zu einem aktiven Vorgang.

PA_{20}: Beweisbare Grundlagen, auf welchen die Zusagen (3a, 3b, 6a, 6b) an den Klient K und den Service S beruhen.

Im folgenden Kapitel wird das erarbeitete Protokoll im Detail Schritt für Schritt vorgestellt und im Anschluss daran gezeigt (s.Kap. 8.3.8), wie dabei den einzelnen eben genannten Anforderungen genüge getan wurde.

8.3.7 Kommunikationsschritte im Detail

Betrachtet man das in Kapitel 8.3.4 und Abbildung 18 vorgestellte Protokoll auf einer weniger abstrakten Ebene stellen sich folgende Fragen, die einer weiteren Untersuchung bedürfen:

1. Welche Informationen müssen von welchem Teilnehmer signiert werden?
2. Welche Informationen sollten verschlüsselt werden?
3. Wie wird eine Anfrage mit ihrer Antwort und den nötigen Zwischenschritten verkettet?

Für die Beantwortung soll nun jeder Kommunikationsschritt in Bezug auf die einzelnen Fragestellungen betrachtet werden. Die Implementation des Prototypen (s.Kap. 8.4) beruht auf diesen Betrachtungen.
Eine Diskussion, ob die unterschiedlichen Verschlüsselungs- bzw. Signaturschritte sowie Nachrichteninhalte auch wirklich notwendig sind und welche möglichen Angriffe, Fehler oder Störungen durch das Protokoll wie abgefangen werden, wird in den darauffolgenden Abschnitten (Kap. 8.3.13 und 8.3.14) geführt.
Eine begleitende detailreichere Darstellung des verwendeten Protokolls, welches in Abbildung 18 in seinen groben Abläufen vorgestellt wurde, ist in Abbildung 19 dargestellt. Alle

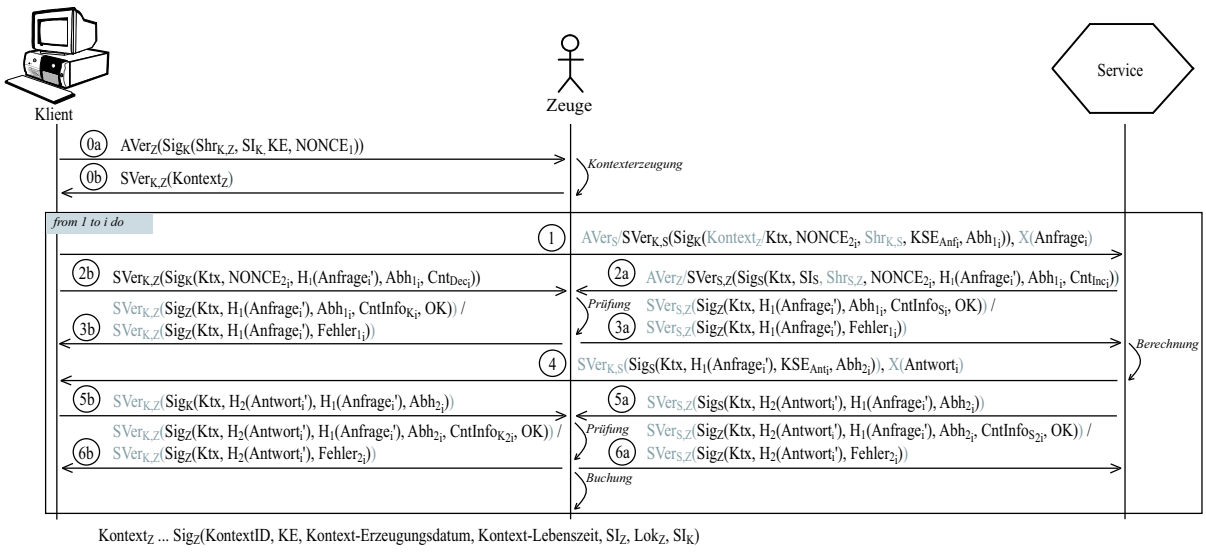


Abbildung 19: Zugriffszähler-Protokoll mit Hash-Informationen über die ausgetauschten Nachrichten für den Zeugen

folgenden Verweise auf bestimmte Kommunikationsschritte beziehen sich auf diese Abbildung. Eine Referenzierung eines Schrittes wird dabei wie folgt dargestellt: (<Schrittnummer>), z.B. für den "Schritt 4": (4).
Grundlegend soll dabei davon ausgegangen werden, dass der Klient über ein Zertifikat mit dem öffentlichen Testschlüssel eines digitalen Signatursystems sowie ein Zertifikat mit dem öffentlichen Schlüssel eines asymmetrischen Konzelationssystems des Zeugen[45] verfügt.

[45] Möglich wäre auch ein einziger Schlüssel eines kryptografischen Systems, welches sowohl asymmetrische Konzelation als auch digitale Signaturen unterstützt (z.B. RSA [Pf00]).

Diese können z.B. mit einer vertrauenswürdigen digitalen Unterschrift versehen von einer öffentlich zugänglichen Stelle bezogen oder vertraulich (persönlich) ausgetauscht worden sein[46]
Eine nähere Beschreibung der im Folgenden verwendeten Notationen kann aus Tabelle 3 auf Seite 86 entnommen werden.
Das x bzw. y ist dabei ein Platzhalter für K, S oder Z, was jeweils für eine der teilnehmenden Parteien Klient, Server oder Zeuge steht.
Das Ergebnis des Vorgangs $Sig_x(\ldots)$ ist immer als eine Einheit, bestehend aus der errechneten Signatur *und* den unveränderten signierten Daten selbst, zu verstehen. Die Signatur wird dabei allerdings nicht über die Ausgangsnachricht, sondern über deren Hash (wesentlich performanter), welcher ebenfalls Inhalt des Ergebnisses von $Sig_x(\ldots)$ ist, berechnet.

8.3.7.1 Initialisierung einer Kommunikationsbeziehung - 0

Um die Verkettung der einzelnen Kommunikationsschritte zu ermöglichen, müssen alle Nachrichten, welche zu einer konkreten Kommunikation (Anfrage und Antwort) gehören, mit ein und der selben eindeutigen Kennung versehen werden. Diese muss sowohl für den Klient, Server und auch den Zeugen eindeutig sein. Eindeutigkeit bedeutet in diesem Kontext, dass potentiell *nie wieder* eine Kennung vergeben werden darf, welche vorher schon einmal im Einsatz war. Der Grund dafür liegt in der Tatsache, dass z.B. Schäden, welche durch eine falsche Antwort vom Server entstehen und wofür dieser haftbar gemacht werden könnte, möglicherweise erst in ferner Zukunft erkannt werden können (z.B. transitive trojanische Pferde) oder überhaupt erst zu Tage treten (z.B. "logische Zeitbomben"). In der Regel wird man allerdings im Einklang mit allen Teilnehmern bestimmte Abmachungen diesbezüglich treffen (Verfallszeit für mögliche Ansprüche etc.).
Da der Zeuge die wichtigste Rolle für den korrekten Ablauf des Protokolls spielt (Koordinator) und diese Rolle wahrscheinlich bei einer Vielzahl von Klient-Server-Kommunikationen erfüllt, liegt es nahe, ihm die Generierung und Vergabe eindeutiger Kennungen zu überlassen. Dazu setzt er einer laufenden Nummerierung eine eigene eindeutige Kennung (z.B. URL) zur Unterscheidung von anderen Zeugen-Instanzen vor. Wie die laufende Nummerierung strukturiert ist, liegt bei der jeweiligen Implementierung und muss nur die Eindeutigkeit innerhalb aller durch diesen Zeugen koordinierten Kommunikationen innerhalb des festgelegten Zeitraums garantieren (z.B. aktuelle Zeit in Millisekunden).
Der Klient initialisiert den Kommunikationsvorgang, indem er eine Nachricht an den Zeugen schickt, in welcher er um die Erstellung eines Kontext für eine neue Kommunikationsbeziehung bittet (0a). Diese ist vom Klienten signiert ($S_K(\ldots)$) und ermöglicht es dem Zeugen unter Verwendung von SI_K, die Identität des anfragenden Klients festzustellen.
Der Parameter KE definiert protokollspezifische Eigenschaften (s.Kap. 8.3.12) für den zu erzeugenden Vorgang, z.B. eine Festlegungen auf zu verwendende Hash-, Signatur- und Verschlüsselungsalgorithmen.
Mit Hilfe des vom Klienten erstellten symmetrischen Schlüssels ($Shr_{K,Z}$, z.B. vom AES) werden alle folgenden Nachrichten zwischen dem Klienten und dem

[46] Bei einem persönlichen Treffen genügt es natürlich, den oder die öffentlichen Schlüssel auszutauschen. Die restlichen Informationen eines Zertifikats (z.B. Name, Adresse etc.) konnte man ja dabei mit eigenen Augen überprüfen.

Bezeichner	Beschreibung
$Shr_{x,y}$	Schlüssel eines symmetrischen Konzelationssystems
SI_x	Sicherheitsinformationen über die Entität x. Diese enthält Informationen, wie und woher Zertifikate zur asymmetrischen Verschlüsselung bzw. Prüfung einer digitalen Signatur von x bezogen werden können (z.B. von einer PKI mittels XKMS).
Lok_x	Informationen darüber, wie der Web Service Endpunkt einer Entität x erreichbar ist (z.B. der konkrete Port).
KE	Kommunikationseigenschaften, z.B. Definition von zu verwendenden Algorithmen, welche für einen logischen Kommunikationsvorgang Gültigkeit besitzen.
KSE_{Anf}	Eigenschaften für die Anfrage in einem Kommunikationsschritt (z.B. ob ein Abgleich über den Zeugen erfolgen soll oder falls nicht, wie bei späteren Verkettungen referenziert wird etc.)
KSE_{Ant}	Eigenschaften für die Antwort in einem Kommunikationsschritt (z.B. ob ein Abgleich über den Zeugen erfolgen soll oder falls nicht, wie bei späteren Verkettungen referenziert wird etc.)
$Kontext_Z$	Vom Zeugen erstellter Kontext, welcher von Z signiert ist, und eine feste Verbindung zum initiierenden Klient K besitzt.
Ktx	Eindeutig dem $Kontext_Z$ zuweisbare Referenz (entweder der $Kontext_Z$ in vollem Umfang selbst, seine ID, sein Hash etc.)
$CntInfo_x$	Information über den Kontostand der jeweiligen Entität. Bei einem Vorgang enthält dieser Parameter zusätzlich die Bestätigung, dass die Preisverhandlungen erfolgreich waren (Mitsenden von Cnt_{Inc} bzw. Cnt_{Dec}).
Cnt_{Dec}	Angabe, in welcher Art das Konto/der Zähler eines Klienten bei erfolgreicher Ausführung belastet/verringert werden soll.
Cnt_{Inc}	Angabe, in welcher Art das Konto/der Zähler eines Services bei erfolgreicher Ausführung eine Gutschrift/Erhöhung erfährt.
$NONCE$	Eindeutiger und bedeutungsloser (z.B. aktuelle Zeit in Millisekunden) Wert, wodurch Wiederholungsangriffe (s.Kap 5.2.2) verhindert werden können.
$Anfrage$	Beliebige Anfragedaten.
$Anfrage'$	Anfragedaten nach möglichen festgelegten Modifikationen (z.B. Entschlüsselung, Entfernung von Signaturen etc.).
$Antwort$	Beliebige Antwortdaten.
$Antwort'$	Antwortdaten nach möglichen festgelegten Modifikationen (z.B. Entschlüsselung, Entfernung von Signaturen etc.).
Abh	Festlegung von Abhängigkeiten (z.B. bei einer Antwort die Verkettung mit der zugehörigen Anfrage).
OK	Bestätigung des Zeugen, dass alle Prüfungen positiv verlaufen sind.
$Fehler$	Mind. eine Prüfung des Zeugen ist negativ verlaufen. Die Fehlermeldung beschreibt den aufgetretenen Fehler sowie die weitere Vorgehensweise.
$Sig_x(\ldots)$	Vorgang der Erstellung einer digitalen Signatur unter Verwendung von S_x.
$AVer_x(\ldots)$	Vorgang der asymmetrischen Verschlüsselung unter Verwendung von Pub_x.
$SVer_{x,y}(\ldots)$	Vorgang der symmetrischen Verschlüsselung unter Verwendung von $Shr_{x,y}$.
$H(\ldots)$	Vorgang der Erstellung eines Hashs über den angegebenen Daten.
$X(\ldots)$	Wird verwendet um zu zeigen, dass die enthaltenen Daten beliebig verschlüsselt/signiert werden können ohne das Protokoll zu beeinflussen.

Tabelle 3: Verwendete Notationen

Zeugen, welche einen Anspruch auf Verschlüsselung besitzen, mit geringerem Aufwand verschlüsselt, als mit asymmetrischen Verfahren.
Es sei an dieser Stelle zusätzlich erwähnt, dass eine konkrete Implementation $Shr_{K,Z}$ bereits in diesem Schritt verwenden könnte, indem ein hybrides Verfahren angewendet wird:
$AEnc_Z(Sig_K(Shr_{K,Z})), SEnc_{K,Z}(Sig_K(SI_K, \ldots))$.
Der $NONCE_1$-Wert garantiert, dass kein Angreifer durch erneutes Senden einer abgehörten Anfrage einen Wiederholungsangriff (s.Kap. 5.2.2) oder einen Angriff auf die Verfügbarkeit des Zeugen mit der "Identität des Klienten" durchführen kann.
Der Zeuge erstellt intern einen Kommunikationskontext und übermittelt die zugehörige Kontextinformationen, versehen mit seiner Signatur als $Kontext_Z$ und verschlüsselt mit dem ausgetauschten symmetrischen Schlüssel $Shr_{K,Z}$ an diesen zurück (0b). Diese Kontextinformation besteht aus einer eindeutigen ID ($KontextID$), dem Zeitpunkt der Erstellung des Kommunikationskontextes und seinem Gültigkeitszeitrum, dem Port für den Zeugenservices (Lok_Z) sowie der Information, woher Zertifikate des Zeugen (SI_Z) bzw. des Klienten (SI_K) für die asymmetrische Verschlüsselung und das Testen digitaler Signaturen bezogen werden können. Der im Kontext enthaltene Wert $NONCE_1$, der aus der Anfrage des Klienten (0a) entnommen worden sein muss, ermöglicht es dem Zeugen nachzuweisen, dass $Kontext_Z$ auf eine legitime Anfrage von K hin erstellt worden ist.
Die Signatur unter dem Kontext stellt sicher, dass der Klient auch tatsächlich vom gewünschten Zeugen eine Kennung zugewiesen bekommt und dies auch vor Dritten beweisen kann.
$Kontext_Z$ wird im weiteren Protokollablauf von den teilnehmenden Parteien, neben den enthaltenen Informationen, dafür verwendet werden, auch bei einer Vielzahl von weiteren nebenläufigen Kommunikationen mit asynchron verschickten Nachrichten eine Zuordnung zum jeweiligen Kommunikationsvorgang zu garantieren.

8.3.7.2 Übertragung der Anfrage - 1

Wurde der Protokoll-Kontext vom Zeugen erstellt, so kann die gewünschte Anfrage vom Klient an den Server geschickt werden. Bei diesem Kommunikationsschritt ist der Protokollkopf mit dem öffentlichen Schlüssel des Service verschlüsselt ($AEnc_S(...)$) und sichert damit, dass nur der Server die enthaltenen Daten entschlüsseln kann.
Auch für diesen Kommunikationsschritt sei am Rande erwähnt, dass eine konkrete Implementation, um den Rechenaufwand, wie bei Schritt 0a, zu verringern, eine hybride Verschlüsselung bestehend aus $AEnc_S(\ldots)$ und $SEnc_{K,S}(\ldots)$, ermöglicht durch den übermittelten Schlüssel $Shr_{K,S}$, verwenden würde.
Handelt es sich bei dieser Kommunikation allerdings nicht um die erste Anfrage an den Service, so kann bereits komplett symmetrisch verschlüsselt werden, da der zugehörige Schlüssel bereits ausgetauscht worden ist. Ähnliches gilt auch für den zu übermittelnden Kontext, welcher im ersten Durchlauf als $Kontext_Z$ übertragen werden muss, wohingegen in den weiteren Durchläufen die Übermittlung von Ktx für eine Zuordnung genügt.
Ein wichtiger Anspruch, welcher bei der Erstellung dieses Protokolls immer im

Hinterkopf behalten wurde, ist der, dass die eigentliche Anfrage keinen Restriktionen in Form von notwendigen Signaturen, Verschlüsselungen o.ä. unterworfen ist. Die ausgegraute Funktion $X(\ldots)$ zeigt an, dass dennoch unabhängig vom Protokoll jede nötige Operation zur Wahrung von Integrität, Vertraulichkeit usw. auf den eigentlichen Anfragedaten ausgeführt werden kann.
Der $NONCE_2$-Wert, welcher nur vom Zielservice entschlüsselt werden kann, verhindert erstens die Wiederholbarkeit der Anfrage durch Außenstehende und zweitens wird über den Zeugen nach Erhalt von (2a) und (2b) (mit enthaltenem $NONCE_2$) eine Prüfung erfolgen, ob sich der vom Klient gewünschte Service für die Abrechnung anmeldet. Die Verschlüsselung verhindert eine Veröffentlichung von $NONCE_2$, welche es anderen Services ermöglichen würde, sich für die Abrechnung zu registrieren und das Protkoll so empfindlich zu stören.
KSE_{Anf} enthält durch den Klienten spezifizierte Festlegungen an S für diesen Kommunikationsschritt, ob z.B. eine "Abrechnung" über den Zeugen für diesen Kommunikationsschritt überhaupt gewünscht ist. Dies kann z.B. der Fall sein, wenn einer kostenpflichtigen Operation eine kostenlose Operation vorausgeht (z.B. Reservierung mit Preiszusage und anschließender Buchung). Dort ist es nicht wünschenswert, *jeden* Schritt über den Zeugen abzurechnen. Über eine entsprechende Festlegung in KSE_{Anf} kann dies unterdrückt und parallel dazu eine Methode vorgeschlagen werden, um später mit Hilfe von Abh_1 darauf zu referenzieren.
Gewünschte Abhängigkeiten werden durch eine entsprechende Angabe mit Hilfe des Parameters Abh_1 angegeben. Was Abhängigkeiten im Einzelnen sein können und wie sie innerhalb des Abh_1-Parameters festgehalten werden, wird im Kapitel 8.3.9 näher vorgestellt.
Die Signatur durch den Klient stellt sicher, dass der Kontext, $NONCE_2$, KSE_{Anf}, Abh_1 und eventuell auch $Shr_{K,S}$ wirklich unverändert und nachweislich von K beim Service S ankommen. Diese Signatur ist notwendig, weil sonst bei einer Verschlüsselung mit dem öffentlichen Schlüssel von S potentiell jeder, der Kenntnis von $Kontext_Z$ hat (z.B. der Zeuge) und bei der Verwendung von $Shr_{K,S}$ für die Verschlüsselung der Service selbst eine nicht zurückverfolgbare, zumindest für den Schritt (1) legitim aussehende Anfrage formulieren könnte. Spätestens das Ausbleiben von (2b) würde diesen Sachverhalt enttarnen, aber bis dahin würden sowohl für S und Z als auch bei der Fehlerbehandlung für K ("Nein, ich habe diese Anfrage nicht gesendet") viele unnötige Kommunikationen entstehen.

8.3.7.3 Übertragung der Anfrage-Prüfsummen (Hash) - 2a/2b

Dieser, aus zwei Kommunikationen bestehende Schritt registriert den Service als Ziel für eine mögliche "Bezahlung" nach dem Erhalt der zugehörigen Antwort durch den Klienten (2a).
Der Service übermittelt dafür seine Sicherheitsinformation SI_S für die Prüfung der Signatur und für die Feststellung der Identität durch den Zeugen. Dem Zeugen wird so ermöglicht, das Konto des Services für die Gutschrift zu identifizieren. SI_S muss theoretisch nur im allerersten Kommunikationsschritt die Information enthalten, woher und wie die zugehörigen Testschlüssel zur Signaturprüfung etc. bezogen werden (bzw. das Zertifikat selbst), da es bei weiteren

Kommunikationen genügt, dieses durch einen zugehörigen, im ersten Durchlauf vereinbarten Namen zu referenzieren (z.B. "Schlüssel von <ServiceName> im Kontext <KontextID>"). Siehe dazu die Diskussion in Kapitel 8.3.13.
Ktx ermöglicht die Zuordnung zu einem der durch den Zeugen verwalteten Vorgänge. Dieser Parameter kann entweder $Kontext_Z$ selbst oder aber eine durch Z einem konkreten Vorgang zuordenbare eindeutige Referenz (z.B. $KontextID$) sein.
Der Wert $NONCE_2$ ist der aus (1) entnommene und nur durch S entschlüsselbare Nachrichtenauthentifikator.
$H_1(Anfrage')$ ist der unter Verwendung eines mit K vereinbarten Algorithmus H_1 entstandenen Hashwert über die Anfrage. Ob dafür die Nachricht, wie sie übermittelt wurde, oder nach einer Transformationen (z.B. Entfernung von Verschlüsselung, Signaturen etc.) verwendet wird, ist ein Implementationsdetail, welches durch das Protokoll vorgeschrieben oder aber zwischen K und S vereinbart worden sein kann.
Die Abhängigkeiten (Abh_1) stellen Referenzen auf vorher ausgetauschte Nachrichten zwischen K und S dar, welche identisch zu Abh_1 in (1) sind. Entspricht Abh_1 aus (1) nicht den Vorstellungen des Services, so müsste dies außerhalb des Protokolls verhandelt und (1) aktualisiert übermittelt werden.
Cnt_{Inc} enthält die durch S übermittelte Vorstellung des "Preises" für die Nutzung seiner Dienstleistung. Diese sollte natürlich vorher mit K ausgehandelt bzw. von diesem aus einer Art Preisliste von S ersichtlich gewesen sein.
Die Signatur über allen Parametern stellt den Beweis für Z dar, dass diese Nachricht unverändert übertragen und von S verfasst worden ist.

Mit Hilfe von (2b) bestätigt K den Versandt der in (1) übermittelten Antwort. Dazu schickt er ebenfalls Ktx, $NONCE_2$, $H_1(Anfrage')$ und die ausgehandelten Abhängigkeiten an Z. Die Signatur stellt auch hier sicher, dass die Nachricht unverändert übertragen und von K verfasst worden ist. Sie stellt weiterhin sicher, dass Z im Falle einer Zusammenarbeit von K und S nachweisen kann, dass seine in (3a) und (3b) übermittelten Bestätigungen rechtmäßig verfasst worden sind.

Die Verschlüsselung von (2a) und (2b) mit einem asymmetrischen, hybriden oder symmetrischen Konzelationssystem ist zumindest für den Wert $NONCE_2$ unbedingt notwendig, da es sonst einem Angreifer durch Abfangen der Nachricht gelingen könnte, sich als Zielservices für das Zählsystem bei Z zumindest anzumelden. Spätestens in Schritt (4) würde dies allerdings erkannt werden können, wenn mit einem symmetrischen System (Sessionschlüssel $Shr_{K,S}$) verschlüsselt wird. Denn nur der wirkliche Zielservice kann beim ersten Durchlauf in Schritt (1) in den Besitz von $Shr_{K,S}$ gelangen. Außerdem ist nur der Zielservice S in der Lage in Schritt (4) die korrekte Signatur zu erzeugen. Trotzdem könnte das Protokoll in seiner Funktionsfähigkeit deutlich eingeschränkt werden.

Hat Z sowohl (2a) als auch (2b) erhalten, kann er diese anhand von Ktx einem offenen, intern verwalteten Vorgang zuordnen. Unter einer zusätzlichen Betrachtung von $NONCE_2$ kann der Zeuge die Nachrichten aus (2a) und (2b) einander zuordnen und mit der eigentlichen Prüfung der enthaltenen Daten beginnen.
Bei dieser Prüfung müssen die Hashes ($H_1(Antwort')$) und die angegebenen Abhängigkeiten beider Nachrichten übereinstimmen. Der Vergleich von Cnt_{Inc}

und Cnt_{Dec} muss ebenfalls in einer Übereinstimmung enden. Bei diesem Vergleich sind Äquivalenz der Nennwerte oder aber auch spezielle zusätzliche Operationen denkbar (z.B. unterschiedliche Währungen anhand von Tageskursumrechnung oder aus einer Liste entnommenen Preise in unterschiedlichen Währung für diesen konkreten Service).
Eine weitere Aufgabe des Zeugen besteht darin zu prüfen, ob der Klient überhaupt noch über das nötige Guthaben verfügt, um den Dienst des Services in Anspruch nehmen zu können. Dazu verwaltet er intern den Zähler/das Konto mit einer Liste von "Zahlungsreservierungen", welche sich durch das Versenden der "OK"-Nachrichten in (3a) bzw. (3b) ergibt und möglicherweise eine bestimmte Verfallszeit (Timeout) für eventuell ausbleibende Antworten besitzt. Erst mit der Bestätigung des Antworterhalts aus (6a) bzw. (6b) wird der Wert endgültig dem Service-Konto gutgeschrieben und vom Klienten-Konto entfernt.

8.3.7.4 Bestätigung/Fehlermeldung für die Anfrage vom Zeugen - 3a/3b

Anhand der nach Kommunikationsschritt (2a) und (2b) durchgeführten Überprüfung durch den Zeugen kann dieser eine Entscheidung über den Inhalt seiner Antwortnachricht an K und S fällen.
War das Ergebnis der Überprüfung positiv, schickt der Zeuge eine "OK"-Nachricht an K und S. Für S bedeutet dies, er kann zusammen mit Z nachweisen, dass die Anfrage von K verschickt worden ist und welche Daten sie beinhaltete. K weiß, dass die Nachricht beim richtigen Service angekommen und nun von ihm bearbeitet werden wird.
Dies (3a/3b)-Nachricht beinhaltet Ktx für die Zuordnung zu einem konkreten Vorgang durch K und S, den in (2a) bzw. (2b) übermittelten, auf Identität geprüften Hash $H_1 Anfrage'$ als Nachweis, dass es sich bei dieser Bestätigung um eine Bestätigung für die erwähnte Anfrage handelt und das beide Hashes übereinstimmten, die Abhängigkeiten Abh_1 als Nachweis, das die angegebenen Abhängigkeiten in (2a) und (2b) identisch waren und den jeweiligen übermittelten Preis (Cnt_{Inc} in (3a) und Cnt_{Dec} in (3b)) innerhalb von $CntInfo_x$ als Nachweis, dass die jeweils übermittelten Preisvorstellungen mit der des anderen identisch waren und die Erhöhung/Verringerung des "Kontostandes" nach Prüfung von den zugehörigen Antwortbestätigungen (5a) und (5b) erfolgen wird.
Die Signatur von Z um alle Parameter dient K und S als Beweis, dass die Prüfung durch Z positiv verlaufen ist. Der Zeuge kann somit später nicht mehr behaupten, dass die Prüfung negative verlief oder gar nicht stattgefunden hätte. Das innerhalb der selben Signatur enthaltenen Flag "OK" stellt sicher, dass eine Verwechslung mit einer eventuellen Fehlermeldung ausgeschlossen werden kann.

Schlug eine der nach (2a) bzw. (2b) durchgeführten Prüfungen fehl oder kam eine der beiden Nachrichten gar nicht erst an, so wird dies vom Zeugen mit einer Fehlermeldung quittiert. Wie diese Fehlermeldung und die daraus resultierende Fehlerbehandlung für die einzelnen Fälle konkret aussieht, wird in Kap. 8.3.14 näher erläutert.
Eine Fehlermeldung enthält für die Zuordnung zu einem konkreten Vorgang bei K und S die Werte Ktx und $H_1(Anfrage')$. Der Hashwert, muss dabei, falls dort ein Fehler festgestellt wurde, für (3a) aus (2a) und für (3b) aus (2b) entnommen worden sein.

Über den Parameter $Fehler_1$ wird den Kommunikationsteilnehmern eine detaillierte Beschreibung des aufgetretenen Problems übermittelt und eventuell eine Fehlerbehandlung vorgeschlagen.
Die Signatur von Z stellt auch hier sicher, dass Z nicht in der Lage ist, die negative Prüfung oder das Stattfinden des Prüfungsvorgangs zu leugnen.
Sowohl bei (3a) als auch bei (3b) ist eine Verschlüsselung des Signaturbereichs empfehlenswert, aber nicht zwingend möglich. Die Empfehlung zur Verschlüsselung wird damit begründet, dass sonst Außenstehende an Informationen gelangen können, welche es Ihnen ermöglichen z.B. über Kontostandsinformationen, Umsätze von S und K, eventuell augehandelte "Sonderpreise" oder über Fehlerinhalte Bescheid zu wissen sowie an bestimmte Verkettungsinformationen zu gelangen (z.B. könnten mehrere Fehlermeldungen auf Grund ihrer möglicherweise identischen Werte $H_1(Anfrage')$ untereinander und mit einer mögliche "OK"-Nachricht am Ende verkettet werden).

8.3.7.5 Übertragung der Antwort - 4

Hat der Service S vom Zeugen in Schritt (3a) eine "OK"-Nachricht erhalten, so kann dieser mit der eigentlichen Bearbeitung der Anfrage beginnen. Deren Ergebnis wird in Schritt (4) an K übertragen. Bei diesem Schritt wurde bei der Erstellung des Protokolls ebenfalls darauf geachtet, dass der Inhalt der Antwort an sich (im Normalfall) keinen Restriktionen, wie zwingender Signatur oder Verschlüsselung unterworfen wird. Dadurch soll, wie bei der Anfrage auch, eine möglichst leichte Integration in bestehende Anwendungen und eine Verwendung nach dem WS-* üblichen Baukastenprinzip (nur die Komplexität zu einer SOAP-Nachricht hinzufügen, die auch wirklich benötigt wird; s.Kap. 1.1) ermöglicht werden. Das ausgegraute $X(\ldots)$ signalisiert auch hier, dass, wenn nötig, jegliche kryptografischen Operationen auf den Antwortdaten, ohne Einfluss auf das Protokoll, durchgeführt werden kann.
Für die Zuordnung der Antwort zu einer von K gestellten Anfrage in einer asynchronen Mehrnachrichten-Umgebung wird Ktx und $H_1(Anfrage')$ in den Nachrichtenkopf eingefügt (der Hash wäre auch als Bestandteil von Abh_2 denkbar, wurde aber zum besseren Verständnis extra aufgelistet).
KSE_{Anf} enthält auch hier an K gerichtete Metainformationen für die Weiterbehandlung der erhaltenen Informationen aus diesem Kommunikationsschritt (z.B. Abrechnung über Z nötig?) und mit Hilfe von Abh_2 spezifiziert S seine Wünsche für mögliche durch Z zu bezeugende Verkettungen zu vorher übermittelten Nachrichten.
Die Signatur stellt sicher, dass die oben beschriebenen Parameter auch tatsächlich von S übermittelt und nicht manipuliert wurden.
Die ausgegraute Verschlüsselungoperation $SVer_{K,S}(\ldots)$ stellt eine empfehlenswerte Option dar. Prinzipiell ist es hier zwar nicht nötig zu verschlüsseln, aber zur Verhinderung der Protokollierung von Verkettungsinformationen durch Außenstehende (z.B. $H_1(Anfrage')$ lässt sich bis (1) zurückverfolgen, falls $Anfrage = Anfrage'$, da jeder den Hash berechnen kann) wird dies nahe gelegt.

8.3.7.6 Übertragung der Antwort-Prüfsummen (Hash) - 5a/5b

In diesem Schritt übermitteln K und S Nachrichten an Z, welche erstens den Inhalt der Antwort nachweisbar machen und zweitens den Erhalt der Antwort

bei K beweisen sollen.
Dazu übermittelt der Service S in (5a) für eine Zuordnung durch Z zu einem Vorgang und zu einer Anfrage Ktx und $H_1(Anfrage')$ an den Zeugen. $H_1(Anfrage')$ wird dabei stillschweigend als Identifikator für die zugehörige Anfrage angenommen. Es würde theoretisch genausogut funktionieren, wenn Z in (3a) und (3b) neben dem Hash eine innerhalb des Vorgangs (Ktx) eindeutige (kürzere) Kennung zur Identifizierung der Anfrage übermittelt hätte (z.B. $AnfrageID$) und diese nun zur Referenzierung verwendet würde. Um die Protokollbeschreibung möglichst einfach zu halten wird der Hash weiterhin als eindeutiger Identifikator für Nachrichten innerhalb von Ktx verwendet (eine konkrete Implementation könnte dies z.B. für Einsparungen an der zu übertragenden Datenmenge durchaus anders lösen). Weiterhin könnte $H_1(Anfrage')$ auch als Teil von Abh_2 betrachtet werden, was an dieser Stelle zu Gunsten der Verständlichkeit des Protokolls nicht getan wurde.
$H_2(Antwort')$ beinhaltet die durch S errechnete Prüfsumme über die, möglicherweise vorher einer Transformation unterzogene, Antwortnachricht und Abh_2 sind genau die Abhängigkeiten, dessen Festhaltung sich S in (4) gewünscht hat. Die Signatur von S sichert Z den Beweis, dass die Nachricht tatsächlich vom Service geschickt worden ist

Mit der Nachricht (5b) bestätigt K den Erhalt der Nachricht aus (4).
Der aufmerksame Leser wird feststellen, das genau diese Stelle einen wichtigen Angriffspunkt für K darstellt. Übermittelt er (5b) nicht, so erhält S keinen Nachweis über den Empfang von (4) durch K. Dieser mögliche Angriff wird durch das Protokoll (es kann bewiesen werde, dass eine Anfrage von K an S geschickt wurde, sowie in Zusammenarbeit von Z mit S kann sogar deren konkreter Inhalt nachgewiesen werden) abgefangen. Die konkrete Behandlung wird in Kapitel 8.3.14 näher vorgestellt.
Die Nachricht (5b) enthält exakt die selben Parameter, wie oben bei (5a) beschrieben.
Die Signatur von K sichert auch hier für Z den Beweis, dass diese Nachricht auch tatsächlich vom Klienten geschickt worden ist.
Hat Z sowohl (5a) als auch (5b) erhalten, ordnet er sie anhand von Ktx und der Anfragekennung (z.B. $H_1(Anfrage')$) einander zu und prüft die übermittelten Prüfsummen $H_2(Antwort')$ sowie die angegeben Abhängigkeiten Abh_2 auf Identität.
Die optionale Verschlüsselung des Nachrichtenkopfes ist auch hier wieder nicht zwingend notwendig, aber auf Grund möglicher zu entnehmender Verkettungsinformationen durch Außenstehende zu empfehlen.

8.3.7.7 Bestätigung/Fehlermeldung für die Antwort vom Zeugen - 6a/6b

Bei einem positiven Abgleich der Nachrichten aus (5a) und (5b) übermittelt der Zeuge eine "OK"-Nachricht sowohl an K als auch an S. K erhält dadurch einen Nachweis über den Inhalt der Antwort (zusammen mit Z bei späteren Ansprüchen nachweisbar) und S erhält einen Nachweis, dass seine Antwort auch tatsächlich beim Klienten angekommen ist. Weiterhin ist es nun Aufgabe des Zeugen, den im zugehörigen Kommunikationsschritt (2a) und (2b) mit Hilfe der Parameter Cnt_{Inc} und Cnt_{Dec} abgemachten und übereinstimmenden Preis auf das jeweilige Konto anzurechnen bzw. abzuziehen. Wäre dies aus irgend einem

Grund nicht möglich (z.B. Timeout), so würde der Zeuge auch dies mit einer Fehlermeldung quittieren und eine entsprechende Fehlerbehandlung initiieren.
Eine "OK"-Nachricht enthält dabei wieder für die Zuordnung zur jeweiligen Antwortnachricht bei K bzw. S die Kontextinformation Ktx und die eindeutige Antwortkennung $H_2(Antwort')$. $H_1(Anfrage')$ muss zusammen mit $H_2(Antwort')$ innerhalb des selben Signaturblocks von Z signiert werden, um den Zusammenhang zwischen beiden für K und S, durch Z bezeugt, nachweisbar zu machen.
Der enthaltene Parameter Abh_2 gilt für S und K als Beweis, dass sie eine Übereinkunft für Abhängigkeiten zu älteren Nachrichten getroffen haben und Z dies bestätigen kann.
Die Signatur von Z um alle Parameter dient K und S als Beweis, dass alle Prüfungen durch Z positiv verlaufen sind. Der Zeuge ist somit nicht mehr in der Lage später zu behaupten, dass die Prüfung negative verlief oder erst gar nicht stattgefunden hätte.
Das auch hier wieder innerhalb der selben Signatur enthaltenen Flag "OK" stellt sicher, dass eine Verwechslung mit einer eventuellen Fehlermeldung ausgeschlossen werden kann.

Schlug eine der nach (5a) bzw. (5b) durchgeführten Prüfungen fehl oder kam eine der beiden Nachrichten gar nicht erst an, so wird dies vom Zeugen wieder mit einer Fehlermeldung quittiert. Wie diese Fehlermeldung und die daraus resultierende Fehlerbehandlung für die einzelnen Fälle konkret aussieht, wird ebenfalls in Kap. 8.3.14 näher erläutert.
Die Fehlermeldung enthält auch hier wieder Ktx und $H_2(Antwort')$, um eine Zuordnung zum zugehörigen Kommunikationsvorgang für K und S zu ermöglichen.
Über den Parameter $Fehler_2$ wird den Kommunikationsteilnehmern auch hier eine detaillierte Beschreibung des aufgetretenen Problems übermittelt und eventuell eine Fehlerbehandlung vorgeschlagen.
Die umfassende Signatur von Z stellt sicher, dass Z nicht in der Lage ist, die negative Prüfung oder das Stattfinden des Prüfungsvorgangs zu leugnen.
Für die optionale Verschlüsselung des Nachrichtenkopfes gilt wieder ähnliches, wie für vorangegangene Beispiele (Verkettungsinformationen).

8.3.8 Überprüfung auf Einhaltung der Protokollanforderungen

In Kapitel 8.3.6 wurden verschiedene Anforderungen an die konkrete Umsetzung des Protokolls aufgelistet. In diesem Kapitel soll nun gezeigt werden, wie diesen Anforderungen in der Detailbeschreibung (s.Kap. 8.3.7) unter Verwendung unterschiedlicher Hilfsmittel genüge getan wurde.

PA_1: $Kontext_Z$ ist für jeden Teilnehmer verfügbar. Dieser besitzt eine feste Verbindung mit SI_Z, SI_K und Lok_Z, wodurch die Zurechenbarkeit zum Klient K über Z gegeben ist.

PA_2: Die Signatur von $Kontext_Z$ ist anhand einer Überprüfung mit Hilfe der enthaltenen Information SI_Z nachweislich von Z erstellt worden.

PA_3: Der Zeuge erhält den Hash der Anfrage in (2a) und (2b) in einer Signatureinheit mit $NONCE_2$. Nur der "richtige" Service ist in der Lage, $NONCE_2$ aus dem verschlüsselten Block in (1) zu bestimmen.

Sind sowohl die Hashs als auch die $NONCE_2$-Werte identisch, so beweist dies, dass die Anfrage unverändert am "richtigen" Service S angekommen ist. Dies bezeugt der Zeuge durch das Mitsenden des Hashs in seiner signierten (3b-OK) Antwortnachricht.

PA_4: Da es als kryptografisch sehr schwer zu erachten ist, zu einem vorgegebenen Hash eine Ursprungsnachricht zu finden, welche erstens die Struktur einer korrekten SOAP-Nachricht besitzt und zweitens einen sinnvollen Betrugsversuch ermöglicht, gilt das Vorzeigen der signierten Nachricht aus (3b) und einer Nachricht die den enthaltenen Hash erzeugt, als Beweis, dass eben diese Nachricht vom Service erhalten wurde.

PA_5: Der Zeuge überprüft die in (2a) und (2b) enthaltenen Kosten- (Cnt_{Dec}) und Gutschriftenveranschlagungen (Cnt_{Inc}) auf Gleichheit (im Sinne von Nennwertgleichheit, Währungsequivalenz etc.) und bestätigt die Übereinkunft durch Senden von $CntInfo_K$ (enthält Cnt_{Dec}) in seiner signierten (3b-OK) Nachricht.

PA_6: Die in (1) unter Verwendung von Abh_1 angegebenen Abhängigkeiten (s.Kap. 8.3.9) werden von S, falls er diese bestätigen kann, in (2b) an den Zeugen geschickt. Der Zeuge prüft die Identität der Abhängigkeitswünsche von K und S und bezeugt eine Übereinstimmung durch Einfügen von Abh_1 in seine signierte (3b-OK) Nachricht. Sowohl K als auch S können die Abhängigkeiten nun zusammen mit Z nachweisen.

PA_7: Der von S signierte Hash $H_2(Antwort')$ in Kommunikationsschritt (5a) wird mit dem durch den Klienten übermittelten Hash aus (5b) verglichen. Sowohl (5a) als auch (5b) sind von S bzw. K signiert und lassen dich dadurch aus dem Kontext heraus eindeutig dem korrekten Klient K und dem korrekten Service S zuweisen. Mit der Signatur von Z unter der "OK"-Nachricht aus (6a) bzw. (6b) bezeugt Z, dass die Prüfsummen aus beiden Nachrichten übereingestimmt haben. Damit ist die Herkunft der Antwort von S für K über die Nachricht (6b) von Z (muss die Nachrichten aus (5a) und (5b) archivieren) beweisbar.

PA_8: Durch die von Z signierte Identität der beiden Hashsummen ($H_2(Antwort)$) aus (5a) und (5b) in (6b) kann der Klient zusammen mit Z den Inhalt der Antwort nachweisen. Er selbst besitzt die Nachricht und muss nur noch zeigen, dass die Prüfsumme identisch derjenigen ist, welche Z bezeugt hat.

PA_9: Die in (4) unter Verwendung von Abh_2 angegebenen Abhängigkeiten (s.Kap. 8.3.9) werden von K, falls er diese bestätigen kann, in (5b) an den Zeugen geschickt. Der Zeuge prüft die Identität der Abhängigkeitswünsche von K und S und bezeugt eine Übereinstimmung durch Einfügen von Abh_2 in seine signierte (6b-OK) Nachricht.
Der besonders wichtige Fall der Abhängigkeit zur zugehörigen Anfrage wird in der Protokollbeschreibung durch Einfügen von $H_1(Anfrage')$ in (6a) bzw. (6b), falls sie in (5a) und (5b) übereinstimmten, abgedeckt (die Möglichkeit der Verschiebung von $H_1(Anfrage')$ in Abh_2 wurde bei der detaillierten Beschreibung in Kapitel 8.3.7 bereits erwähnt.).
Die Signatur von Z garantiert die Beweisbarkeit dieses Sachverhalts sowohl für K als auch S in Zusammenarbeit mit Z.

PA_{10}: Der von K signierte Hash $H_1(Anfrage')$ in Kommunikationsschritt (2b) wird mit dem durch den Service übermittelten Hash aus (2a) verglichen. Sowohl (2a) als auch (2b) sind von S bzw. K signiert und lassen dich dadurch aus dem Kontext heraus eindeutig dem korrekten Klient K und dem korrekten Service S (unter zusätzlicher Verwendung

von $NONCE_2$) zuweisen. Mit der Signatur von Z unter der "OK"-Nachricht aus (3a) bzw. (3b) bezeugt Z, dass die Prüfsummen aus beiden Nachrichten übereingestimmt haben. Damit ist die Herkunft der Anfrage von K für S über die Nachricht (3a) von Z (muss die Nachrichten aus (2a) und (2b) archivieren) beweisbar.

PA_{11}: Durch die von Z signierte Identität der beiden Hashsummen ($H_2(Antwort)$) aus (2a) und (2b) in (3a) kann der Service zusammen mit Z den Inhalt der Anfrage nachweisen. Er selbst besitzt die Nachricht und muss nur noch zeigen, dass die Prüfsumme identisch derjenigen ist, welche Z bezeugt hat.

PA_{12}: Der Zeuge überprüft die in (2a) und (2b) enthaltenen Kosten- (Cnt_{Dec}) und Gutschriftenveranschlagungen (Cnt_{Inc}) auf Gleichheit (im Sinne von Nennwertgleichheit, Währungsequivalenz etc.) und bestätigt die Übereinkunft durch Senden von $CntInfo_S$ (enthält Cnt_{Inc}) in seiner signierten (3a-OK) Nachricht.

PA_{13}: Der Zeuge sichert dem Service mit seiner signierten Nachricht (3a) und dem enthaltenen Wert $CntInfo_S$, neben der Tatsache, dass sich S und K über den Preis einig sind, die Zahlungsfähigkeit von K zu (Zahlungswert muss durch den Zeugen als "bis auf Weiteres reserviert" gekennzeichnet werden um einer eventuellen "Überbuchung" vorzubeugen).

PA_{14}: Auf Grund der Tatsache, dass nur K dazu in der Lage ist, die Signatur für (5b) zu erstellen (Z ist seit (0a) im Besitz von SI_K) und Z die Identität von $H_2(Antwort')$ aus (5a) und (5b) mit (6a) bestätigt, liefert S den Beweis (über Z), dass K die Antwort erhalten haben muss.

PA_{15}: Durch die von Z signierte Identität der beiden Hashsummen ($H_2(Antwort)$) aus (5a) und (5b) in (6a) kann der Service zusammen mit Z den Inhalt der Antwort nachweisen. Er selbst besitzt die Nachricht und muss nur noch zeigen, dass die Prüfsumme identisch derjenigen ist, welche Z bezeugt hat.

PA_{16}: Mit der "OK"-Nachricht aus (3a) hat der Zeuge dem Service zugesichert, dass der Klient über einen genügend hohen "Kontostand" verfügt, um für die Dienstleistung bezahlen zu können und hat diesen Betrag zudem reserviert.
Die "OK"-Nachricht aus (6a) liefert dem Service erstens den Beweis, dass die Antwort nachweislich korrekt beim Klienten angekommen ist und dass zweitens der durch Cnt_{Inc} und Cnt_{Dec} ausgehandelte Betrag korrekt vom Klienten- zum Servicekonto transferiert worden ist ($CntInfo_x$).

PA_{17}: Die Abhängigkeit der Antwort zur zugehörigen Anfrage wird durch die gemeinsame Signatur von $H_1(Anfrage')$ und $H_2(Antwort)$ in der "OK"-Nachricht (6a) durch Z festgehalten. Diese liegt in einer Gleichheitsprüfung der genannten Werte aus (5a) und (5b) begründet und kann von Z anhand dieser signierten Nachrichten nachgewiesen werden.

PA_{18}: Der Klient wird anhand des erstellten Kontextes (enthaltene Information SI_K aus (0a)), welcher permanenter Bestandteil aller ausgetauschten Nachrichten bleibt, identifiziert.
Der Service wird nach korrekter Übermittlung von SI_S in Zusammenhang mit dem aus (1) entnommenen $NONCE_2$-Wert in (2a) identifiziert und der entsprechenden Sitzung sowie dem entsprechenden Vorgang zugeordnet.

PA_{19}: Auf Grund des in jeder Nachricht übermittelten Kontextes $Kontext_Z$ bzw. Ktx erfolgt eine eindeutige Zuordnung jeder Nachricht zum zugehörigen durch Z intern verwalteten Vorgang.
Die Zuordnung zur entsprechenden Entität in einem Kommunikationsvorgang erfolgt anhand der übermittelten SI_x-Informationen, anhand einer zum jeweiligen Kommunikationspartner etablierten Session ($Shr_{X,Z}$) oder anhand der jeweils vorliegenden Signatur. Zur korrekten Identifikation des zur Signaturtestung bzw. zum Entschlüsseln zu verwendenden Schlüssels muss eine entsprechende Information (z.B. "Schlüssel von <KlientID> und Kontext <KontextID>") in den signierten Teil jeder Nachricht eingebaut werden. Dies ist in Abbildung 19 auf Seite 84 nicht bei jedem Schritt direkt ersichtlich und wurde nur der Übersichtlichkeit halber weggelassen.

PA_{20}: Voraussetzung für die signierten Nachrichten (3a) und (3b) bzw. (6a) und (6b) sind die Nachrichten (2a) und (2b) bzw. (5a) und (5b). Diese Nachrichten sind durch K bzw. S signiert und entsprechend einander zuordenbar ((2x)[47]: $NONCE_2$; (5x): $H_1(Anfrage')$).
Treten Streitigkeiten auf, muss Z die archivierten Nachrichten aus (2x) und (5x) nur vorzeigen und kann so den legitimen Versandt von (3x) und (6x) beweisen.
Den Beweis für die korrekte Verwaltung der Kontostände liefert Z durch verschicken der Kontoinformation $CntInfo_x$ in (3x) und (6x). In diesem Kontoinformationen werden in (3x), zusätzlich zum aktuellen Kontostand und aktiven Reservierungen zu Gunsten oder zu Ungunsten des Teilnehmers, die Informationen $CntInc$ bzw. Cnt_{Dec} aus (2a) bzw. (2b), als Nachweis für deren Übereinstimmung, mitgesendet.
Jeder Teilnehmer kann so, beweisbar durch die Signatur von Z, jederzeit nachweisen, wie sich sein eigener Kontostand entwickelt hat. Treten bei einer Bezahlung Probleme auf (z.B. der Zeuge hat nicht korrekt reserviert und ein Klient verfügt "plötzlich" nicht mehr über einen ausreichenden Kontostand) kann anhand, des mitgeloggten Verlaufs von $CntInfo$ erstens die Schuld des Zeugen gezeigt und zweitens nachgewiesen werden, dass z.B. für eine Dienstleistung nicht korrekt (in Höhe des vereinbarten Betrags aus (2a) und (2b)) bezahlt worden ist. Der Service ist somit in der Lage, seine Forderungen gegen den Klienten bzw. gegen den Zeugen vor einem Gericht einzuklagen.

8.3.9 Abhängigkeiten zu anderen Nachrichten

Abhängigkeiten (in der Protokollbeschreibung mit *Abh* bezeichnet) kennzeichnen die Zugehörigkeit von früher verschickten Nachrichten zur aktuell übermittelten und durch Z zu bezeugenden Nachricht. Diese werden verwendet, um anzuzeigen, dass der Inhalt der gerade gesendeten Nachricht immer im Kontext der in *Abh* angegebenen weiteren Nachrichten zu betrachten ist.
Ein Beispiel für solch einen Fall wäre ein Ticket-Verkaufssystem, welches auf einer Reservierungsanfrage hin ein Preisangebot schickt, auf dessen Grundlage dann der Kauf erfolgen soll. Die Reservierungsanfrage ist eine kostenlose Dienstleistung und muss daher nicht zwingend über den Zeugen ablaufen. Man stelle sich nun vor, der Kaufvorgang (kostenpflichtig) wird anschließend durch "<ReservierungsID>; Kaufen!" ausgelöst. Dies funktioniert natürlich nur dann, wenn der Klient sich sicher sein kann, den vorher ausgemachten Kaufpreis auch wirklich zu erhalten. Dies stellt er sicher, indem er im Zählsystem in Schritt (1) eine Abhängigkeit (in Abstimmung mit S) zur Reservierungsanfrage und -antwort herstellt,

[47] "(ix)" soll in diesem Zusammenhang für "(ia) und (ib)" stehen (i = Kommunikationsschritt).

aus denen ja der ausgemachte Preis ersichtlich ist. Verweigert der Service S die Zusage *Abh*, so verzichtet der Klient kurzerhand auf die Verwendung des kostenpflichtigen Services "Buchung".
Die Herstellung der Abhängigkeit kann auf zwei Arten erfolgen:

8.3.9.1 Abhängigkeiten per Prüfsummenliste

In diesem Fall einigen sich Service und Klient per KSE_{Anf} bzw. KSE_{Ant} darauf, dass sie die Prüfsummen über die Anfrage bzw. Antwort für eine spätere Verwendung zwischenspeichern. Der große Vorteil dabei liegt in der Einsparung der sonst notwendigen Kommunikationen mit Z. Es können beliebig viele kostenlose Operationen ohne Z ausgeführt werden, bevor in deren Kontext eine kostenpflichtige Operation ausgeführt wird.
Für das oben genannten Beispiel bedeutet dies, dass sich Service und Klient $H(Reservierungsanfrage)$ und $H(Reservierungsantwort)$ merken (per KSE vereinbart) und diese auf "Wunsch" des Klients als *Abh* in die Buchungsanfragebestätigung durch Z (oder in anderen Szenarien auch auf "Wunsch" des Services in seine Antwortbestätigung denkbar) mit einbauen.
Bei Streitigkeiten kann der Klient dann zusammen mit Z und der von ihm signierten *Abh*-Liste aus (3b) nachweisen, in welchem Kontext die Buchung geschah.

8.3.9.2 Abhängigkeiten per Referenzierung beim Zeugen

Gesetzt den Fall, es gibt keinen Grund auf die zusätzlichen Hash-Übermittlungen an Z zu verzichten, könnte man auch jede kostenlose Kommunikation unter der Einstellung $Cnt_{Inc} = Cnt_{Dec} = 0$ vom Zeugen bestätigen lassen.
Die Reservierungsantwort beim obigen Beispiel würde somit auf die Reservierungsanfrage (per sowieso vorhandenem $H_1(Antwort')$ in (6x)) und die Buchungsanfrage auf die Reservierungsantwort (per $H_2(Reservierungsantwort')$ bzw. einer anderweitig durch Z vergebenen $AntwortID$ in der *Abh*-Liste von (1)) usw. verweisen.
Der Zeuge würde somit intern einen zum jeweiligen Vorgang (=Kontext) zugeordneten Referenzierungsbaum verwalten, der für das Beispiel, wie in Abbildung 20 dargestellt aussehen würde.
Der Vorteil dabei liegt darin, dass sich K und S nur die ID der jeweils vorherigen Kommunikation untereinander merken müssten und der Verwaltungsaufwand zu Z verschoben würde. Dies führt zu einer Vereinfachung (Einsparung je nach Komplexität des Anwendungsfalls) der Implementation von K und S, wohingegen Z diese Art der Referenzierung ohnehin unterstützen muss und somit hier keine Verkomplizierung auftreten würde.
Bei auftretenden Streitigkeiten könnten somit K oder S mit Hilfe von Z den Inhalt aller vorausgegangenen Nachrichten als Voraussetzung für eine kostenpflichtige Kommunikation nachweisen.

8.3.10 Lebenszyklus eines Kontext-Objektes

Durch die Initiierung eines Kommunikationsvorgangs durch den Klienten in (0a) legt der Zeuge einen internen Vorgang (Kontext) zur Verwaltung aller Kommunikationen an, welche vom Klienten aus zu Services geführt werden, die eine Zugriffszählung durchführen möchten.

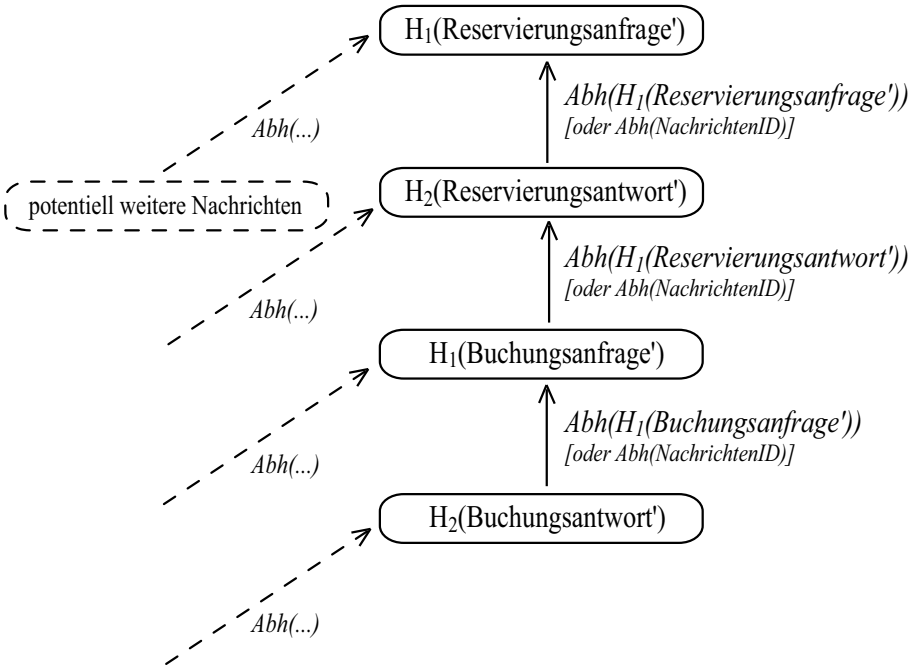


Abbildung 20: Vom Zeugen verwalteter Referenzierungsbaum für das Beispiel "Ticket-Verkaufssystem"

Dabei wird die Sicherheitsinformation SI_K fest in den Kontext integriert. Sämtliche Signaturen von Bestätigungsnachrichten durch K ((2b) bzw. (5b)) müssen anschließend eine Überprüfung mit SI_K positiv überstehen, um akzeptiert zu werden.
Jeder angelegte Kontext besitzt nur eine begrenzte Gültigkeitsdauer, welche sich durchaus über längere Zeiträume (Tage, Wochen, Monate ...) hinziehen kann. Diese Information wird zusammen mit der eindeutigen $KontextID$, dem Erstellungszeitpunkt, den Kommunikationseigenschaften KE, den Informationen über K (SI_K), sowie den Informationen über den Zeugen (SI_Z, Lok_Z) in einen transportablen, durch Z signierten Token integriert. Dieser Token wird später durch alle an einem Vorgang teilnehmenden Kommunikationspartner referenziert, um eine Zählung mit Hilfe des Zeugen durchführen zu können.
Der Klient muss seine eigenen Kontexte und der Zeuge seine verwalteten Kontexte auch vor Ablauf deren Gültigkeitszeitraumes zerstören können. Dies kann z.B. nötig werden, wenn bestimmte Sicherheitsinformationen bekannt werden, welche zu einem Brechen eines der verwendeten kryptografischen Teile und damit zu einer Beeinträchtigung des Protkollablaufs führen kann (z.B. Signatur von K oder Z). Der Klient speziell muss natürlich auch in der Lage seine, eigene Kontexte, ohne Angabe von Gründen, jederzeit zu zerstören.
Bei der Invalidierung von Kontexten muss allerdings darauf geachtet werden, dass sich der jeweilige Kontext in einem Zustand befindet, der dies zulässt. So könnte diese Funktionalität vom Klienten auch verwendet werden, um z.B. nach Erhalt von (4) den Kontext zu zerstören, um sich der Bezahlung zu entziehen. Eine Zerstörung des Kontext durch den Klienten ohne wichtige Gründe sollte allgemein dann verboten werden, wenn davon ausgegangen werden kann, dass der Service seine Dienstleistung gerade erbringt, und eine Abbruch in diesem Augenblick eine Bezahlung verhindern würde (konkret zwischen (3a) und (6a)).
Bei einem wichtigen Grund (z.B. Brechen der Signatur von K) muss dies allerdings trotz-

dem sofort möglich sein, da sonst der Klient in Mitleidenschaft gezogen werden kann. Für diesen Fall muss sich eine entsprechende Behandlung konkret überlegt werden (Beweise, dass eine Anfrage von K geschickt wurde, sind unter Umständen trotzdem vorhanden).

8.3.11 Zugriffszählung durch den Zeugen

Erhält der Zeuge aus (2a) und (2b) die Informationen Cnt_{Inc} und Cnt_{Dec}, so prüft er diese vor dem Verschicken von (3a) bzw. (3b) auf Gleichheit (in Bezug auf Nennwert, Währung etc.) und überprüft, ob K über einen ausreichend hohen Kontostand verfügt, um den Service in Anspruch nehmen zu können. Ist dies der Fall so zieht er den Betrag X von dessen Konto temporär ab und überträgt ihn in einen Reservierungszustand. Damit wird verhindert, dass später der Fall eintreten kann, dass K zwar vor (3x) in der Lage war zu zahlen, aber auf Grund zwischenzeitlicher Nutzung anderer kostenpflichtiger Kommunikationsvorgänge nach (5x) plötzlich nicht mehr in der Lage ist die Zahlung zu leisten.
K und S erhalten durch $CntInfo_K$ bzw $CntInfo_S$ in (3a) bzw (3b) eine Bestätigung von Z, dass eine Übereinkunft in Bezug auf die Höhe der zu leistenden Zahlung einvernehmlich getroffen wurde. Um es S bei auftretenden Problemen (z.B. einer Zusammenarbeit von K und Z) zu ermöglichen eine Beweis anzutreten, wie sein aktueller Kontostand bei Z aussehen muss, fügt Z in $CntInfo_x$ zusätzlich die Information ein, wie der aktuelle Kontostand von x aussieht und welche positiven/negativen Reservierungen aus Sicht von X durchgeführt wurden.
Die Integration von $CntInfo_x$ sowohl in (3x) als auch (6x) sichert dem Teilnehmer einen lückenlosen Nachweis ausstehender Zahlungen bzw. über Reservierungsfehler seitens des Zeugen.
Arbeitet Z mit K zusammen, um es K zu ermöglichen weniger zu zahlen, so erkennt dies S anhand eines Vergleichs seiner eigenen Daten mit den Daten aus $CntInfo_S$ in (3a) und kann die Berechnung verweigern, beziehungsweise anderweitige Maßnahmen zur Korrektur einleiten. Z ist darauf hin nicht in der Lage zu behaupten "S habe in (2a) einen anderen (niedrigeren) Betrag genannt", da er die Signatur von S für die Nachricht (2a) nicht fälschen und einen entsprechenden Beweis nicht vorweisen kann.
Arbeiten Z und S zusammen, um vom Konto des Klienten mehr abzuheben als abgemacht, so kann K dies, analog zum gerade genannten Szenario, in dem Z und K zusammen versuchten zu betrügen, beweisen.

8.3.12 Aushandeln der Protokollparameter

Der in Schritt (0a) übermittelte Parameter KE, welcher daraufhin von Z fest in den transportablen Kontext $Kontext_Z$ integriert wird, beinhaltet verschiedene Festlegungen über Eckdaten des Protokolls. Dabei muss sich K natürlich darüber bewusst sein, dass diese sowohl von ihm selbst, als auch von den aufzurufenden Service unterstützt werden müssen. Dazu ist es notwendig, bereits vor der Erstellung des Kontextes durch Z zu prüfen, ob die gewünschten Services mit diesen Festlegungen einverstanden sind. Dies kann z.B. durch die Einsicht in die Metabeschreibung des Zielservices erreicht werden.
Die wichtigste Festlegung ist dabei der zu verwendenden Hashalgorithmus in (2x) bzw. (5x). Eine fehlende Übereinstimmung hier würde zu einem latenten Fehler bei der Prüfung durch Z führen, welcher nicht sofort erkannt werden könnte und damit das Protokoll behindern würde.
Eine Absprache zwischen K und S vor (2x) bzw. (5x) ist nicht vorgesehen und würde zu unnötigen zusätzlichen Kommunikationsschritten führen.

Weiterhin könnte bei einer Protokollimplementation mit mehreren Modi der zu verwendende Modus in KE festgelegt werden. Vorstellbare Modi wären:

- Erzwingen der Verschlüsselung von Kommunikationsschritten, welche durch das Protokoll als optional festgelegt wurden (z.B. (3x), (5x) oder (6x)).
- Wie werden Abhängigkeiten grundsätzlich dargestellt (s.Kap. 8.3.9)?
- Welche Informationen soll $CntInfo_X$ enthalten?

8.3.13 Diskussion über mögliche Aufwandseinsparungen am Protokoll

Betrachtet man die detaillierte Darstellung des Protokolls in Abbildung 19 auf Seite 84 genauer, so fällt auf, dass immer der gesamte Nachrichtenkopf signiert bzw. verschlüsselt wurde. Dies wurde dort und für die Implementation des Prototypen so angenommen, um die Übersichtlichkeit möglichst hoch und die Komplexität der kryptografischen Operationen möglichst gering zu halten.
Bereits in der detaillierten Beschreibung in Kapitel 8.3.7 wurde darauf hingewiesen, das bestimmte Kommunikationsschritte nicht zwingend verschlüsselt werden müssen (ausgegraute Darstellung), wenngleich es empfohlen und in der prototypischen Implementation auch so umgesetzt wird.

In Schritt (0a) ist es nicht nötig $Shr_{K,Z}$ zu signieren, da die asymmetrische Verschlüsselung sicherstellt, dass nur Z diesen Parameter entschlüsseln kann.
KE muss nur signiert und nicht verschlüsselt werden, da ein Bekanntwerden der Kommunikationseigenschaften für einen Vorgang keine Einschränkungen für das Protokoll mit sich bringt.
Der Wert $NONCE_1$ soll nur sicherstellen, dass ein klassischer Wiederholungsangriff an dieser Stelle nicht möglich ist. Dazu genügt es, ihn entweder verschlüsselt oder im Signaturblock zu übermitteln. Es muss allerdings durch die Blockung eine eindeutige Verbindung zu den anderen Parameter hergestellt werden, da ein Angreifer sich sonst für einen Wiederholungsangriff eine Nachricht nach dem Baukastenprinzip aus anderen Nachrichten von K zusammenpuzzeln könnte.

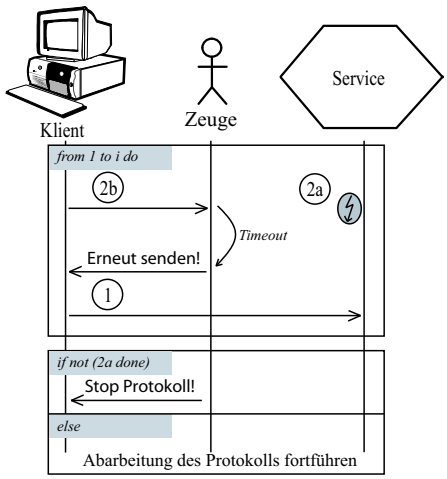
In Schritt (1) könnten $Shr_{K,S}$ nur verschlüsselt und unsigniert übertragen werden, da die asymmetrische Verschlüsselung sicherstellt, dass nur S diesen symmetrischen Schlüssel entziffern kann.
Für $NONCE_2$ gilt dasselbe, da er für S von K signiert keinerlei Beweiskraft besitzt und nur als Nachrichtenauthentifikator über den Zwischenschritt $Zeuge$ eingesetzt wird.
KSE_{Anf} und Abh_1 sind Parameter, welche zwar von K signiert werden (Beweis für S, dass er sich an das Protokoll gehalten hat), aber nicht verschlüsselt werden müssen. Ein Bekanntwerden der Kommunikationsschritteigenschaften und der Abhängigkeiten führt zu keiner Einschränkung des Protkolls.

Neben diesen möglichen Einsparungen wurde bereits in der detaillierten Beschreibung darauf hingewiesen, dass der Kontext ($Kontext_Z$) ab Schritt 2a bzw. 2b oder bei einem erneuten Durchlauf mit dem selben Service dahingehend reduzierbar ist, dass jeder Kommunikationsteilnehmer anhand der übermittelten Daten nur noch in der Lage sein muss, den gewünschten und bereits vorhandenen $Kontext_Z$ in seinen verwalteten Daten zu lokalisieren. Denkbar dafür wäre z.B. nur die eindeutige $KontextID$, eine Prüfsumme oder nur die Signatur-bits (ohne den signierten Inhalt).

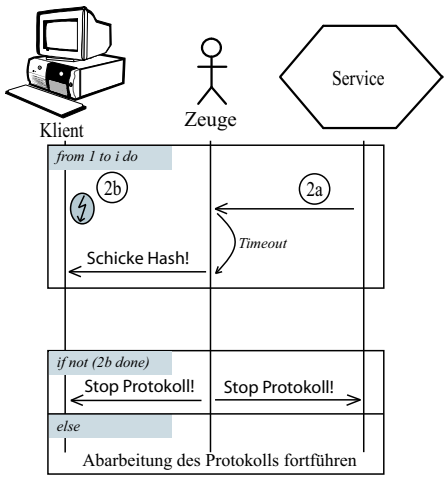
Auch die möglicherweise sehr umfangreiche Sicherheitsinformation SI_x (z.B. komplette Binärdaten eines Zertifikats) können, falls bereits beim Empfänger aus einer vorherigen Kommunikationsbeziehung bekannt, auf Referenzierungsinformationen reduziert werden (z.B. der String "Öffentliches Zertifikat vom Klienten <Klientenkennung>").

8.3.14 Störungs-, Fehler- und Angriffsbehandlung

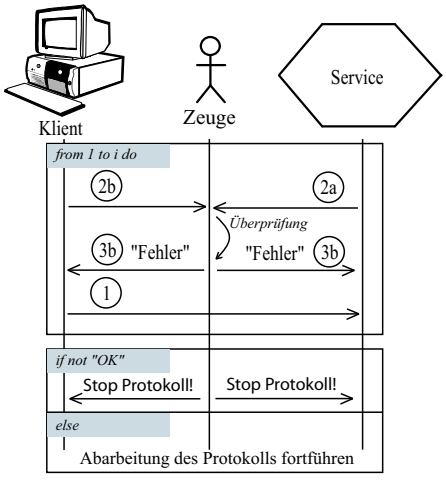
1. Was passiert, wenn zwei Parteien gemeinsam versuchen die dritte zu betrügen? Wie kann das verhindert werden?
2. Wie wird bei einem plötzlichen Abbruch durch einen Benutzer oder bei einem Netzwerkfehler verfahren?



Fall 1: Service schickt keinen Hash über die Anfrage



Fall 2: Klient schickt keinen Hash über die Anfrage



Fall 3: Abgleich der Hash-Werte schlägt fehl

Abbildung 21: Fehlerbehandlung, bei der Anfragebestätigung

8.4 Implementation

Untersucht man das in Kapitel 8.3 hergeleitete und beschriebene Protokoll zur Umsetzung eines Zählers für Web Service-Zugriffe auf grundlegende Strukturen, welche bereits in

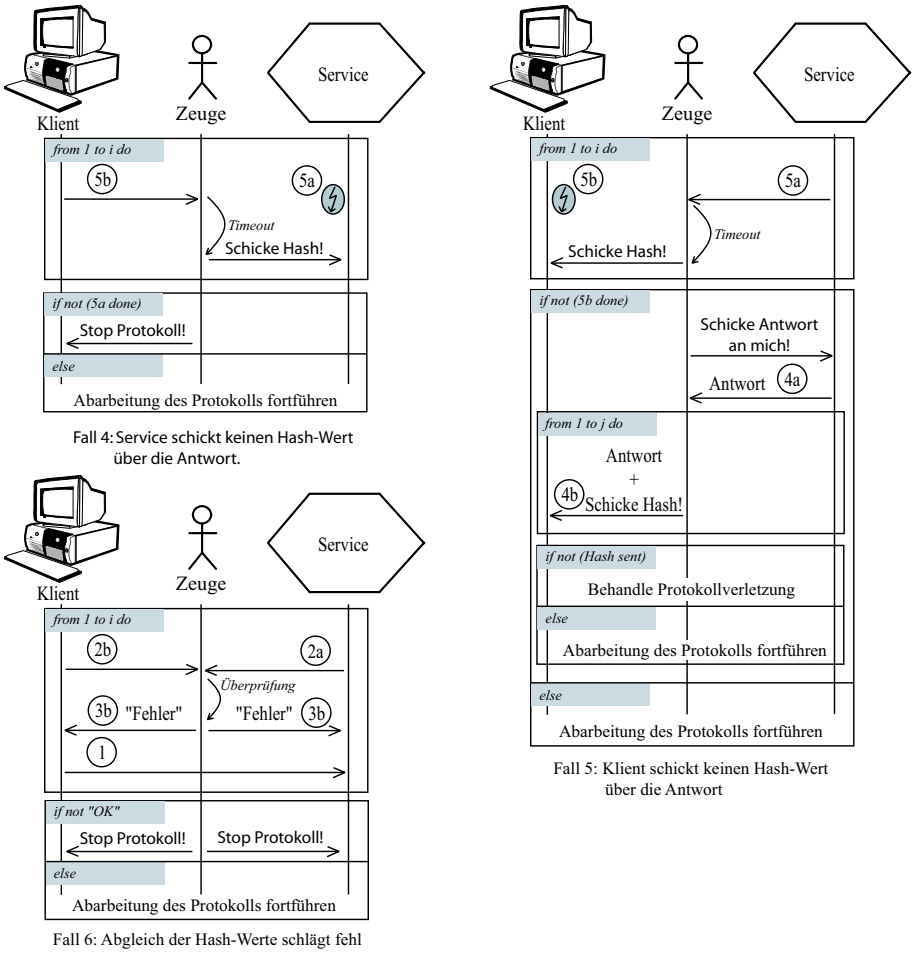


Abbildung 22: Fehlerbehandlung, bei der Antwortbestätigung

existierenden Spezifikationen umgesetzt wurden, so können eindeutige Parallelen zur WS-SecureConversation- (s.Kap. 6.7) und WS-Coordination-Spezifikation (s.Kap. 3.6) erkannt werden.
Die WS-Coordination-Spezifikation spielt im Prototyp eine wichtige Rolle, da der Zeuge die Aufgaben eines Koordinators übernimmt. Im Schritt 0a fordert der Klient den Zeugen auf, einen neuen Koordinationskontext ($Kontext_Z$) zu erstellen, welcher dann über die Dauer des Protokolls die Zuordenbarkeit von Nachrichten beim Zeugen, und implementationsabhängig auch beim Klient und Server, zu einem Vorgang garantiert. Im weiteren Verlauf des Protokolls hängt das Voranschreiten im wesentlichen von den Hash-Vergleichen und den daraus resultierenden "OK"- bzw. "Fehler"-Meldungen des Zeugen ab.

Im Folgenden wird von drei unterschiedlichen Typen von *Kontexten* gesprochen:

- dem *Koordinationskontext*, welcher den globalen Kontext zur Koordination aller möglichen Kommunikation zwischen dem Klienten, welcher den Kontext initiiert hat, beliebig vielen Services und dem koordinierenden Zeugen beschreibt,

- dem *Kommunikationskontext*, welcher sich auf den *einmaligen* Durchlauf der Kommunikationsschritte 1 bis 6 (s.Abb. 19 S. 84) bezieht,

- und dem *Zählkontext*, welcher als Vereinigung eines konkreten Koordinations- und eines konkreten Kommunikationskontextes verstanden wird.

8.4.1 Verwendete API's

Der vorliegende Prototyp wurde unter Verwendung des Java Standard Development Kit Version 1.5.0_02 entwickelt. Folgende zusätzlichen API's kamen zum Einsatz:

Log4J Mit Hilfe dieser API [Log4J] (verwendete Version: 1.2.9) werden Logging-Nachrichten mit einfach festlegbarer Wichtigkeit an leicht konfigurierbaren Positionen (Dateien, Konsolen, extern Tools etc.) ausgegeben. Dadurch wird das Nachvollziehen von Abläufen und das Auffinden von Problemen bei der Entwicklung und zur Laufzeit erleichtert.

Axis Dieses Projekt [Axis] der Apache Foundation [Apache] in der verwendeten Version 1.2RC2 wurde als grundlegende API zur Erzeugung von SOAP-Nachrichten und deren Versand eingesetzt. Der strukturierte Aufbau der Nachrichtenverarbeitung durch Handler-Ketten und deren leicht konfigurierbarer sequentieller Aufbau führten zu einer übersichtlichen modularen Struktur des Prototyps (z.B. s.Abb. 35 im Anhang).

Tomcat Der Tomcat Servlet Container [Tomcat] wurde in der Version 5.5.3 verwendet und dient als Laufzeitumgebung für die verwendete Webserviceimplementation *Axis*.

XSS4J Die XML Security Suite [XSS4J] (verwendeter Build: 29.07.2004) der IBM AlphaWorks [AlphaWorks] wird in diesem Projekt für die Erzeugung von enveloped bzw. enveloping Signaturen (s.Kap. 6.1) sowie für die Verschlüsselung eingesetzt. WS-Security (s.Kap. 6.4) wird von diesem Projekt nicht unterstützt.

TSIK Das Trust Services Integration Kit [TSIK] von Verisign [Verisign] unterstützt Signaturen und Verschlüsselung im WS-Security-Standard und kommt überall dort zum Einsatz, wo der Nachrichtenkörper (Body) einer SOAP-Nachricht signiert bzw. verschlüsselt werden soll. Ein selektives signieren/verschlüsseln ist allerdings in der verwendeten Version 1.10 nicht möglich. An diesen Stellen wird weiterhin auf XSS4J zurückgegriffen.

8.4.2 Paket-Struktur

Alle folgenden Angaben sind relativ zur Basis-Paketstruktur *de.tu_dresden.diplom.richter_mirko_mat2628335* zu verstehen.

8.4.2.1 clients

Alle Klassen und Unterpakete innerhalb des Pakets *clients* sind Teil der Implementation für den prototypischen Klient. Beschreibungen für die Abläufe innerhalb der einzelnen Komponentenimplementationen befinden sich in Kapitel 8.4.6. Folgende Paketnamen sind relativ zum Paket *clients* zu verstehen:

context Klassen für die Verwaltung der Kontextinformationen (s.Abb 25 und 27 im Anhang).

evidence Klassen für die Beweisverwaltung (s.Abb. 28 und 29 im Anhang).

8.4.2.2 common

In diesem Paket sind alle Klassen und Unterpakete enthalten, welche von mehr als einer Partei verwendet werden. Dazu gehören auch parteiübergreifende Konstanten.
Folgende Paketnamen sind relativ zum Paket *common* zu verstehen:

account Klassen zur Darstellung von Geldmenge, Währung und für die Identifikation von Konten beim Zeugen für Transfers im Rahmen eines Zählkontextes.

callback Standardimplementation für den Callback-Service bei den Teilnehmern Klient/Service, welcher im Falle bestimmter Fehlerbehandlungsroutinen durch den Zeugen gerufen wird.

config Schnittstellen für generische Konfigurationsklassen bei Handler-Operationen (z.B. Prüfung, ob ein Zeuge von einem Service akzeptiert wird, oder nicht).

context Basisklassen für die Kontextverwaltung bei den teilnehmenden Parteien sowie Bean-Klassen zur Abbildung der unterschiedlichen Kontext-Objekte. Das Unterpaket *comm* enthält Anfrage- und Antwortobjekte zur SOAP-Steuerung des Lebenszyklus eines Koordinationskontextes.

crypto Helferklassen für Signatur- und Verschlüsselungsoperationen.

dependency Klassen zur Beschreibung von Abhängigkeiten, welche per SOAP vom Klient/Service zum Zeugen übertragen werden können.

digestAgreement Anfrage- und Antwortklassen, welche für die Bearbeitung von Vorgängen zum Hashabgleich verwendet werden.

entity Basisklassen für die Abbildung von Entitäten (konkreter Klient, Zeuge oder Service).

evidence Basisklassen für die Beweisverwaltung und Beweisklassen, welche sowohl vom Klient als auch vom Service verwendet werden (s.Abb. 28 und 29 im Anhang).

handler Handlerklassen, welche von mindestens zwei beteiligten Parteien eingesetzt werden (s.Kap 8.4.3 sowie Abb. 34, 35 und 36 im Anhang). Das Unterpaket *data* enthält alle Klassen für zur Darstellung handlerbegleitender Daten (z.B. [IF]HandlingInfo; s.Kap. 8.4.9), *ex* enthält Exceptions, welche bei der Abarbeitung durch die Axis-Handler geworfen werden können und *utils* enthält Helferklassen (z.B. Axis-Pivot für den Hashabgleich).

info Objekte, welche als Ergebnis einer SOAP-Anfrage an die Informations- bzw. Verwaltungsschnittstelle des Zeugen zurückgeliefert werden (s.Kap. 8.4.12 und 8.4.13).

keystore Hilfsklassen zur Verwaltung der Zertifikate für Signaturen und Verschlüsselung.

listener Listenerschnittstellen, Standardimplementationen und Events für die Listenerfähigkeiten von Klient, Service und Zeuge.

management Anfrage- und Antwortobjekte für die Verwaltungsschnittstelle des Zeugen (s.Kap. 8.4.13).

result Klassen mit Basisfunktionalität für Ergebnisobjekte.

transform Klassen zur Beschreibung von Transformationen auf SOAP-Nachrichten vor der Berechnung des Hashs.

util Allgemeine Helferklassen für die Lösung von Standardproblemen bei der Implementation.

8.4.2.3 services

Alle Klassen und Unterpakete innerhalb dieses Pakets werden bei der Umsetzung der Zählfunktionalität auf Service-Seite verwendet. Beschreibungen für die Abläufe innerhalb der einzelnen Komponentenimplementationen befinden sich in Kapitel 8.4.7. Die folgenden Paketnamen sind relativ zum Paket *services* zu verstehen:

context Klassen zur Verwaltung der Kontextinformationen (s.Abb. 25 und 27 im Anhang).

evidence Klassen zur Verwaltung von Beweisen (s.Abb. 28 und 29 im Anhang).

handler Axis-Handler, welche *nur* zur Umsetzung der Zählfunktionalität bei Services zum Einsatz kommen (s.Abb. 35 im Anhang).

ticketBooking Service-Implementation für das "Buchen von Tickets" zur Durchführung der Evaluation (s.Kap. 8.5).

ticketReservation Service-Implementation für das "Reservieren von Tickets" zur Durchführung der Evaluation (s.Kap. 8.5).

8.4.2.4 witness

In diesem Paket befinden sich alle zeugenspezifischen Klassen und Unterpakete. Beschreibungen für die Abläufe innerhalb der einzelnen Komponentenimplementationen befinden sich in Kapitel 8.4.8. Die folgenden Paketnamen sind relativ zum Paket *witness* zu verstehen:

account Klassen für die Verwaltung von Nutzerkonten (s.Abb. 30 im Anhang).

context Klassen zur Verwaltung der Kontextinformationen beim Zeugen (s.Abb. 26 und 27 im Anhang). Das Unterpaket *ex* enthält Ausnahmen, welche bei der Verarbeitung von Kontextinformationen auftreten werden können.

dependency Klassen zur Verwaltung von Abhängigkeiten (s.Abb. 31 im Anhang).

entity Klassen für die Verwaltung der teilnehmenden Entitäten und ihrer Zertifikate.

evidence Klassen für die Verwaltung von Beweisen durch den Zeugen (s.Abb. 29 im Anhang).

handler Axis-Handler, welche *nur* zur Umsetzung der Zählfunktionalität beim Zeugen zum Einsatz kommen (s.Abb. 34 im Anhang).

process Implementierungen für die einzelnen Vorgänge, welche beim Zeugen ausgelöst werden können (z.B. Abb. 33 im Anhang). Dazu gehört beispielsweise die Initiierung einer Koordinationskontexterstellung oder eines Hashabgleichs. Das Unterpaket *result* enthält Ergebnisklassen, welche innerhalb der Prozessabarbeitung aufreten können.

8.4.3 Axis-Handler

Im vorliegenden Prototyp wurde von der Aufteilungsmöglichkeit der Funktionalität nachrichtenverarbeitender Handler in so genannten "Handlerketten" (*handlerchain*) intensiv

Gebrauch gemacht. Dieser von Axis [Axis] unterstützte Mechanismus führt zu wiederverwendbaren Komponenten (Handlern), welche bei Bedarf baukastenartig für die Bereitstellung einer bestimmten Funktionalität zusammengebaut und über zugehörige WSDD-Konfigurationsdateien (Web Service Deployment Descriptor) angepasst werden können (s.Abb. 34, 35 und 36 im Anhang).
Der nachrichtenbezogene Teil der Umsetzung des in Kapitel 8.3 entworfenen Protokolls basiert bei jedem Kommunikationsvorgang auf der passenden "Aneinanderreihung" einer Untermenge dieser Handler.
Im Folgenden sollen die entwickelten Handler, ihre Aufgabe und ihre Konfigurationsmöglichkeiten detailliert vorgestellt werden. Die Angabe von Parametern in der WSDD-Datei erfolgt über die Notation

```
<parameter name="${paramName}" value="${paramWert}"/>
```

innerhalb des zugehörigen Handlers. Weitere Axis-Konfigurationsmöglichkeiten können der offiziellen Internetseite [Axis] bzw. den Beispielen auf der beigelegten CD-ROM entnommen werden.
Jeder Handler verfügt über die Möglichkeit über den Parameter *logInfo* einen Text anzugeben, welcher dann von Log4J [Log4J] über das Ausgabe-Pattern *%XlogInfo* in die Ausgabenachrichten dieses Handlers integriert wird. Dies ist z.B. hilfreich für die Unterscheidung, bei welchem Teilnehmer eine Nachricht durch einen Handler ausgegeben wurde, falls ein und dieselbe Handlerimplementation von mehreren Teilnehmern verwendet wird.

8.4.3.1 BodyEncrypter

Verschlüsselt den Inhalt des Nachrichtenkörpers (*Body*). Folgende Parameter werden unterstützt:

Name	Wert

8.4.3.2 BodyDecrypter

Entschlüsselt den Inhalt des Nachrichtenkörpers (*Body*). Folgende Parameter werden unterstützt:

Name	Wert

8.4.3.3 BodySigner

Signiert den Inhalt des Nachrichtenkörpers. Folgende Parameter werden unterstützt:

Name	Wert
signingEntity	Kurzbeschreibung der Entität, mit welcher signiert werden soll (s.Kap. 8.4.8.3). Die Angabe *@handlingInfo* weist darauf hin, dass die im Interface IFHandlingInfo festgelegte Entität für die Body-Signatur verwendet werden soll.
signatureType	Legt fest, ob *enveloped* oder *wssec* (Referenzierung, wie in WS-Security (s.Kap. 6.4) vorgegeben) signiert werden soll.
signatureImpl	Welche konkrete Implementation soll zur Erzeugung der Signatur verwendet werden? Möglichkeiten: *tsik* oder *xss4j*.

Bei den möglichen Signaturimplementationen *tsik* [TSIK] bzw. *xss4j* [XSS4J] stehen nicht alle Signaturtypen zur Verfügung: *xss4j* nur *enveloped* und bei *tsik* nur *wssec*!

8.4.3.4 BodySignatureValidator

Prüft die möglicherweise vorhandene Signatur über den Inhalt des Nachrichtenkörpers. Bei der Auswahl des Signaturtyps und der -implementation muss darauf geachtet werden, dass diese den im zugehörigen *BodySigner* eingestellten Parameterwerten entsprechen. Das Ergebnis der Validierung wird im *IFSignatureHandlingInfo*-Objekt für den Nachrichtenkörper im begleitenden *IFHandlingInfo*-Objekt festgehalten. Folgende Parameter werden unterstützt:

Name	Wert
signatureType	Legt fest, ob *enveloped* oder *wssec* (Referenzierung, wie in WS-Security (s.Kap. 6.4) vorgegeben) signiert werden soll.
signatureImpl	Welche konkrete Implementation soll zur Erzeugung der Signatur verwendet werden? Möglichkeiten: *tsik* oder *xss4j*.

8.4.3.5 CountingContextGenerator

Wird beim Zeugen eingesetzt und generiert aus dem im *MessageContext* (begleitender Axis-Kontext bei der Abarbeitung der Handler) abgelegten Attribut *"contextCreated"* vom Typ *IFCoordinationContext* (der vom Zeugenprozess konfigurierte Koordinationskontext) die XML-Darstellung des *acp:CountingContext* (s.Kap. 8.4.4) und schreibt diesen in den SOAP-Header.

8.4.3.6 CountingContextReader

Wird von Klient sowie Service verwendet und liest das *acp:CountingContext*-Element aus dem SOAP-Header und übergibt die ausgelesenen Daten (als Property-Bean *und* DOM-Baum inkl. Zeugen-Signatur für die Weiterverwendung) an das begleitende *IFHandlingInfo*-Objekt.

8.4.3.7 CountingContextWriter

Schreibt das im *IFHandlingInfo* vorhandene Element *acp:CountingContext* in den SOAP-Header. Dabei wird die Signatur des Zeugen, welche durch den *CountingContextReader* gelesen wurde und die Integrität/Authentizität des Koordinationskontextes sicherstellt, mit in die neue Nachricht übertragen.

8.4.3.8 CountingContextEncrypter

Verschlüsselt den Inhalt des Elements *acp:CountingContext*, falls vorhanden. Folgende Parameter werden unterstützt:

Name	Wert

8.4.3.9 CountingContextDecrypter

Entschlüsselt den Inhalt des Elements *acp:CountingContext*. Folgende Parameter werden unterstützt:

Name	Wert

8.4.3.10 CountingContextSigner

Signiert, falls vorhanden, das Element *acp:CountingContext* mit dem Signierschlüssel der gewünschten Entität und fügt die Signatur als *enveloped*-Signatur (s.Kap. 6.1) in das Element ein.

Name	Wert
signingEntity	siehe *signingEntity* beim Handler *BodySigner*

8.4.3.11 CountingContextSignatureValidator

Validiert die Signatur innerhalb des *acp:CountingContext*-Elements und prüft diesen somit auf Integrität und Authentizität des Ausstellers. Das Ergebnis der Validierung wird als *IFSignatureHandlingInfo* für den Zählkontext im begleitenden *IFHandlingInfo*-Objekt abgespeichert.

8.4.3.12 CoordinationContextSigner

Dieser Handler kommt einzig beim Zeugen zum Einsatz und signiert, falls vorhanden, das Element *acp:CoordinationContext* mit dem Signierschlüssel (des Zeugen) und fügt die Signatur als *enveloped*-Signatur (s.Kap. 6.1) in das Element ein.

Name	Wert
signingEntity	siehe *signingEntity* beim Handler *BodySigner*

8.4.3.13 CoordinationContextSignatureValidator

Validiert die Signatur innerhalb des *acp:CoordinationContext*-Elements und prüft diesen somit auf Integrität und Authentizität (Aussteller: Zeuge). Das Ergebnis der Validierung wird als *IFSignatureHandlingInfo* für den Koordinationskontext im begleitenden *IFHandlingInfo*-Objekt abgespeichert. Folgende Parameter werden unterstützt:

Name	Wert
config	Voll qualifizierter Klassenname einer Implementation der Schnittstelle *IFCoordCtxConfig*, welche dazu bestimmt ist, festzustellen, ob der im Koordinationskontext angegebene Zeuge vom Service unterstützt wird.

8.4.3.14 StepContextWriter

Schreibt das aus dem im *IFHandlingInfo* enthaltenen Objekt vom Typ *IFStepContext* generierte Element *acp:StepContext* in das (bereits vorhandene) *acp:CountingContext*-Element im SOAP-Header.

8.4.3.15 StepContextSigner

Signiert, falls vorhanden, das Element *acp:StepContext* mit dem Signierschlüssel der gewünschten Entität und fügt die Signatur als *enveloped*-Signatur (s.Kap. 6.1) in das Element ein. Folgende Parameter werden unterstützt:

Name	Wert
signingEntity	siehe *signingEntity* beim Handler *BodySigner*

8.4.3.16 StepContextSignatureValidator

Validiert die Signatur innerhalb des *acp:StepContext*-Elements und prüft diesen somit auf Integrität und Authentizität des Ausstellers. Das Ergebnis der Validierung wird als *IFSignatureHandlingInfo* für den Kommunikationskontext im begleitenden *IFHandlingInfo*-Objekt abgespeichert.

8.4.3.17 ServiceContextManagerFeeder

Hilfsklasse, welche beim Service zur Anwendung kommt und für die korrekte Registrierung der Zählkontexte beim *IFServiceContextManager* (s.Kap. 8.4.7) verantwortlich ist. Der eigentlich Service muss deshalb nicht abgeändert werden, um die Verwaltung dieser Kontexte zu unterstützen.

8.4.3.18 DigestAgreementRequester

Ist dafür verantwortlich, den Abgleich des aus dem aktuellen Axis-*MessageContext* erstellten Nachrichtenhash mit dem der jeweils anderen Partei beim Zeugen zu beantragen. Dazu wird synchron oder asynchron eine neue Axis-Kommunikation zum Zeugen initiiert. Folgende Parameter werden unterstützt:

Name	**Wert**
fork	*false* für synchrone und *true* für asynchrone Abarbeitung des Hashabgleichs. *Synchron* stoppt die umgebende Kommunikation bis der Abgleich erfolgreich durchgeführt wurde.
confFile	Voller Pfadname zum Axis-Konfigurationsfile, welches für die Kommunikation zum Abgleichung der Hashs verwendet werden soll.
contextMethod	Bezugsmethode, über welche die Implementation eines *IFContextManager* zur Verwaltung des Abgleichvorganges bezogen werden kann. Derzeit unterstützt: *factory*.
contextFactoryClass	Falls als Methode *factory* gewählt wurde, wird hier der voll qualifizierte Name der Factory-Klasse angegeben.
contextFactoryMethod	Falls als Methode *factory* gewählt wurde, wird hier der Name der statisch implementierten und parameterfreien Methode für den Bezug der *IFContextManager*-Implementation angegeben.

8.4.4 Aufbau der Kontext-Information im SOAP-Header

Der auf Grund einer Anfrage eines autorisierten Klienten in Schritt 0a (s.Abb. 19) durch einen Zeugen erstellt Zählkontext (Schritt 0b) wird im SOAP-Header an den Klienten übermittelt. Er besitzt dabei den in Abbildung 23 dargestellten Aufbau.
Die dabei verwendeten Namensräume besitzen folgende Bedeutung:

ds *xmlns:http://www.w3.org/2000/09/xmldsig#* - Referenzierung zum Namensraum für digitale Signaturen (s.Kap. 6.1).

wsu *xmlns:http://schemas.xmlsoap.org/ws/2002/07/utility* - Referenzierung zum Namensraum für die Spezifikation von Basis-Daten und -Konstrukten, welche in einer Vielzahl von offiziellen Spezifikation Verwendung finden (z.B. Definition für das Format einer URL).

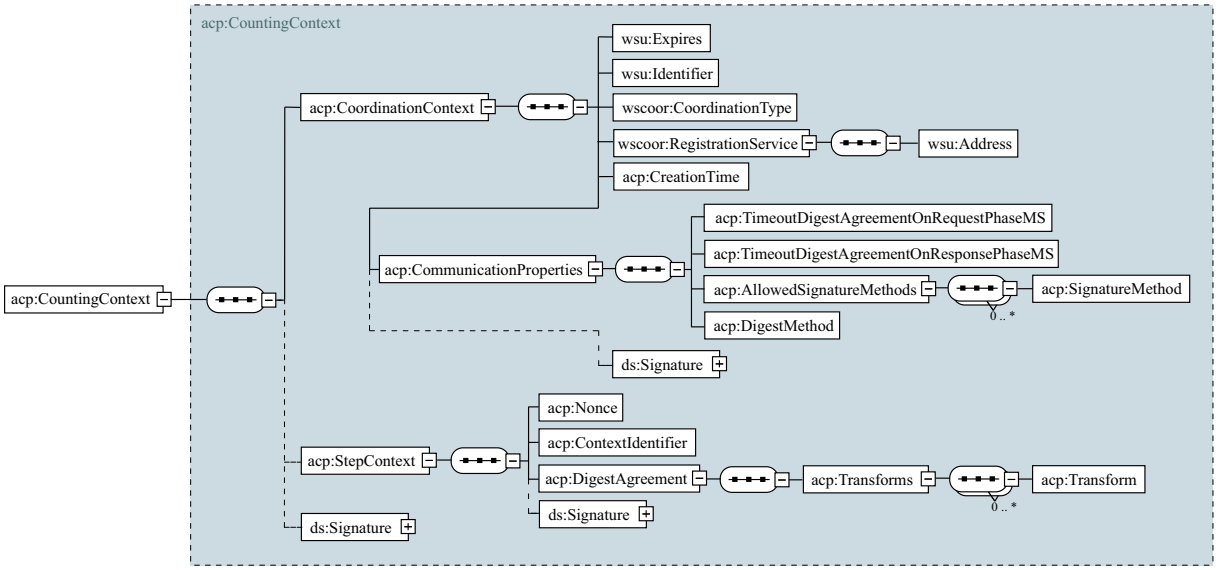


Abbildung 23: Aufbau der Kontext-Information im SOAP-Header

wscoor *xmlns:http://schemas.xmlsoap.org/ws/2004/10/wscoor* - Referenzierung zum Namensraum für WS-Coordination (s.Kap. 3.6).

acp *xmlns:http://diplom.compago.de/2005/03/acp* - Referenzierung zum Namensraum des Access Counter Protocol, einer XSD, welche im Rahmen dieser Arbeit entwickelt wurde.

Der *acp:CountingContext* ist das äußerste Element der transportablen Kontextinformationen und ist dafür vorgesehen in den Header einer SOAP-Nachricht eingefügt zu werden (Kindelement von *soapenv:Header*; Namensraum: *http://schemas.xmlsoap.org/soap/envelope/*). Es besteht aus drei Kindelementen: dem *acp:CoordinationContext*, dem *acp:StepContext* und einer *ds:Signature*. Das optionale *ds:Signature*-Element enthält die Daten einer möglichen *enveloped Signatur*, für die Sicherstellung der Integrität und der Authentizität eines *acp:CountingContext*-Elements und seines Inhaltes.
Das notwendige Element *acp:CoordinationContext* beinhaltet sämtliche Informationen, welche für den globalen Kontext (potentiell mehrere Kommunikationsbeziehungen mit mehreren Services) wichtig sind, wohingegen das optionale Element *acp:StepContext* Eigenschaften definiert, welche für eine einzelne Kommunikation (Klient-Server) einzuhalten sind.

Das *acp:CoordinationContext*-Element besitzt dabei den nachfolgenden Aufbau:
Das Element *wsu:Expires* besitzt keine Attribute und enthält als Textwert die Information, wann dieser Kontext ausläuft (Millisekunden seit 1980). Im *Identifier*-Element wird die eindeutige Kennung dieses Kontextes als Textwert abgelegt und an interessierte Parteien wetergegeben.
Für die Definition, um was für eine Art von Kontext es sich hierbei handelt, enthält das Element *CoordinationType* eine entsprechende Beschreibung in seinem Textteil (für den Prototypen: *http://diplom.compago.de/2005/03/acp*).
Das Element *wscoor:RegistrationService* enthält in seinem untergeordneten Element *wsu:Address* die Adresse des zu verwendenden Zeugenservice. Diese Information wird bei dem Service benötigt, der eine Anfrage eines Klienten bearbeiten soll. Anfragen für den Abgleich

der Hashwerte und Nachrichten zur Fehlerbehandlung werden an diese URL gesendet.
Im *acp:CreationTime*-Element wird vom Zeugen die Information, wann der Kontext erzeugt wurde (Millisekunden seit 1980) abgelegt.
Das wichtigste Element für die Festlegung, wie die weitere Kommunikation ablaufen soll ist *acp:CommunicationProperties*. Die Elemente *acp:TimeoutDigestAgreementOnRequestPhaseMS* und *acp:TimeoutDigestAgreementOnResponsePhaseMS* definieren, wie lange der Zeuge nach Erhalt eines Hashwertes auf den Erhalt des anderen Hashwertes warten soll, bevor er andere Maßnahmen einleited. Die *Request*-Phase spezifiziert dabei den Vorgang des Hashvergleiches aus der Request-Nachricht (s.Abb. 19 - Schritte 2a und 2b) und die *Response*-Phase den aus der korrespondierenden Response-Nachricht (Schritt 5a und 5b). Die Angabe erfolgt dabei in Millisekunden.
Innerhalb des Elements *acp:AllowedSignatureMethods* wird definiert, welche Verfahren zur Erzeugung einer Signatur unterstützt werden. Dies bezieht sich sowohl auf de Integritäts- und Authentifikationsschutz zwischen den Teilnehmern (Klient und Server) untereinander als auch zwischen ihnen und dem Zeugen. Dabei können in beliebig vielen untergeordneten *acp:SignatureMethod*-Elementen in dessen *algorithm*-Attributen Verweise auf entsprechende Festlegungen erfolgen (ähnlich wie bei XML-Signature in Kap. 6.1).
Das *acp:DigestMethod*-Element legt über den Inhalt seines *algorithm*-Attributs fest, welcher Algorithmus zur Erzeugung der Hashwerte verwendet werden muss. Dieser muss offensichtlicherweise bei Klient und Service identisch sein, da sonst ein Vergleich dieser Werte beim Zeugen scheitern würde.
Das für das prototypische Protokoll notwendige Element *ds:Signature* unterhalb des *acp:CoordinationContext*-Elements stellt die Integrität und die Authentizität des Koordinationskontextes sicher ("Unterschrift" des Zeugen).

Ein *acp:StepContext*-Element besitzt folgenden Aufbau:
Das *acp:Nonce*-Element wird als Schutz vor Replay-Angriffen (s.Kap. 5.2.2), zur sicheren Feststellung, dass ein Service, welcher einen Hashwert-Abgleich beim Zeugen "in Auftrag gibt", auch tatsächlich der vom Klienten gewünschte Service ist (verschlüsselt mit dessen Schlüssel) und zur Zuordnung der Request/Response-Nachrichten Nachrichten bei Klient, Service und Zeuge verwendet. Der *Nonce*-Wert erhält dadurch zusätzlich die Bedeutung eine eindeutige Identifikationskennung eines Kommunikationsschrittes (einmaliger Durchlauf der Schritte 1 bis 6 in Abb. 19) zu sein.
Das Element *acp:ContextIdentifier* ist eine Referenz auf den Textwert des *acp:Identifier*-Elements innerhalb von *acp:CountingContext*. Dadurch wird eine eindeutige Zuordnung des *acp:StepContext* zum *acp:CoordinationContext* erreicht, welche es ermöglicht, diese unabhängig voneinander zu signieren.
Die Details für den Abgleich der Hashwerte von Klient und Server werden im *acp:DigestAgreement*-Element festgehalten. Das *acp:Transforms*-Element definiert mögliche Transformationen, die auf der SOAP-Nachricht ausgeführt werden, bevor der Hash berechnet wird (ähnlich wie bei XML-Signaturen in Kap. 6.1). Jedes der beliebig oft vorhandenen *acp:Transform*-Elemente enthält dazu entweder einen Link auf einen Algorithmus innerhalb seines *Algorithm*-Attribut bzw. mögliche XPath-Ausdrücke [XPath] als Textinhalt. Im Prototyp ist es allerdings nur möglich über den Algorithmus *http://diplom.compago.de/-transformAlgorithmDigest#body* das *soapenv:Body*-Element als Ziel des Hash-Algorithmus zu definieren.

Nur, wenn sich Klient, Server und Zeuge über die festgelegten und im *acp:CountingContext*-Element übertragenen Eigenschaften einig sind, kann die gewünschte Kommunikation zu-

stande kommen.

8.4.5 Zertifikate

Die Erstellung und Verwaltung der für kryptografische Operationen notwendigen Schlüssel erfolgte für den Prototyp unter Verwendung des mit dem SDK ausgelieferten Werkzeug *keytool*. Das Batch-Skript zur Erstellung der notwendigen Zertifikate befindet sich auf der beiliegenden CD-ROM (*/cert/keytoolgen.bat*).
Die Funktionalität zum Auslesen der Zertifikate aus dem Dateisystem und die Zuordnung zu den jeweiligen applikationsspezifischen Entitäten wurde in den Klassen des Paketes *common.keystore* sowie in der Zertifikat-Entität-Abbildungsklasse *common.entity.EntityPool* umgesetzt.

8.4.6 Teilnehmer: Klient

Der Klient ist der Initiator eines Zählkontextes und verfügt über Mechanismen zur Verwaltung seiner Kontexte und seiner Beweise.
Die Implementation des Zählprotokolles kann entweder innerhalb des eigentlichen Klientencodes oder aber als konfigurierbarer vorgelagerter Handler erfolgen. Im vorliegenden Prototyp wurde der Klient selbst mit Code für die Abarbeitung einiger Protokollschritte (Anforderung neuer Kontexte, Registrierung neuer Kontexte etc.) implementiert. Andere Protokollschritte, wie z.B. der Hashabgleich wurden allerdings als konfigurierbarer Axis-Handler umgesetzt.

8.4.6.1 Kontext-Management

Die für die Kontextverwaltung verwendeten Klassen und ihre Abhängigkeiten werden in den Abbildungen 25 und 27 (siehe Anhang) als Klassendiagramme dargestellt.
Abbildung 25 zeigt dabei die eigentliche Verwaltungsklasse, den *ClientContext-Manager* (Verwendung über das Interface *IFClientContextManager*), welcher über das Factory-Pattern aus der *ServiceContextManagerFactory* bezogen wird. Jeder neu erzeugte Kontext, sowie jeder Kommunikationsvorgang und jeder Hashabgleich wird hier registriert und verarbeitet.
Weiterhin kann mit Hilfe des *ServiceContextManager* geprüft werden, ob ein Hashabgleich erfolgreich bzw nicht erfolgreich war, oder ob sich ein entsprechender Vorgang noch in der Abarbeitung (beim Zeugen) befindet.
Der *ClientContextManager* verfügt außerdem über die Möglichkeit unterschiedliche Listener zu registrieren. Diese werden dann je nach Typ bei der Initiation einer Kommunikation (einmaliges Durchlaufen der Schritt 1 bis 6), bei deren Beendigung, bei der Initiation eines Hashabgleichs sowie bei dessen Beendigung informiert.

Die zu verwaltenden Daten werden in *ClientContextInfo*-Objekten verwaltet (Verwendung über das Interface *IFClientContextInfo*) und im *ClientContext-Manager*, durchsuchbar nach der eindeutigen KontextID, abgespeichert (s.Abb. 27).

8.4.6.2 Beweis-Management

Die Verwaltung der notwendigen Beweise erfolgt im *ClientEvidenceManager* (Verwendung über das Interface *IFClientEvidenceManager*), welcher ebenfalls über das Factory-Pattern von der *ClientEvidenceManagerFactory* bezogen wird. Eingehende, als Beweis verwertbare Nachrichten werden hier registriert und verwaltet. Dazu wird der vom Zeugen erstellte Koordinationskontext mit dessen Signatur, der Kommunikationskontext (Inhalt des Elements *acp:StepContext*; s.Kap. 8.4.4) aus der Antwortnachricht vom Server mit dessen Signatur sowie die beiden Antwortnachrichten aus den Schritten 3b und 6b miteinander verkettet und abgespeichert.
Beweise werden in der prototypischen Implementation sowohl von der Klientimplementation selbst, als auch vom Handler *DigestAgreementRequester* (s.Kap. 8.4.3) beim Beweismanager registriert.

8.4.7 Teilnehmer: Service

Der Service ist der Teilnehmer am Protokoll, welcher Anfragen entgegen nimmt und seine erbrachten Leistungen beim Zeugen abrechnet. Der Prototyp wurde so entwickelt, dass für die Einführung einer Zugriffszählung bzw. -abrechnung keine Änderungen am Code bereits existierender Axis-basierter Services durchgeführt werden müssen. Die einzigen notwendigen Anpassungen beschränken sich auf die Axis-WSDD-Konfigurationsdateien für den jeweiligen Zielservice. Bereits existierende eigene Axis-Handler können weiterverwendet werden.

8.4.7.1 Kontext-Management

Ähnlich wie beim Klient wurde auch beim Service ein Mechanismus zur Verwaltung der Kontextinformationen angelegt. Die Abbildungen 25 und 27 im Anhang verdeutlichen die entworfene Klassenstruktur.
Die eigentliche Verwaltungsklasse, der *ServiceContextManager* (Verwendung über das Interface *IFServiceContextManager*) wird dabei auch hier über das Factory-Pattern von der *ServiceContextManagerFactory* bezogen.
Der *ServiceContextManager* wird auch hier, wie an der stark ähnelnden Ableitungshierarchie zum *ClientServiceContextManager* erkennbar, dazu verwendet, um Kontexte, Kommunikationsvorgänge und Hashabgleiche zu verwalten. Der aktuelle Zustand laufender Kommunikationsschritte können hier ebenso abgefragt und die bereits vorgestellten Typen von Listenern registriert werden.

Die zu verwaltenden Daten werden in *ServiceContextInfo*-Objekten verwaltet (Verwendung über das Interface *IFServiceContextInfo*) und im *ServiceContextManager*, durchsuchbar nach der eindeutigen KontextID, abgespeichert (s.Abb. 27).

8.4.7.2 Beweis-Management

Der *ServiceEvidenceManager* (Verwendung über das Interface *IFServiceEvidenceManager*; Bezug über die Factory *ServiceEvidenceManagerFactory*) wird dazu verwendet, eingehende verwertbare Beweise zu verwalten. Dem eigentlichen Service vorgeschaltete Handler registrieren hier eingehende signierte Koordinationskontexte, Kommunikationskontexte und Antwortnachrichten aus den Schritten 3a und 6a.
Die Verkettung zusammengehöriger Beweise im *ServiceEvidenceManager* garantiert die beweisbare Rekonstruierbarkeit von Kommunikationsvorgängen.

8.4.8 Teilnehmer:Zeuge

Der Zeuge nimmt im vorliegenden Protokoll die Rolle des Koordinators ein. Er führt

8.4.8.1 Kontext-Management

Zur Verwaltung der globalen Koordinationskontexte wurde beim Zeugen eine Klassenstruktur entworfen, wie sie in den Abbildung 26 und 27 (Anhang) dargestellt ist. Die Hauptverwaltungsklasse, der *WitnessContextManager* (Zugriff über das Interface *IFWitnessContextManager*), wird auch hier wieder aus einer Factory (*WitnessContextManagerFactory*) bezogen.
Koordinationskontexte werden von einem referenzierten Objekt vom Typ *ContextInfoPool* verwaltet und von einer Instanz des *LifecycleManager* überwacht. Der *LifecycleManager* ist dafür verantwortlich zu prüfen, ob ein Kontext zum Zeitpunkt seiner Verwendung überhaupt noch gültig ist oder ob eine durch den Klient initiierte Löschung des Kontext zum jeweiligen Zeitpunkt durchgeführt werden kann, ohne das Protokoll zu verletzen.
Eingehende Anfragen zum Abgleich von Hashwerten werden durch Objekte des Typs *DigestAgreementProcess* (Abspeicherung im zugehörigen *IFWitnessContextInfo*-Objekt; Zugriff über das Interface *IFDigestAgreementProcess*) verwaltet. Sind zwei zusammengehörige Anfragen von Klient und Service eingetroffen, wird bei Übereinstimmung der Hashwerte, Abhängigkeiten und Kosten für die Nutzung eine Erfolgsnachricht an die Teilnehmer versendet. Anderenfalls wird, wie in Kapitel 8.4.11 beschrieben, die Fehlerbehandlung eingeleitet.

8.4.8.2 Beweis-Management

Der *WitnessEvidenceManager* (Verwendung über das Interface *IFWitnessEvidenceManager*; Bezug über die Factory *WitnessEvidenceManagerFactory*) wird dazu verwendet, eingehende verwertbare Beweise zu verwalten. Dazu wird die signierte Anfrage des Klienten zur Erzeugung eines Kontextes und die signierten Hashabgleichsnachrichten aus den Schritten 2a, 2b, 5a und 5b abgespeichert und je nach Zusammengehörigkeit untereinander verkettet. Mit Hilfe dieser Daten kann der Zeuge die Rechtmäßigkeit der Kontexterstellung nachweisen und zusammen mit einem der beiden anderen Teilnehmer den Kommunikationsverlauf nachweisbar rekonstruieren.

8.4.8.3 Entitäts-Management

Auflistung der im Pool vordefinierten Entitäten und ihre Zugriffskürzel nicht vergessen!

8.4.9 Das *IFHandlingInfo*-Objekt

HandlingInfo-Objekte (Paket: *common.handler.data*) werden dazu verwendet (Zugriff über das Interface *IFHandlingInfo*), sämtliche Daten, die bei den Verarbeitungsschritten der eingehenden Axis-Handlerketten entstehen, zwischenzuspeichern und für spätere Auswertungen verfügbar zu machen. Weiterhin werden sie zur Laufzeit-Konfiguration ausgehender Axis-Handlerketten verwendet (s.Abb. 34, 35 und 36 im Anhang).
Die Abspeicherung dieser Objekte erfolgt dabei Thread-Lokal (siehe *java.lang.ThreadLocal*

[JSun]) als statischer Member innerhalb der Klasse *common.handler.BaseHandler*. Der Aufbau dieser Datenstruktur ist aus dem Klassendiagramm in Abbildung 32 (Anhang) ersichtlich.

8.4.10 Kommunikationsschritte

Erwähnen der Konfigurationsfiles für client-Server-Aktivitäten (z.B. digestAgreement)!

8.4.11 Fehlerbehandlung

Beschreiben, was in Fehlerfällen (digest bleibt aus, teilnehmer reagiert nicht...) passiert.

8.4.12 Informationsschnittstelle des Zeugen

Beschreiben der Schnittstelle zum Einholen von Informationen über eigene Konten, eigene Kontexte etc.

8.4.13 Verwaltungsschnittstelle des Zeugen

Möglichkeiten zur Administration des Zeugen.

8.4.14 Unit-Tests

8.5 Evaluation

8.6 Ausblick

- Personaliserung der Listener in ClientContextManager (über IFEntity)
- Erweiterung der WS-Policy des Servers für Protokollanforderungen
- Signatur des CountingContext nach WS-Security

9 Abkürzungsverzeichnis

API **A**dvanced **P**rogramming **I**nterface

DoS **D**enial **o**f **S**ervice

DNS **D**omain **N**ame **S**ervice

URL **U**niform **R**esource **I**dentifier

URL **U**niform **R**esource **L**ocator

10 Referenzen

Literatur

[AlphaWorks] IBM AlphaWorks, Offizielle Website, 2005.
URL: http://www.alphaworks.ibm.com/

[An05] answer.com: *Wikipedia: Paul Mockapetris, the proposer of DNS*
URL: http://www.answers.com/topic/paul-mockapetris; Abruf: 21.04.2005

[Apache] Apache Foundation, Offizielle Website, 2005.
URL: http://www.apache.org/

[ASN.1] Abstract Syntax Notation One Consortium, 2005.
URL: http://www.asn1.org

[Axis] Webservices - Axis, Offizielle Website, 2005.
URL: http://ws.apache.org/axis/index.html

[Be02] Bellwood, Tom: *Rocket ahead with UDDI V3*, November 2002.
URL: http://www-106.ibm.com/developerworks/webservices/library/ws-uddiv3/

[Ce99] CERT® Coordination Center: *Denial of Service Attacks*, 1997.
URL: http://www.cert.org/tech_tips/denial_of_service.html; Abruf: 04.02.2005

[Ce05] CERT® Coordination Center: *CERT/CC Statistics 1988-2004*, Januar 2005.
URL: http://www.cert.org/stats/cert_stats.html; Abruf: 22.03.2005

[Ch03] Chiusano, Joseph: *UDDI and ebXML Registry: A Co-Existence Paradigm*, April 2003.
URL: http://www.mywebservices.org/index.php/article/articleview/984/1/21; Abruf: 20.12.2004

[Ci02] Cibulskis, Kristian: *The ebXML Registry*, 2002.
URL: http://www.sys-con.com/xml/article.cfm?id=315; Abruf: 07.01.2005

[Di05] Diffie, Whitfield: *Dr. Whitfield Diffie's Research Home Page*, April 2005.
URL: http://research.sun.com/people/diffie/; Abruf: 16.04.2005

[DTD] W3C XML Specification - Data Type Definition, 2005.
URL: http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm; Abruf: 04.03.2005

[Ec02] Eckert, Claudia: *Sicherheit*, XML-Workshop FhG-IPSI, Darmstadt, November 2002.

[FR97] Franklin, Matthew K. | Reiter, Michael K.: *Fair Exchange with a Semi-Trusted Third Party*, 4th ACM Conference on Computer and Communications Security, ACM Press, New York 1997, 1-5, Zürich, April 1997.

[Gi04] Gibson, Steve: *The Strange Tale of the Denial of Service Attacks against GRC.COM*, Gibson Research Corporation, Juni 2004.
URL: http://grc.com/dos/grcdos.htm; Abruf: 07.02.2005

[Go00] Gottschalk, Karel: *Web Services architecture overview*, September 2000.
URL: http://www-106.ibm.com/developerworks/webservices/library/w-ovr/?dwzone=webservices; Abruf: 16.04.2005

[GS03] Garfinkel, Simson | Spafford, Gene: *Practical UNIX & Internet Security*, O'Reilly, ISBN: 0-596-00323-4, Februar 2003.

[HM+01] Hinkelman, Scott | Macroibeaird, Sean | Manes, Anne Thomas | McKee, Barbara: *Using UDDI to Find ebXML Reg/Reps*, Mai 2001.
URL: http://www.ebxml.org/specs/rrUDDI.pdf

[Ho03] Holznagel, Bernd: *Modul X: Grundlagen des Rechts der IT-Sicherheit*, Vorlesungsscript Rundfunkrecht, Universität Münster, WS 2003/2004.

[Ha05] Hauben, Michael: *History of ARPANET*, Januar 2005.
URL: http://www.dei.isep.ipp.pt/docs/arpa.html; Abruf: 21.04.2005

[IETF] The Internet Engineering Task Force, 2005.
URL: http://www.ietf.org/

[JSun] Sun - Java, Offizielle Website, 2005.
URL: http://java.sun.com/

[JZ03] Jeckle, Mario | Zengler, Barbara: *Sicherung von Web Services durch Firewalls*, DaimlerChrysler Forschungszentrum Ulm
URL: http://www.jeckle.de/files/oop2003.pdf; Abruf: 04.12.2004

[Ko78] Kohnfelder, Lauren: *Towards a Practical Public-Key Cryptosystem*, Mai 1978.
URL: http://www.simson.net/ref/1978/kohnfelder78.pdf; Abruf: 16.04.2005

[Log4J] Log4J - Logging Services, Offizielle Website, 2005.
URL: http://logging.apache.org/log4j/docs/index.html

[LW03] Little, Mark | Webber, Jim: *Introducing WS-Coordination - A big step toward a new standard*, Web Service Journal, April 2003.
URL: http://www.sys-con.com/webservices/articleprint.cfm?id=542; Abruf: 22.12.2004

[Mc01] McLaughlin, Brett: *Java & XML - 2nd Edition*, O'Reilly, ISBN: 0-596-00197-5, August 2001.

[Mue04] Müller, Günter: *Telematik 4 / IT-Sicherheit*, Vorlesungsscript Telematik, Freiburg, WS 2004/2005.

[Ns04] Network Secure: *Die neue Script-Kiddie Kultur beleuchtet*, März 2004.
URL: http://www.network-secure.de/index.php?option=com_content&task=view&id=-1948&Itemid=661; Abruf: 16.04.2005

[OA04] OASIS: *UDDI Version 3.0.2 Spec Technical Committee Draft*, Oktober 2004.
URL: http://uddi.org/pubs/uddi_v3.htm

[OASIS] Organization for the Advancement of Structured Information Standards, 2005.
URL: www.oasis-open.org

[On02] O'Neill, Mark: *Web Service Security*, McGraw-Hill/Osborne, Berkley, California, 2003.

[Pf00] Pfitzmann, Andreas: *Sicherheit in Rechnernetzen*, Vorlesungsscript 2000/2001, TU-Dresden, Oktober 2000.

[PS98] Pfitzmann, Birgit | Schunter, Matthias: *Die SEMPER Sicherheitsarchitektur für elektronischen Handel im Internet*, Universität des Saarlandes, 1998.

[Ri04] Richter, Mirko: *Adaption zustandsbehafteter Komponenten*, Großer Beleg, Technische Universität Dresden, 2004.

[RSALab] RSA Security, RSA Laboratories, 2005.
URL: http://www.rsasecurity.com/rsalabs/

[Ru02] RUS-CERT: *Die Sicherheit hinkt der Funktionalität hinterher*, November 2002.
URL: http://cert.uni-stuttgart.de/presse/CZ200211-2.php; Abruf: 20.04.2005

[Sc00] Schunter, Matthias: *Optimistic Fair Exchange*, Dissertationsthese, Universität des Saarlandes, Oktober 2000.

[SEMPER] Secure Electronic Marketplace for Europe, Offizielle Website, 2005.
URL: http://www.semper.org

[SOAP] Simple Object Access Protocol Version 1.2, 2005.
URL: http://www.w3.org/TR/soap/; Abruf: 05.02.2005

[Systinet] Systinet, Offizielle Website, 2005.
URL: http://www.systinet.com/

[Thawte] Thawte, Offizielle Website, 2005.
URL: http://www.thawte.com/

[Tomcat] Tomcat - Servlet Container, Offizielle Website, 2005.
URL: http://jakarta.apache.org/tomcat/index.html

[TSIK] Verisign Trust Services Integration Kit, Offizielle Website, 2005.
URL: http://www.xmltrustcenter.org/developer/verisign/tsik/index.htm

[UDDI] Universal Description, Discovery and Integration, Offizielle Website, 2005.
URL: www.uddi.org

[Verisign] Verisign Inc., Offizielle Website, 2005.
URL: http://www.verisign.com/

[W3C] World Wide Web Consortium, Offizielle Website, 2005.
URL: http://www.w3.org/

[WSCoor] *Web Service Coordination Specification*, 2005.
URL: ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf; Abruf: 10.03.2005

[WSDL] Web Service Description Language Version 1.1, März 2001.
URL: http://www.w3.org/TR/wsdl

[WSPolAss] Web Services Policy Assertions Language (WS-PolicyAssertions), Mai 2003.
URL: http://www-128.ibm.com/developerworks/library/ws-polas/; Abruf: 18.04.2005

[WSPolAtt] Web Services Policy Attachment (WS-PolicyAttachment), September 2004.
URL: http://www-128.ibm.com/developerworks/library/specification/ws-polatt/; Abruf: 18.04.2005

[XML] eXtensible Markup Language (XML), August 2004.
URL: http://www.w3.org/XML/; Abruf: 15.12.2004

[XPath] XML Path Language (XPath) Version 1.0, November 1999.
URL: http://www.w3.org/TR/xpath; Abruf: 23.03.2005

[XSD] XML Schema Definition Version 1.0, Mai 2001.
URL: http://www.w3.org/XML/Schema; Abruf: 15.12.2004

[XSig] XML-Signature Syntax and Processing, Februar 2001.
URL: http://www.w3.org/TR/xmldsig-core/; Abruf: 27.02.2005

[XSL]

[XSpy] Altova: *XML Spy Evaluation*, Dezember 2004.
URL: http://www.altova.com; Abruf: 02.12.2004

[XSS4J] IBM XML Security Suite, Offizielle Website, 2005.
URL: http://www.alphaworks.ibm.com/tech/xmlsecuritysuite

[Zi05] Zimmermann, Philip: *Phil Zimmermann's Home Page*, April 2005.
URL: http://www.philzimmermann.com/EN/background/index.html; Abruf: 16.04.2005

11 Anhang

Eigenschaften / Kommschritt (nach ...)	Verfügbare Daten: Klient (K)	Verfügbare Daten: Server (S)	Verfügbare Daten: Zeuge (Z)	Verfügbare Beweise: Klient	Verfügbare Beweise: Server	Verfügbare Beweise: Zeuge	Angriffsmöglichkeiten: Klient	Angriffsmöglichkeiten: Server	Zeuge
initial	SI_Z, $Shr_{K,Z}$, $Shr_{K,S}$, KE, KSE_{Anf}, Abh_1 Anfrage	(KE wird von S unterstützt da K diese anhand der verfügbaren Metadaten von S festgelegt hat)	/	-> B_K1	-> B_S2	/	/	/	/
0a	-	-	SI_K, $Shr_{K,Z}$, KE, $NONCE_1$, $Kontext_Z$	-	-	Nachweis (Sig_K(...) und Eindeutigkeit von $NONCE_1$), dass K Kontext erstellen möchte	-	-	versucht sich als K auszugeben
0b	$Kontext_Z$	-	-	Zeuge hat $Kontext_Z$ erstellt (Sig_Z(...))	-	-	-	-	-
1	-	SI_K, $Kontext_Z$, SI_Z, $Shr_{K,S}$, KSE_{Anf}, Abh_1, $NONCE_2$, Anfrage	-	-	Klient hat Anfrage gestellt (Sig_K(Anfrage) und SI_K) -> B_S1	-	-	-	-
2a + 2b	-	-	H_1(Anfrage'), $NONCE_2$, SI_S, $Shr_{S,Z}$, $Abh1$, Cnt_{Inc}, Cnt_{Dec}, Ktx (zugehöriger Kontext)	-	-	kennt Hash der Anfrage (-> B_Z1), kennt ausgehandelten Preis und Abhängigkeiten, Sig_K(...) Sig_S(...) und $NONCE_2$ garantieren spätere Beweisbarkeit der in (3x) getroffenen Entscheidung	-	-	-
3a + 3b	-	Antwort	-	Zeuge kann korrekten Erhalt der Anfrage bei Server bezeugen (Sig_Z(...)); indirekter Nachweis über Inhalt	Zeuge kann bezeugen, dass Anfrage wirklich von Klient (Sig_Z(...)); indirekter Nachweis über Inhalt	-	-	versucht abzurechnen, ohne Gegenleistung zu erbringen -> M_S1	-
4	Antwort	-	-	Server hat Antwort geschickt (Sig_S(Antwort)) -> B_K2	-	-	versucht Antwort unabgerechnet zu behalten -> M_K1	-	-
5a + 5b	-	-	$Sig_K(H_1$(Antwort')), $Sig_S(H_1$(Antwort))	-	-	kennt Hash der Antwort (-> B_Z2)	-	-	-
6a + 6b	-	-	-	Zeuge kann bezeugen, dass Antwort wirklich vom Server (Sig_Z(...)); indirekter Nachweis über Inhalt	Zeuge kann korrekten Erhalt der Antwort bei Klient bezeugen (Sig_Z(...)); indirekter Nachweis über Inhalt -> B_S3	-	-	-	-

Abbildung 24: Betrachtung der Beweislage und des Angriffspotentials

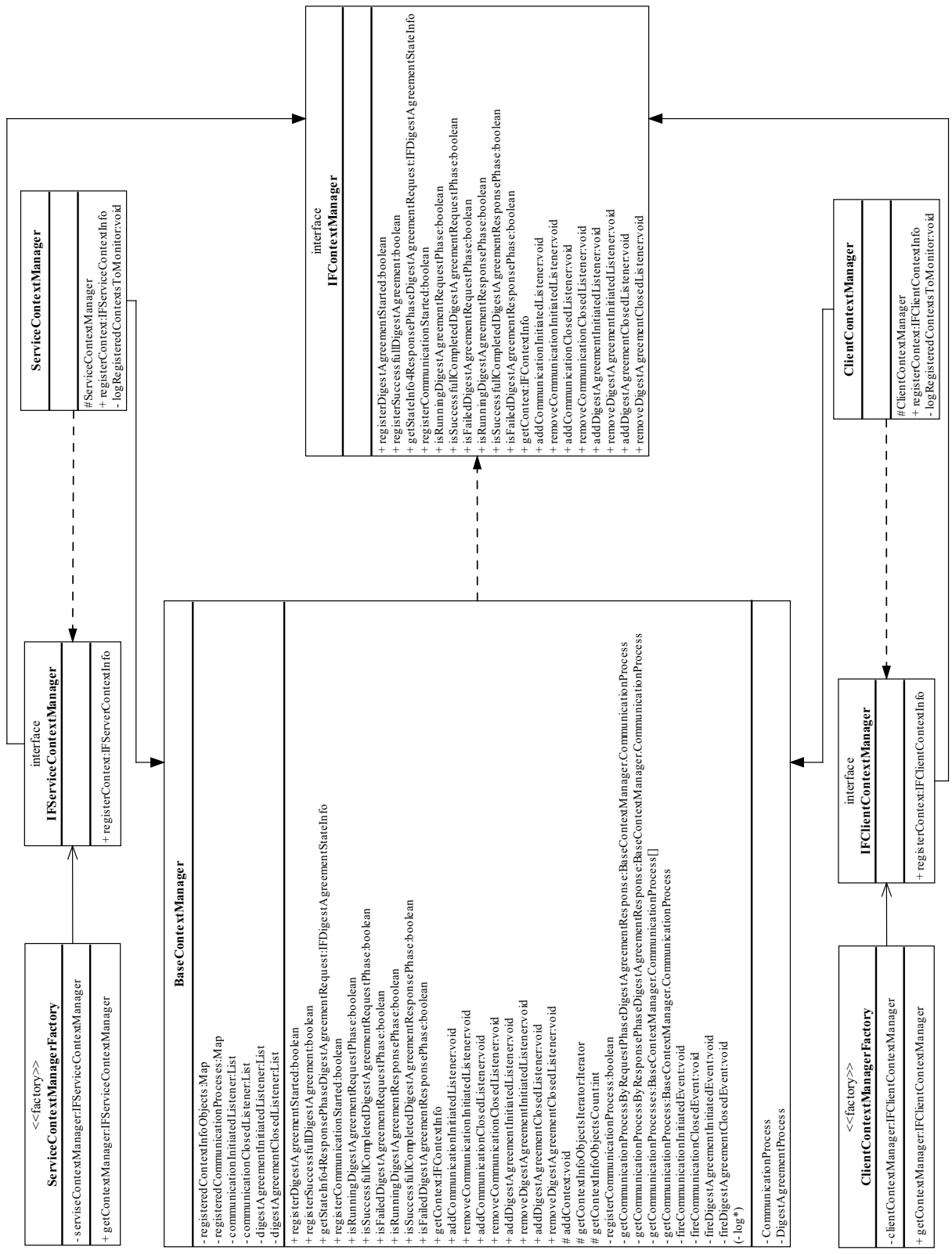


Abbildung 25: Klassendiagramm: ContextManager - Client und Service

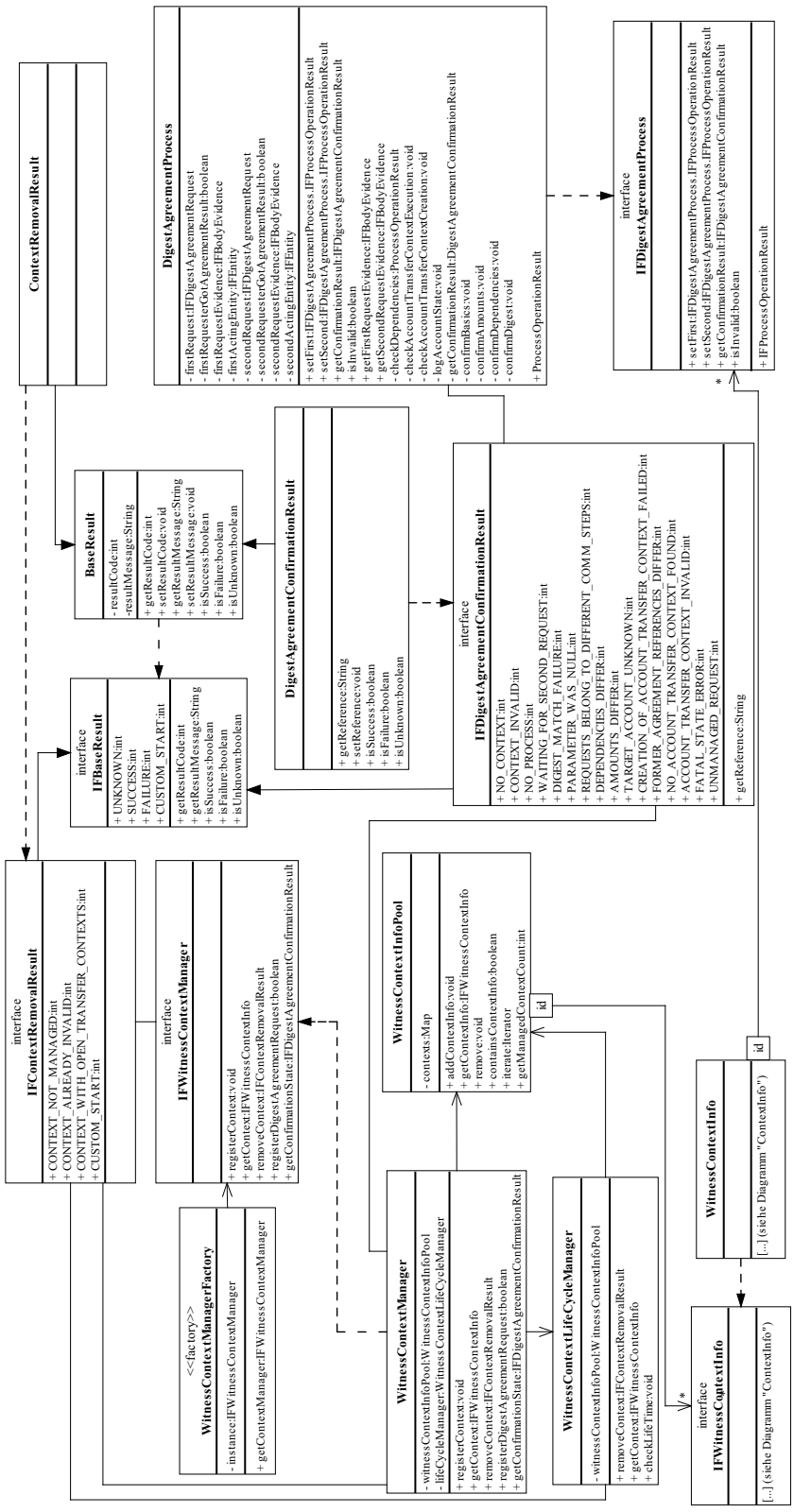


Abbildung 26: Klassendiagramm: ContextManager - Zeuge

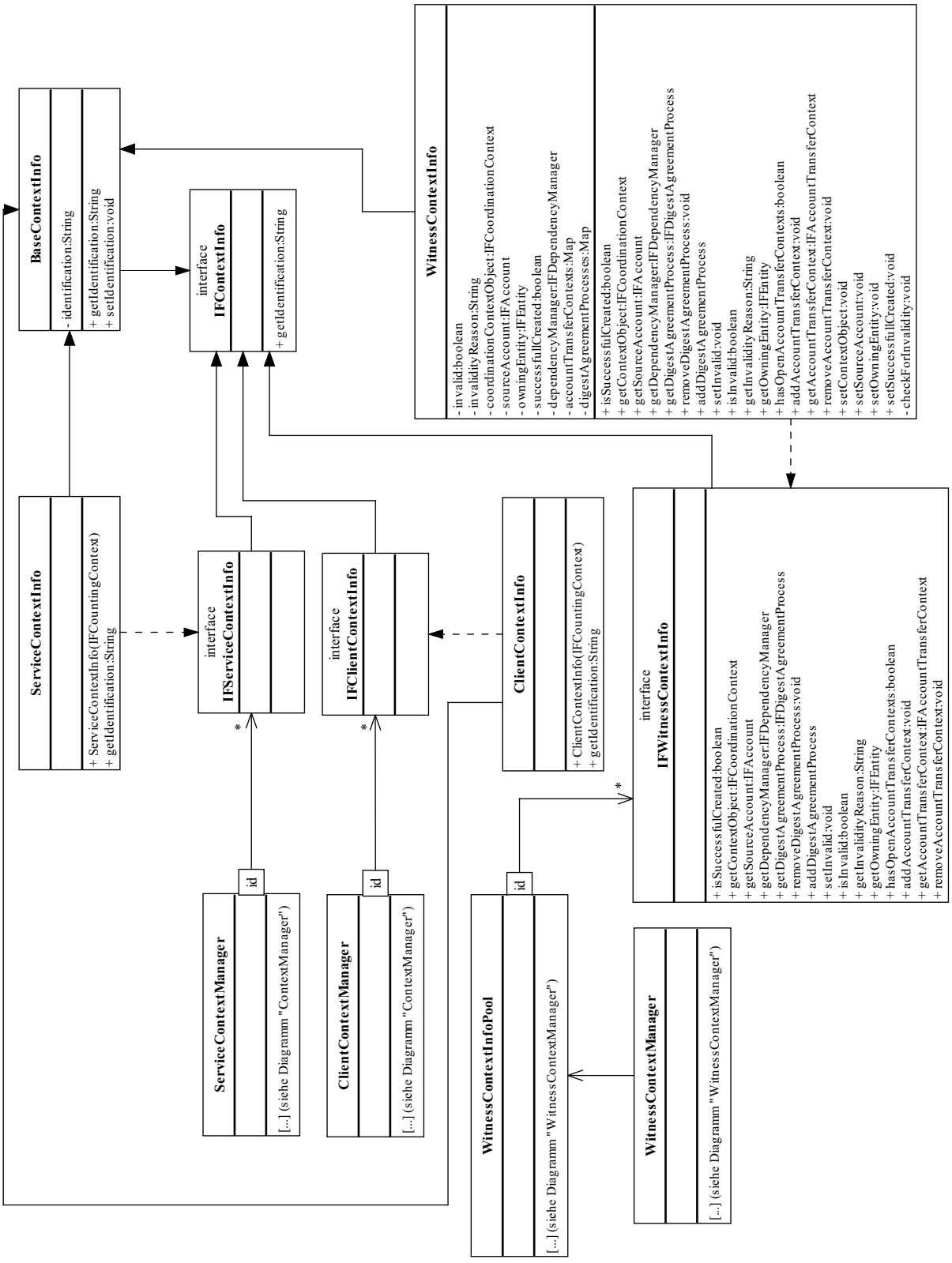


Abbildung 27: Klassendiagramm: ContextInfo - Client, Service und Zeuge

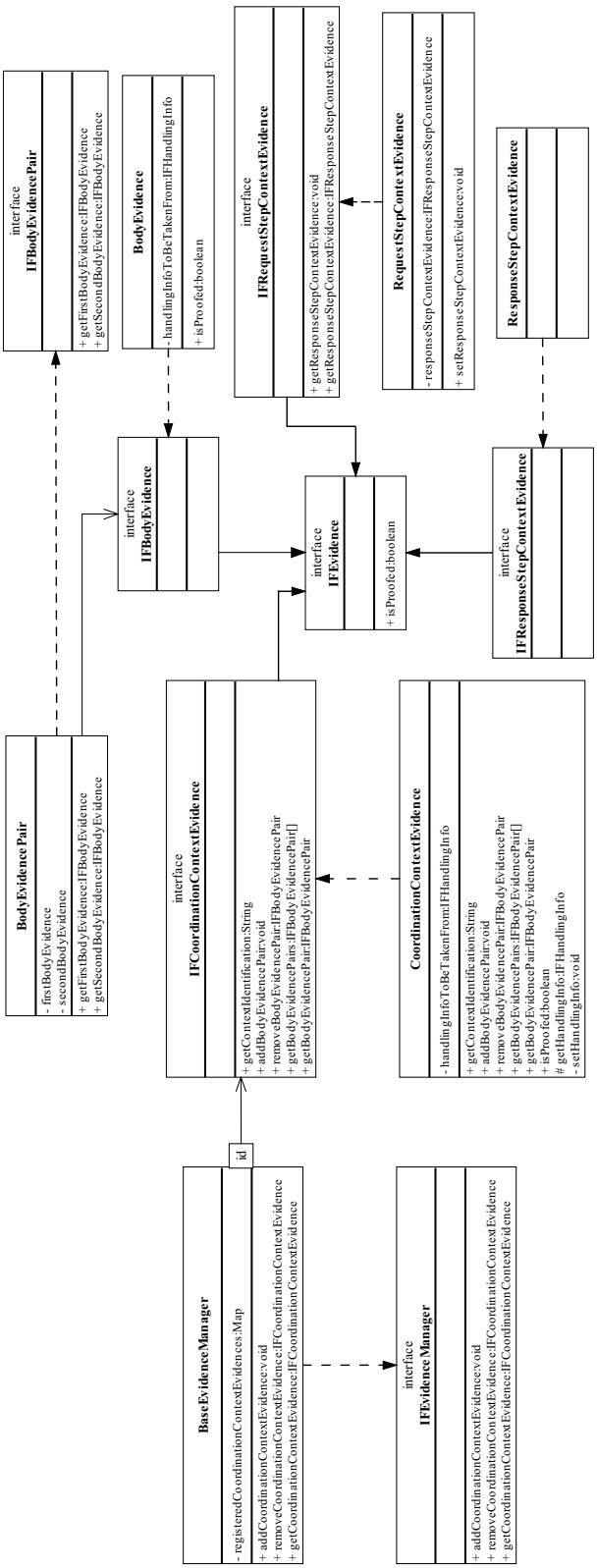


Abbildung 28: Klassendiagramm: Basis-Beweismanagement

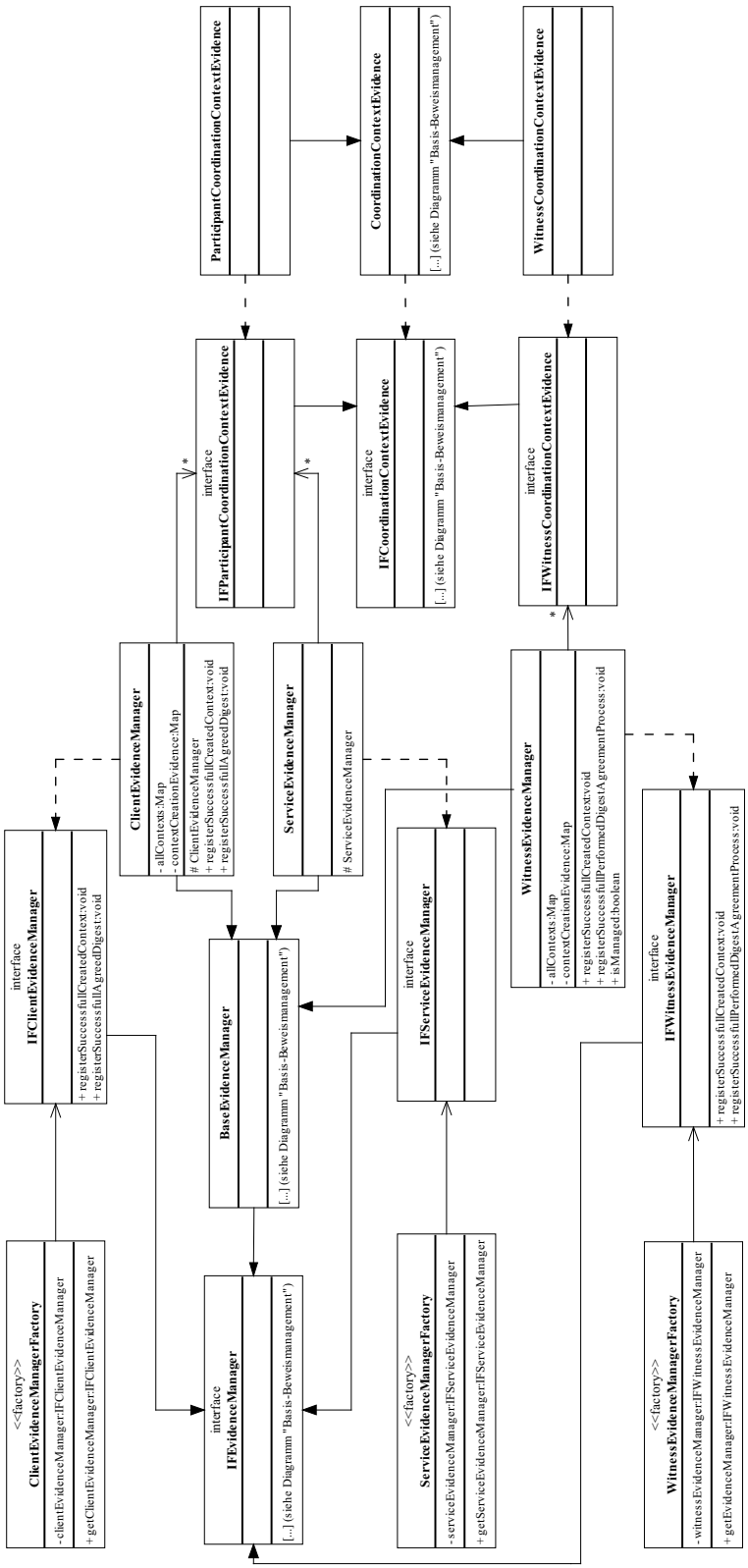


Abbildung 29: Klassendiagramm: Beweismanagement Client, Server und Zeuge

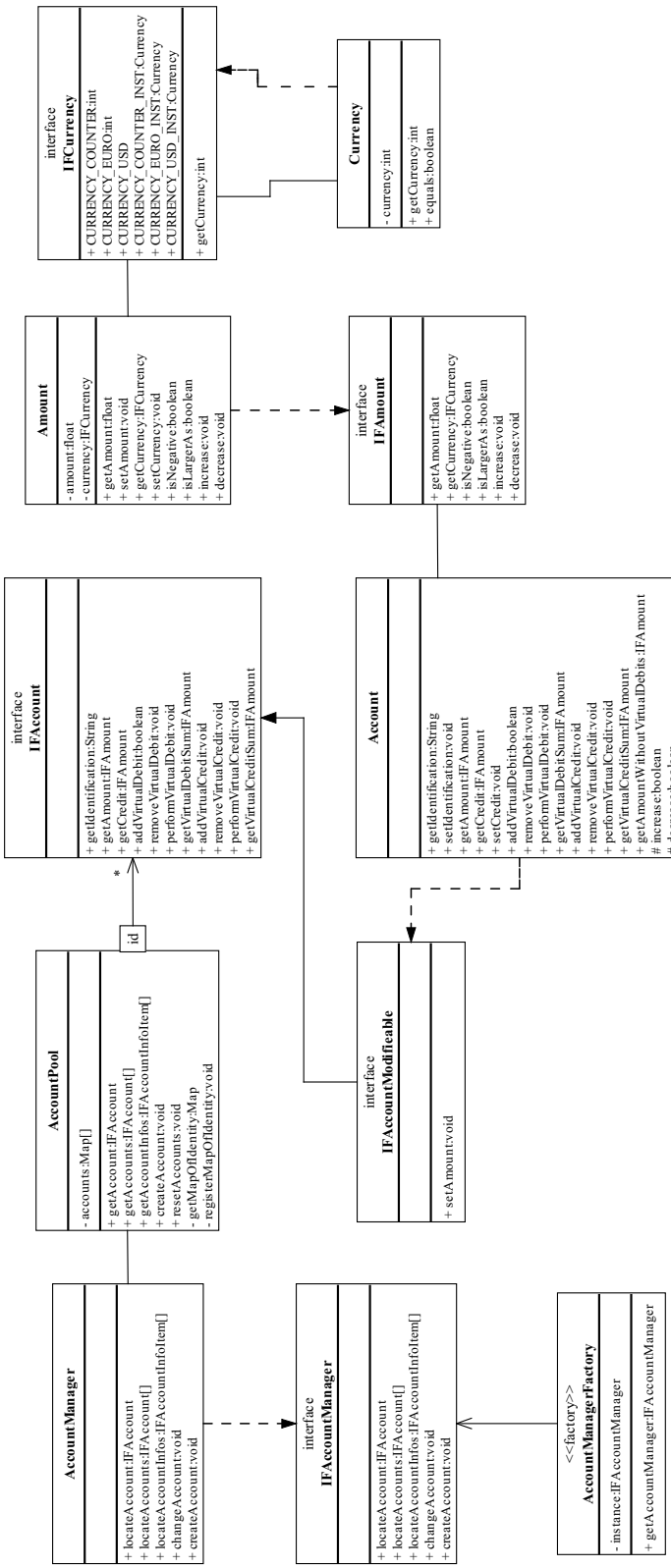


Abbildung 30: Klassendiagramm: AccountManager

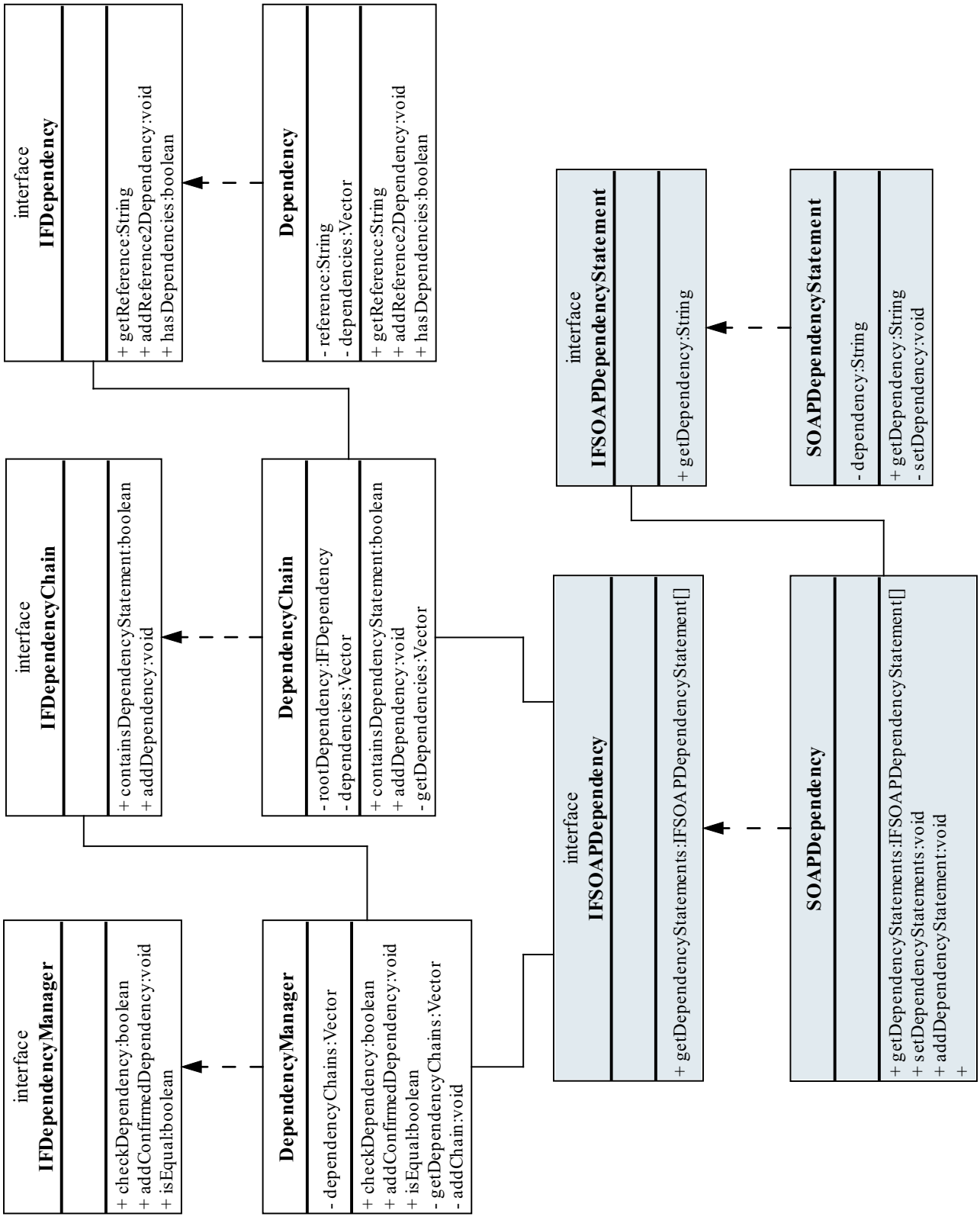


Abbildung 31: Klassendiagramm: DependencyManager

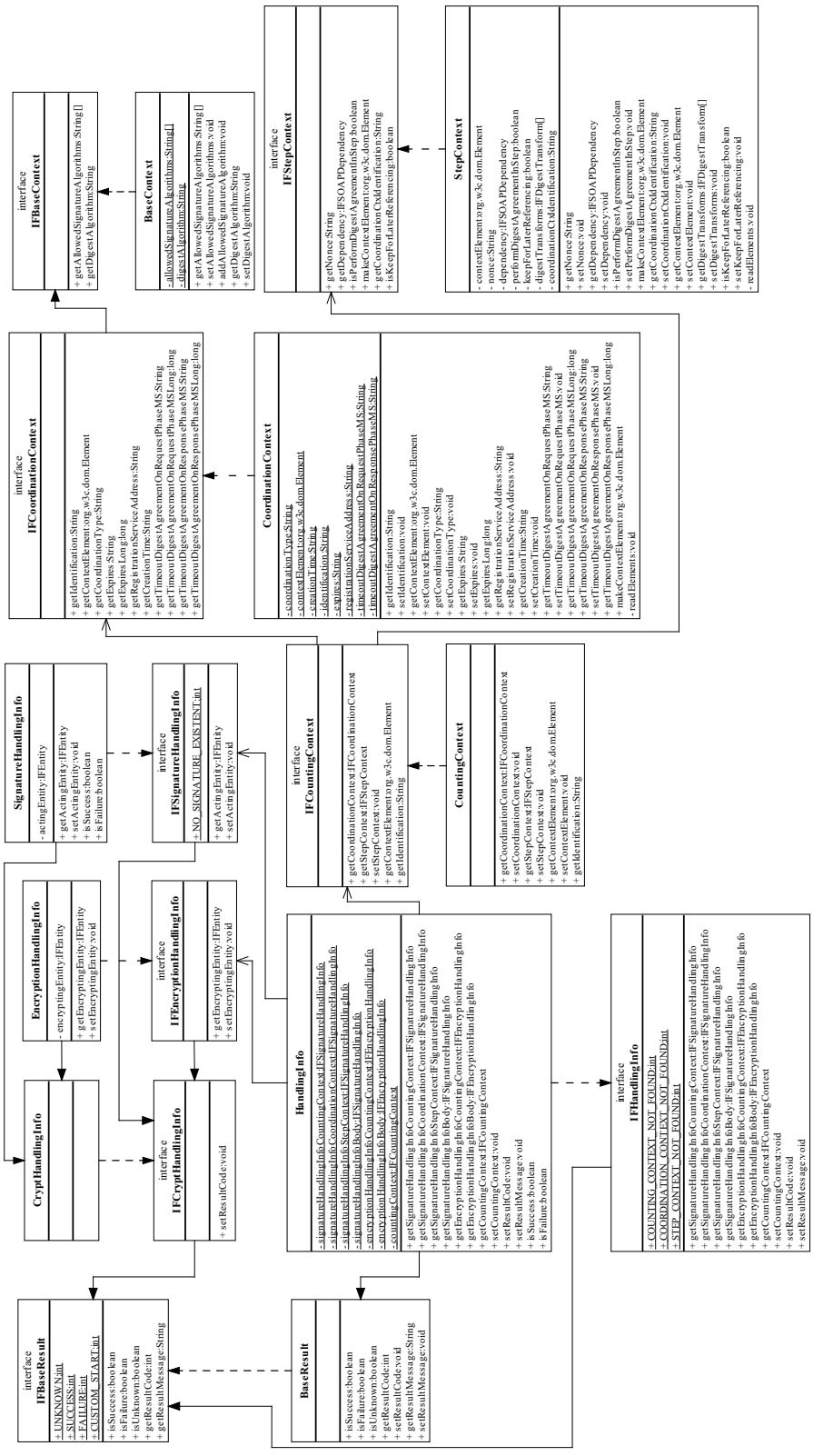


Abbildung 32: Klassendiagramm: Struktur des HandlingInfo-Objektes

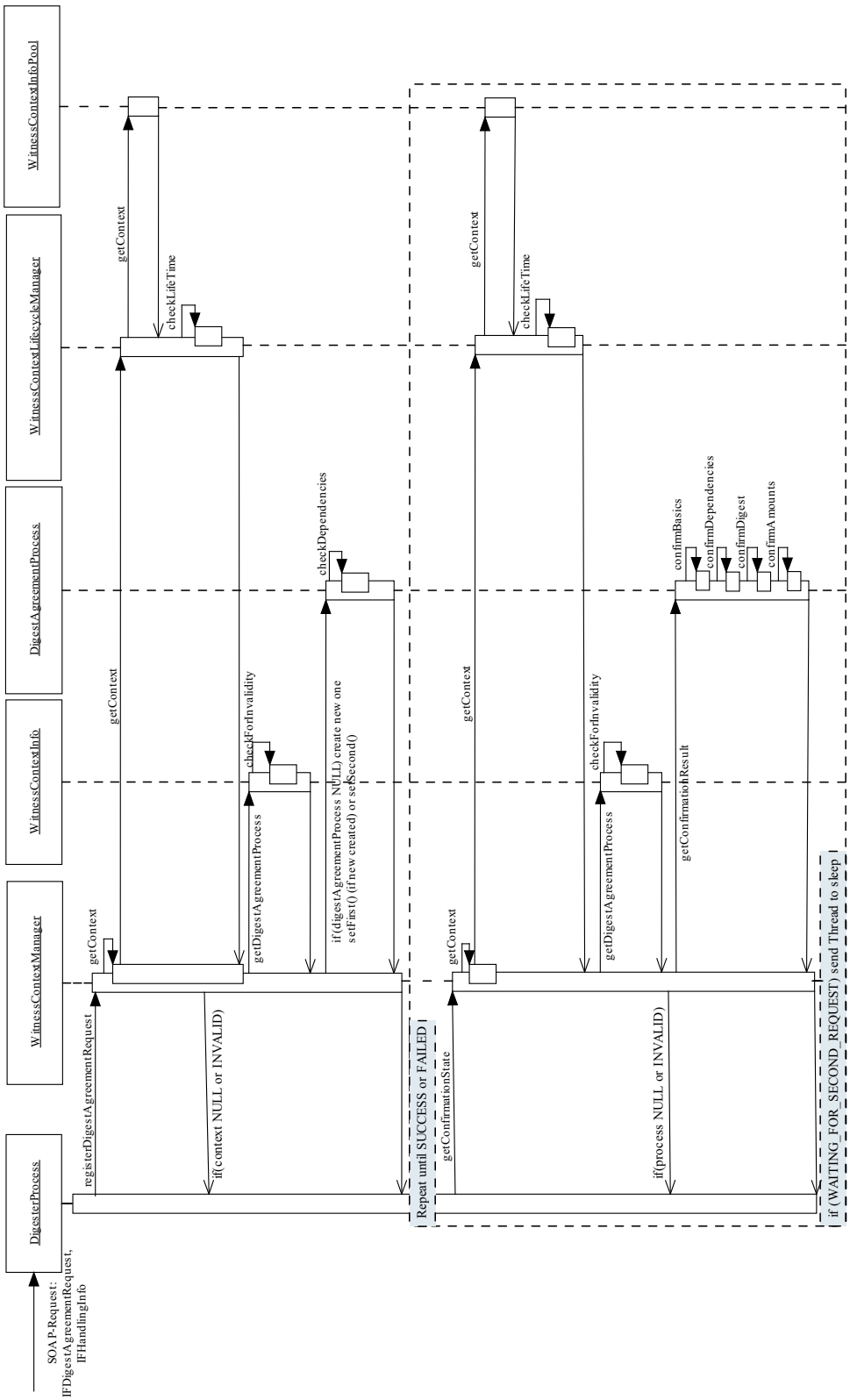


Abbildung 33: Sequenzdiagramm: DigestAgreement-Prozess beim Zeugen

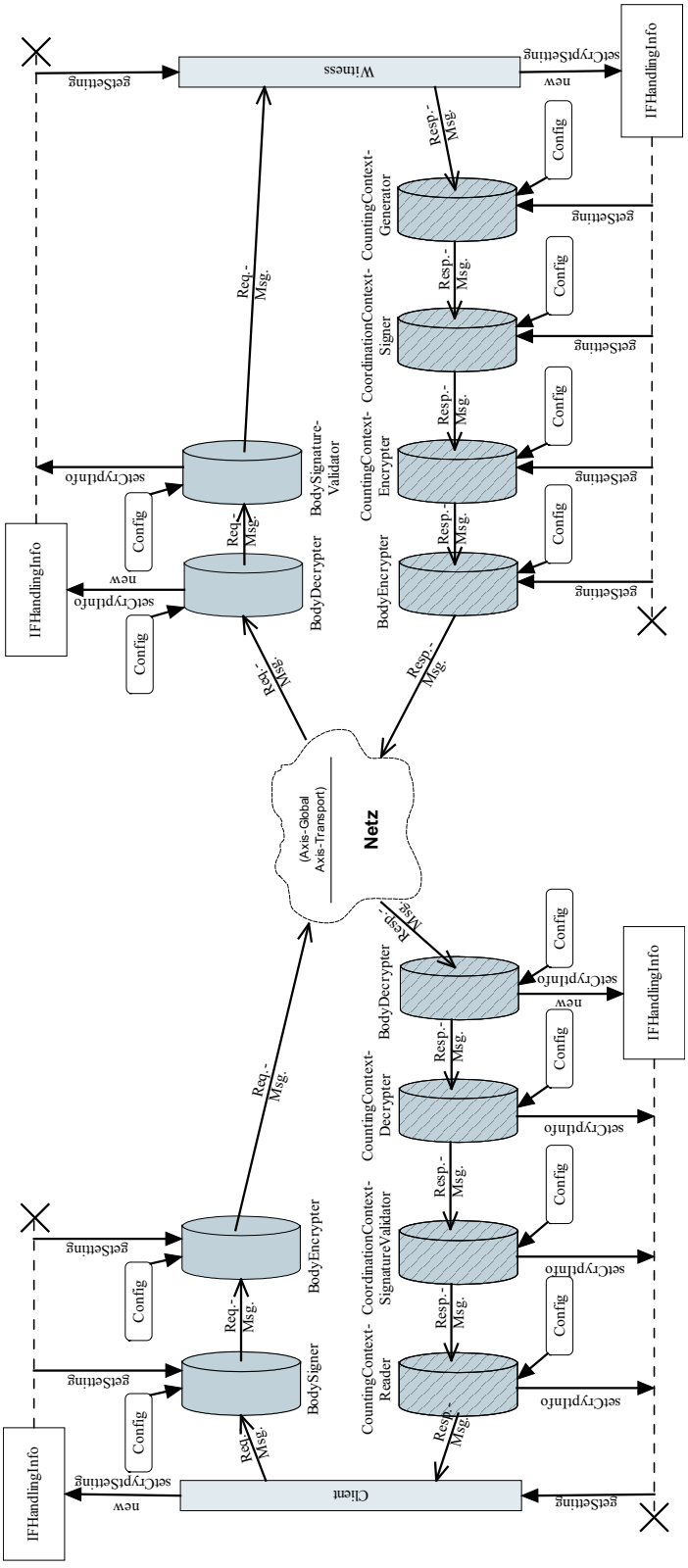


Abbildung 34: Handler-Kette: Request/Response zur Erzeugung eines Zähl-Kontextes (Klient-Zeuge)

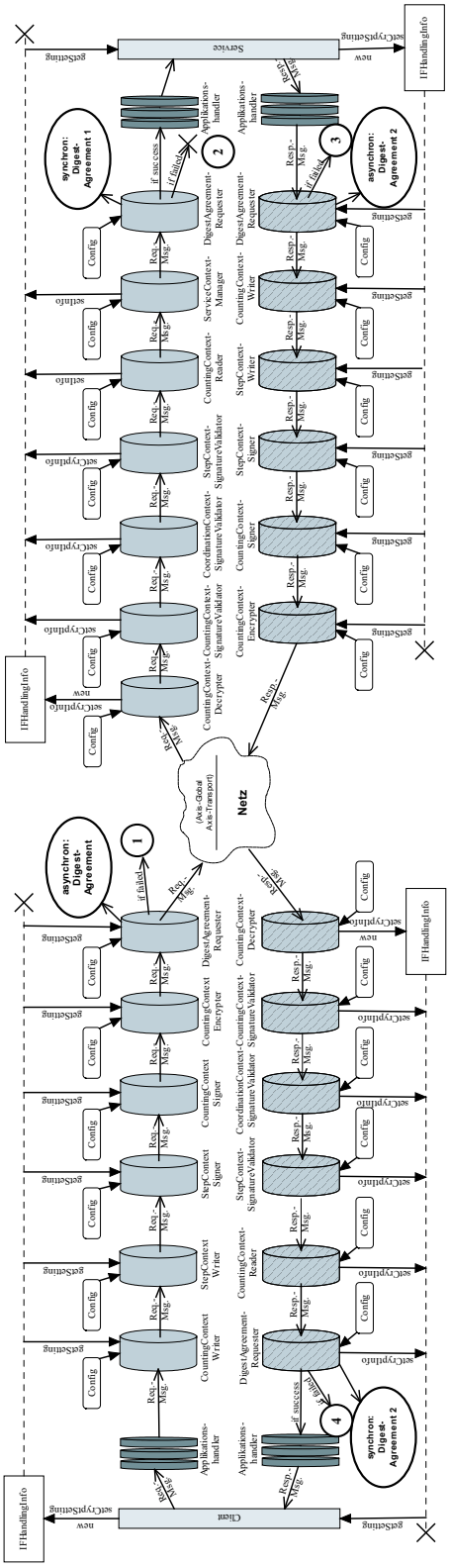


Abbildung 35: Handler-Kette: Request/Response für den Aufruf eines Dienstes (Klient-Service)

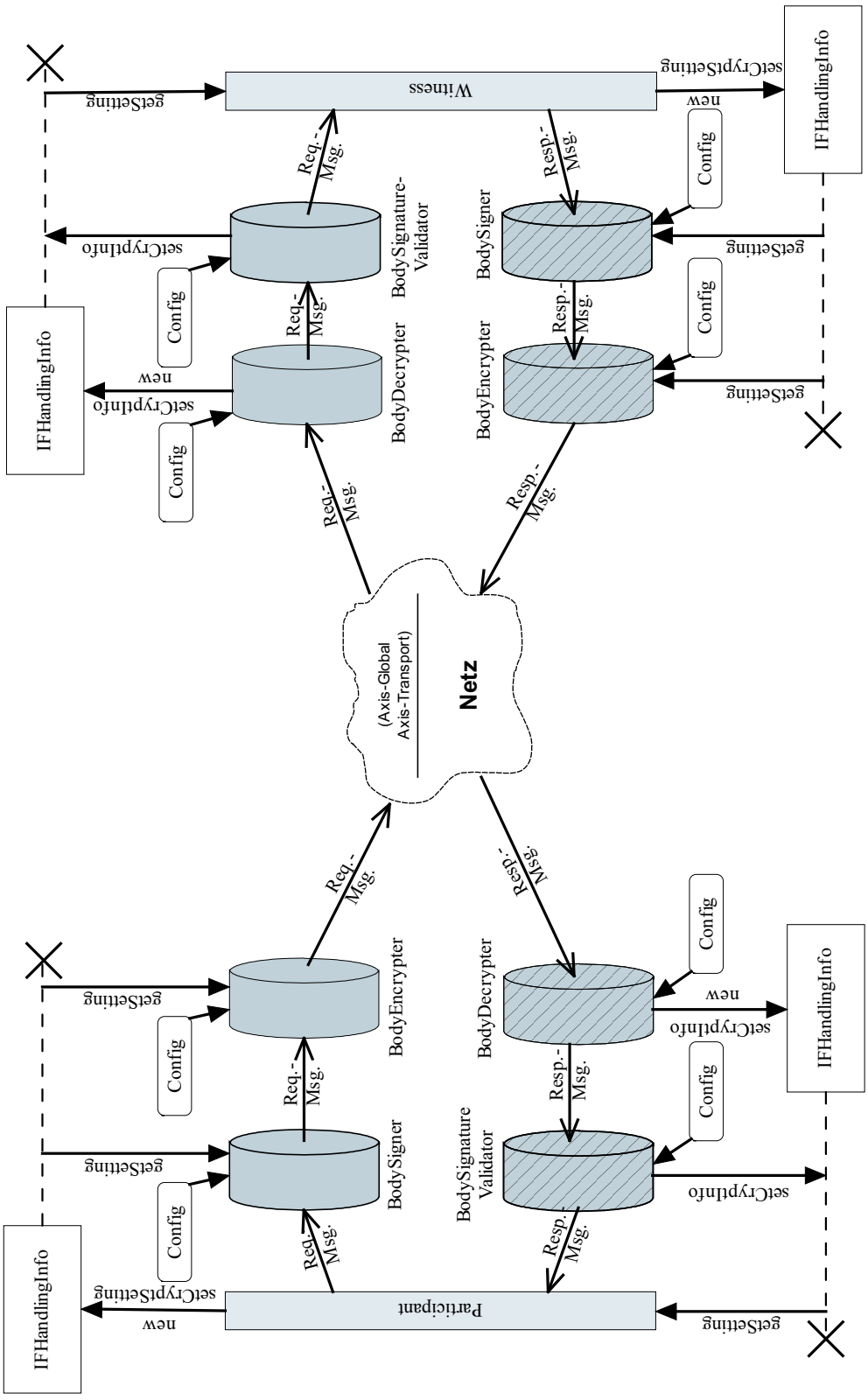


Abbildung 36: Handler-Kette: Request/Response für den Abgleich der Hash-Werte (Teilnehmer-Zeuge)

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der aufgeführten Literatur erstellt habe.

Dresden, den 11. Mai 2005