Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Thies Rubarth

Sicherheitskonzepte in global verteilten Anwendungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Thies Rubarth

Sicherheitskonzepte in global verteilten Anwendungen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Martin Hübner
Zweitgutachter : Prof. Dr. Kai von Luck

Abgegeben am 26. April 2007

Thies Rubarth

Thema der Masterarbeit

Sicherheitskonzepte in global verteilten Anwendungen

Stichworte

SOA, Sicherheit, Web Services, WS-Security, WS-Policy, WS-Trust, WS-Federation

Kurzzusammenfassung

Diese Arbeit untersucht die Anforderungen, die eine global verteilte Anwendung an ein Sicherheitsframework stellt. Auf Basis dieser Anforderungen und vorhandenen Spezifikationen wird ein Sicherheitsframework für global verteilte Anwendungen entworfen. Das Sicherheitsframework ist in der Lage beliebige Sicherheitsprotokolle umzusetzen, die mit den verwendeten Spezifikationen definiert werden können. Dazu müssen die Sicherheitsanforderungen der Anwendung den Spezifikationen entsprechend formal beschrieben werden. Das Framework setzt die Sicherheitsprotokolle anhand der beschriebenen Anforderungen dynamisch um.

Thies Rubarth

Title of the master thesis

Security concepts in global distributed applications

Keywords

SOA, Security, Web Services, WS-Security, WS-Policy, WS-Trust, WS-Federation

Abstract

In this thesis the requirements for a security framework for global distributed applications are analysed. Based on these requirements and existent specifications a suitable security framework is designed. The framework enables the applications to implement any user-defined security protocol defined with the used specifications. For this purpose the application's security requirements must be described in a formal way defined by the specifications. The framework dynamically executes the security protocol according to the described requirements.

Danksagung

Es gibt viele, die es mir ermöglicht haben diese Arbeit zu schreiben, sei es durch Aufmunterungen, Ermutigungen, motivierende Bestätigungen oder ähnliches. Einige haben jedoch besonders dazu beigetragen. Diesen möchte ich an dieser Stelle meinen Dank aussprechen.

Ich danke ich meinen Eltern, die mir es überhaupt ermöglichten zu Studieren. Im Besonderen danke ich meinem Vater, der mich nicht nur als Vorbild in meiner Studien- und Berufswahl geprägt hat, sondern mir schon früh den Einstieg in den Beruf ermöglichte, wo ich wertvolle Erfahrungen für das Studium sammeln konnte. Darüber hinaus danke ich ihm für seine wertvollen Anregungen und Anmerkungen zu dieser Arbeit.

Weiterhin danke ich meinen Betreuern Prof. Dr. Martin Hübner und Prof. Dr. Kai von Luck für die vielen Anregungen und Diskussionen und dafür, dass sie mich auf den Weg zu dieser Arbeit gebracht haben. Des Weiteren danke ich Ihnen auch für die guten Gespräche, die sich nicht um diese Arbeit drehten.

Ich danke ebenfalls meinen Kommilitonen Sven Stegelmeier, Mark Thomé und Piotr Wendt, die mir in den letzten Monaten sehr geholfen haben mich zu disziplinieren und meine Arbeit Korrektur gelesen haben.

Nicht zuletzt danke ich Gott, dem ich nicht nur mein Leben zu verdanken habe, sondern auch viele wertvolle Gaben, die mir das ermöglicht haben, was ich bis jetzt im Studium und im Beruf (und nicht nur dort) erreicht habe. Darüber hinaus hat er mir in den letzten Monaten immer wieder Kraft und wertvolle Geistesblitze geschenkt, die mir das Schreiben dieser Arbeit (mindestens) erleichtert haben.

Thies Rubarth, April 2007

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einführung

Im Laufe ihrer Entwicklung hat die Informationstechnologie zunehmend an Bedeutung für den geschäftlichen Erfolg von Unternehmen gewonnen. Welches Unternehmen „das Rennen macht“, wird nicht mehr allein dadurch bestimmt, ob es sein Geschäft am besten beherrscht, sondern wesentlich dadurch beeinflusst wie flexibel die IT-Prozesse des Unternehmens auf neue Anforderungen reagieren können.

Im Idealfall ermöglicht eine flexible IT die schnelle und einfache Anbindung von Partnerunternehmen, so dass eine unternehmensübergreifende Geschäftsprozessoptimierung möglich wird, bei der Aktivitäten, die das eigene Unternehmen nicht so effizient ausführen kann, von Partnerunternehmen übernommen werden (Vgl. Weerawarana u. a., 2005). Die Aufgaben und angebotenen Leistungen eines Unternehmens können somit leicht an die zur Verfügung stehenden Ressourcen und an die Nachfrage der Kunden angepasst werden.

Mit der Flexibilisierung und Öffnung der eigenen IT-Systeme einher geht ein wachsender Bedarf an Sicherheitsmechanismen, um die teilweise personenbezogenen und Geschäftsdaten vor dem Ausspionieren und der Manipulation durch Dritte zu schützen, da die Anwendungen nicht mehr in einem abgeschotteten Netzwerk laufen, sondern aus Diensten unterschiedlicher Unternehmen bestehen, die über das Internet miteinander kommunizieren.

Um Dienste anderer Unternehmen schnell und effizient in die eigenen Prozesse integrieren zu können, werden standardisierte Schnittstellen benötigt. Insbesondere werden Standards für die Sicherheit benötigt, damit die Anwendungen mit möglichst geringem Aufwand sicher gemacht werden können.

1.1 Motivation

Die Softwarearchitektur hat sich im Laufe ihrer Evolution von monolithischen Stand-Alone-Systemen zu immer loser gekoppelten Systemen hinentwickelt (Vgl. Woods, 2003). Mit der Entwicklung zu immer komplexeren Systemen wuchsen auch die Ansprüche an die Sicherheit. Waren die Anwendungen zunächst nur durch physikalischen Zugriff auf den Rechner zu erreichen, stieg mit dem Einsatz von lokalen Netzwerken innerhalb der Organisationen auch die Zahl der Angriffsmöglichkeiten, wobei für einen Angriff auf die Anwendungen immer noch

ein physikalischer Zugang zu dem Netzwerk nötig war. Organisationsübergreifender Datenaustausch über öffentliche Leitungen fand in der Regel nur zwischen bekannten Partnern statt, die sich untereinander über das Format und die notwendigen Sicherheitsvorkehrungen einigten.

Von der Koppelung auf Datenebene entwickelte sich die Softwarearchitektur hin zu einer Koppelung auf Ebene der Anwendungslogik, wodurch verteilte Systeme entstanden. Durch die Etablierung des Internets wurde die globale Verteilung von Anwendungen ermöglicht.

Der aktuelle Architekturansatz SOA[1] sieht Anwendungen als Kompositionen von unabhängigen Diensten. Diese Dienste stellen Geschäftslogik zur Verfügung, bieten eine offene und maschinenlesbare Schnittstelle und sind mit einfachen Standardtechnologien erreichbar. Durch diese Eigenschaften können Dienste leicht in Anwendungen integriert oder durch andere Dienste ersetzt werden. Insbesondere ist es möglich, Dienste anderer Unternehmen zu verwenden bzw. eigene Dienste anderen Unternehmen zur Verfügung zu stellen. Der aktuelle Standard zur Bereitstellung von Diensten sind Webservices. Webservices basieren auf einfachen Standards wie XML[2] und HTTP[3] und können somit über das Internet weltweit angeboten und genutzt werden.

In einer auf Webservices basierenden Anwendung stellen sich völlig neue Anforderungen an die Sicherheit, da die Kommunikation über ein öffentliches Netzwerk läuft und die zunehmende Nutzung des Internets als IT-Infrastruktur in den letzten Jahren zu einer deutlichen Vermehrung der Angriffe von außerhalb der Unternehmen geführt hat. Die Einrichtung eines VPN[4] zum Schutz gegen externe Angriffe ist nicht möglich, wenn es darum geht, Dienste öffentlich für jedermann anzubieten.

Auch bei global verteilten Anwendungen, bei denen die kooperierenden Partner bekannt sind und somit die Einrichtung eines VPN möglich ist, müssen zusätzliche Maßnahmen für die Sicherheit der Anwendung getroffen werden, da Studien zeigen (z.B. Gordon u. a., 2006), dass Angriffe von internen Mitarbeitern nach wie vor eine der größten Bedrohungen im Bereich der IT-Sicherheit sind. Die gängige Praxis, die Anwendungen im internen Netz ungeschützt zu lassen, ist deshalb ein nicht zufriedenstellender Zustand.

Um die Kommunikation von Webservices gegenüber Angriffen abzusichern, gibt es bereits eine Vielzahl an Spezifikationen. Einige dieser Spezifikationen befassen sich nur mit einzelnen Aspekte von Sicherheit, während andere Spezifikationen versuchen einen Großteil des benötigten Funktionsumfangs abzudecken. Die große Zahl an Spezifikationen und deren Überschneidungen im Funktionsumfang erschweren die Auswahl für ein konkret zu erstellendes Sicherheitskonzept.

[1] Service Oriented Architecture
[2] EXtensible Markup Language
[3] Hypertext Transfer Protocol
[4] Virtual Private Network

Die Spezifikationen allein oder in Kombination reichen zum Erstellen einer geschlossenen Sicherheitsarchitektur nicht aus, da sie nur grundlegende Techniken wie z.B. Verschlüsselung und Signierung zur Verfügung stellen. Welche Maßnahmen konkret getroffen werden müssen, um die Sicherheit einer Anwendung zu garantieren, bleibt offen. Nicht alle Sicherheitsprobleme, die in einer Anwendung auftreten können, werden durch die Spezifikationen abgedeckt. In einer konkreten Anwendung muss deshalb analysiert werden, welche Sicherheitsprobleme mit den Spezifikationen gelöst werden können und bei welchen Problemen zusätzliche Maßnahmen notwendig sind.

In den Spezifikationen ist nur das Verhalten der Dienste nach außen festgelegt. Wie die Spezifikationen in einer konkreten Anwendung zu implementieren sind, bleibt offen. Um die Fehler beim Anwenden der Spezifikationen so gering wie möglich zu halten, wird ein Framework benötigt, dass die Spezifikationen umsetzt und leicht in andere Anwendungen eingebunden werden kann (Vgl. Viega und Epstein, 2006).

1.2 Zielsetzung

Ziel dieser Arbeit ist der Entwurf eines Frameworks, das diejenigen Sicherheitsmaßnahmen für global verteilte Anwendungen zur Verfügung stellt, die von den bestehenden Spezifikationen abgedeckt werden. Bei der Betrachtung beschränkt sich diese Arbeit auf global verteilte Anwendungen, die auf dem Architekturansatz SOA und der Technologie Webservices basieren. Für eine solche Anwendung wird eine Sicherheitsarchitektur entworfen, die festlegt, welche Maßnahmen zur Gewährleistung der Sicherheit notwendig sind.

Das Framework übernimmt in der Sicherheitsarchitektur die Funktionalität, die für die Anwendungssicherheit notwendig ist, wie z.B. das Verschlüsseln und Signieren von Daten, deren Überprüfung und das Ausführen von Sicherheitsprotokollen, (z.B. Challenge-Response-Verfahren[5]).

Das Framework bietet Sicherheit für global verteilte Anwendungen „Out-of-the-box“ an, d.h. es kann leicht in eine Anwendung integriert werden, um in dieser die klassischen Schutzziele Vertraulichkeit und Integrität zu gewährleisten. Das Framework sorgt für die Sicherheit der Webservice-Aufrufe beispielsweise dadurch, dass die Nachrichten auf der Seite des Aufrufers entsprechend den Anforderungen des Webservice verschlüsselt und signiert werden. Auf der Seite des aufgerufenen Dienstes übernimmt das Framework z.B. das Entschlüsseln der Nachricht und die Überprüfung der Signatur zur Authentifikation und Autorisierung des Nutzers.

[5] Bei Challenge-Response-Verfahren stellt der aufgerufene Dienst eine „Gegenfrage“, mit der z.B. überprüft wird, ob der Aufrufer wirklich derjenige ist, für den er sich ausgibt.

1.3 Vorgehensweise

Verschiedene Sicherheitsprobleme, die in einer global verteilten Anwendung auftreten, werden anhand eines konkreten Anwendungsszenarios diskutiert. Daraus werden die Anforderungen an das Framework abgeleitet.

Anhand der Anforderungen und der Anwendungsfälle des Szenarios wird das Framework iterativ-inkrementell entwickelt. Dabei werden bestehende Webservice-Spezifikationen verwendet. Es wird eine Sicherheitsarchitektur für das Szenario entworfen, die zeigt, welche Rolle das Framework für die Sicherheit einer global verteilten Anwendung hat, und wie es mit den anderen Komponenten der Anwendung zusammenarbeitet.

Um zu zeigen, dass das Framework implementierbar ist und dass es in eine beliebige Anwendung, die den in dieser Arbeit gestellten Anforderungen entspricht, integriert werden kann, werden kritische Teile davon prototypisch implementiert.

1.4 Notation

In diesem Abschnitt werden die in der Arbeit verwendeten Notation kurz erläutert.

1.4.1 XML-Namespaces

In den XML-Beispielen werden aus Gründen der Übersichtlichkeit die URLs der verwendeten Namespaces weggelassen. Die verwendeten Kürzel für die Namespaces sind in Tabelle 1.1 dargestellt.

1.4.2 Sicherheitsprotokolle

Für die schematischen Darstellungen der Sicherheitsprotokolle wird eine Notation verwendet, die sich nach der in der BAN-Logik (Vgl. Burrows u. a., 1990) verwendeten Notation richtet. In Tabelle 1.2 ist die verwendete Notation dargestellt.

Kürzel	URL	Spezifikation
S	http://schemas.xmlsoap.org/soap/envelope/	SOAP
ds	http://www.w3.org/2000/09/xmldsig#	XML-Signature
enc	http://www.w3.org/2001/04/xmlenc#	XML-Encryption
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	WS-Security
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-*1.x*.xsd	WS-Security
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512	WS-Trust
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy
sp	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	WS-SecurityPolicy
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing
saml	urn: oasis:names:tc:SAML:1.0:assertion	SAML Assertions

Tabelle 1.1: Verwendete XML-Namespaces

Symbol	Bedeutung
A, B, S	Symbole für Principals
$K1$, $K2$, ...	Generierte symmetrische Schlüssel (nummeriert)
K_A	Öffentlicher Schüssel von A
K_A^{-1}	Privater Schlüssel von A
K_{AB}	Geheimer Schlüssel von A und B
$\{X\}_K$	X ist mit K verschlüsselt
$\langle X \rangle_K$	X ist mit K signiert
$A \rightarrow B : X$	A sendet an B die Nachricht X

Tabelle 1.2: Formale Notation für Sicherheitsprotokolle

2 Grundlagen

In diesem Kapitel werden die grundlegenden Technologien vorgestellt, die zum weiteren Verständnis der Arbeit notwendig sind. Zu diesen Technologien gehören SOA[1] und Webservices, mit denen global verteilte Anwendungen realisiert werden können, sowie Kryptographie und Authentifikationskonzepte, mit denen Anwendungen sicher gemacht werden können.

2.1 Service Oriented Architecture

Aus Sicht der Geschäftslogik, bietet eine SOA die Dienste einer Organisation zur einfachen Nutzung und Integration in Geschäftsprozesse an. Die Dienste einer SOA stellen in sich geschlossene Geschäftsmethoden dar, die keine weitere Verarbeitung benötigen. Dadurch können die Dienste beliebig verwendet und kombiniert werden, um eine Anwendung an neue Geschäftsanforderungen anzupassen.

Technisch stellen sich diese Dienste als lose gekoppelte Anwendungskomponenten dar, deren Schnittstellen universell beschrieben sind und die plattformunabhängig genutzt werden können. Durch die universelle Beschreibung der Schnittstelle ist jeder, der Zugang zu dem Dienst bekommt, in der Lage diesen zu Nutzen und in seine Anwendung zu integrieren. Durch die Plattformunabhängigkeit spielt es dabei keine Rolle, was für ein Gerät, welches Betriebssystem oder welche Programmiersprache dabei verwendet wird (Vgl. Bieberstein u. a., 2006).

2.2 Webservices

Während die SOA ein Architekturkonzept für lose gekoppelte Anwendungen ist, sind Webservices ein Ansatz eine SOA zu implementieren und Dienste interoperabel anzubieten. Das W3C definiert Webservices wie folgt (Vgl. Booth u. a., 2004):

[1] Service Oriented Architecture

> Ein Webservice ist ein Softwaresystem, dass darauf ausgelegt ist die netzwerkbasierte und interoperable Interaktion zwischen Maschinen zu unterstützen. Die Schnittstelle eines Webservice ist maschinenlesbar beschrieben (WSDL[2]). Andere Systeme interagieren mit dem Webservice seiner Beschreibung entsprechend mit SOAP[3]-Nachrichten, die in der Regel in XML serialisiert sind und über HTTP[4] in Verbindung mit anderen Internetstandards versendet werden.

Die drei wesentlichen Protokolle für Webservices sind SOAP, WSDL und UDDI[5]. SOAP ist ein Protokoll, dass den interoperablen RPC[6]-Aufruf mittels XML-Serialisierung unterstützt. WSDL dient der maschinenlesbaren Beschreibung der Schnittstelle eines Webservices. UDDI stellt einen Verzeichnisdienst für Webservices zur Verfügung über den geeignete Webservices gefunden werden können.

Um eine qualitativ hochwertige SOA mit Webservices zu realisieren, reichen die drei Basisprotokolle nicht aus. Es existieren zahlreiche Zusatzspezifikationen, mit denen Sicherheit, Transaktionen und andere Aspekte der Dienstgüte erreicht werden können. In Abbildung 2.1 ist dargestellt, in welche Bausteine diese Spezifikationen eingeteilt werden. Wie in der Abbildung angedeutet wird, existieren für die einzelnen Bausteine nicht jeweils eine Spezifikation, sondern mehrere Teilspezifikationen. Insgesamt gibt es weit über hundert verschiedene Spezifikationen, was es schwer macht den Überblick über die benötigten Spezifikationen und deren Funktionalität zu behalten.

Für einige Bausteine gibt es auch mehrere Spezifikationen, die sich in ihrer Funktionalität überschneiden. Das macht den Weg zur Standardisierung schwerer, so dass die meisten WS-*-Spezifikationen noch nicht standardisiert sind. Dennoch existieren für einzelne Spezifikationen bereits Implementierungen, die jeweils die Funktionalität der aktuellsten Version realisieren (Vgl. Frotscher u. a., 2007).

Für den Bereich WS-Security gibt es folgende Teilspezifikationen.

- SOAP Message Security
- WS-Trust
- WS-SecurityPolicy
- WS-Federation
- WS-SecureConversation

[2] Web Service Definition Language
[3] Simple Object Access Protocol
[4] Hyper Text Transfer Protocol
[5] Universal Description Discovery and Integration
[6] Remote Procedure Call

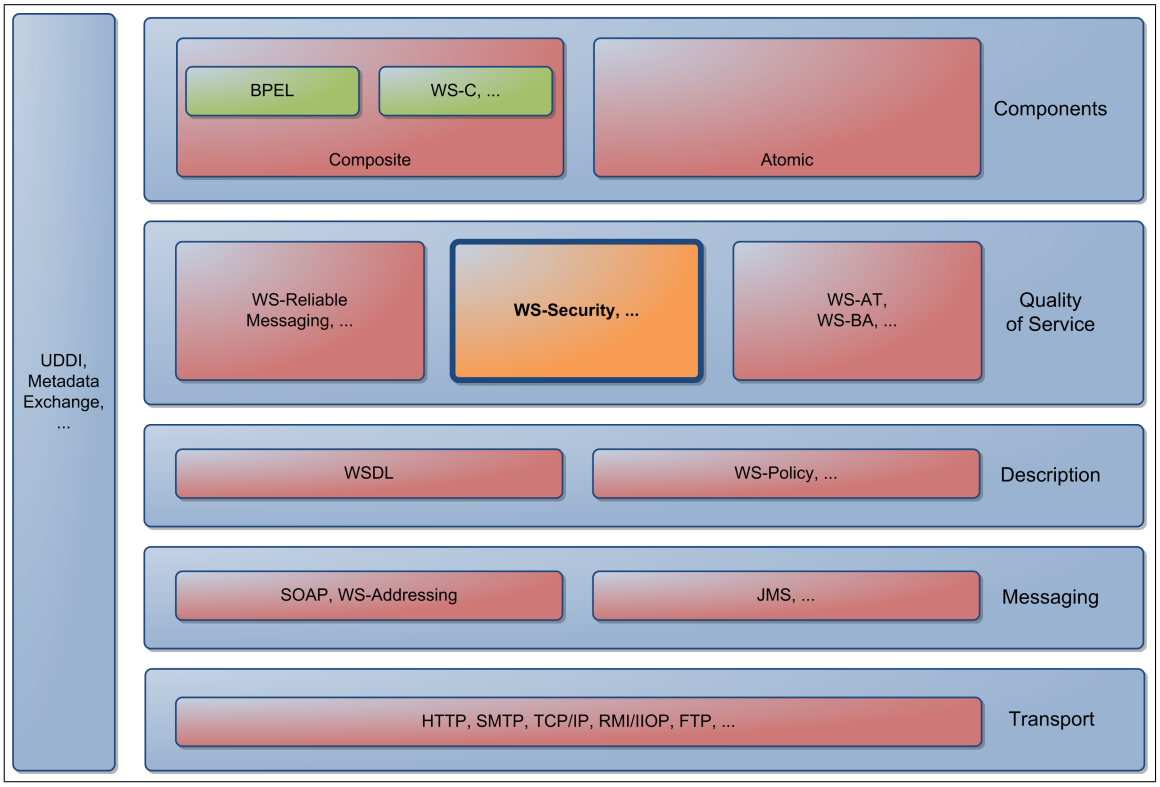

Abbildung 2.1: Webservice-Architektur (Vgl. Weerawarana u. a., 2005)

- (WS-Authorization)
- (WS-Privacy)

WS-Authorization und WS-Privacy werden im Übersichtsplan der WS-Security-Spezifikationen beschrieben, sind aber noch nicht in Arbeit[7].

Darüber hinaus gibt es noch weitere Spezifikationen, die nicht zu der WS-Security-Road-Map gehören, aber für die Sicherheit von Webservices relevant sind (Vgl. Naedele, 2003):

- XML-Encryption
- XML-Signature
- SAML[8]
- XACML[9]

[7] Aussage von Frank Leymann bei einem Vortrag bei der GI Localgroup im September 2006
[8] Security Assertion Markup Language
[9] eXtensible Access Control Markup Language

- XKMS[10]

Diese Spezifikationen können teilweise separat oder in Verbindung mit den entsprechenden WS-Security-Spezifikationen eingesetzt werden. SAML, XACML, XKMS werden in dieser Arbeit nicht betrachtet, da sie nicht zu den WS-*-Spezifikationen gehören und sich mit deren Funktionalität überschneiden.

2.2.1 SOAP Message Security

SOAP Message Security ist ein OASIS-Standard, der aus der ursprünglichen Spezifikation WS-Security entstanden ist. Dieser Standard definiert, wie Webservice-Nachrichten durch Anreicherung mit Security-Tokens sicher gemacht werden können und wie SOAP-Nachrichten oder einzelne Teile davon verschlüsselt und signiert werden können. Für die Verschlüsselung und die Signierung werden die Spezifikationen XML-Encryption und XML-Signature verwendet (Vgl. Lawrence und Kaler, 2006c).

Im wesentlichen werden die folgenden Security-Tokens von der Spezifikation definiert:

Username-Token Mit einem Username-Token kann ein Benutzer seinen Benutzernamen und ggfs. sein Passwort angeben (Vgl. Lawrence und Kaler, 2006d).

X509-Token Der X509-Token definiert, wie X.509-Zertifikate in Webservicenachrichten eingebunden und zur Signierung und Verschlüsselung verwendet werden können (Vgl. Lawrence und Kaler, 2006e).

Kerberos-Token Der Kerberos-Token ermöglicht die Verwendung von Kerberos-Tickets bei Webserviceaufrufen (Vgl. Lawrence und Kaler, 2006a).

SAML-Token Der SAML-Token ermöglicht es SAML-Assertions, wie Authentifizierungs- und Autorisierungsinformationen ohne Verwendung des SAML-Protokolls in Webservice-Nachrichten zu verwenden.

Darüber hinaus erlaubt es SOAP Message Security, eigene Security-Tokens zu definieren. Die Spezifikation legt nicht fest, welche Security-Tokens benötigt werden und wie Verschlüsselung und Signierung eingesetzt werden müssen, damit eine Nachricht sicher ist (Vgl. Lawrence und Kaler, 2006b).

[10] XML Key Management Specification

2.2.2 WS-Trust

WS-Trust definiert ein Vertrauensmodell für Webservices (siehe Abbildung 2.2). Das Vertrauensmodell basiert auf dem Austausch von Security-Tokens, die explizit oder implizit Zusicherungen über bestimmte Eigenschaften (Claims) ihres Besitzers machen. Jedes Subjekt kann in seiner Policy definieren, welche Security-Tokens von einem Kommunikationspartner erwartet werden.

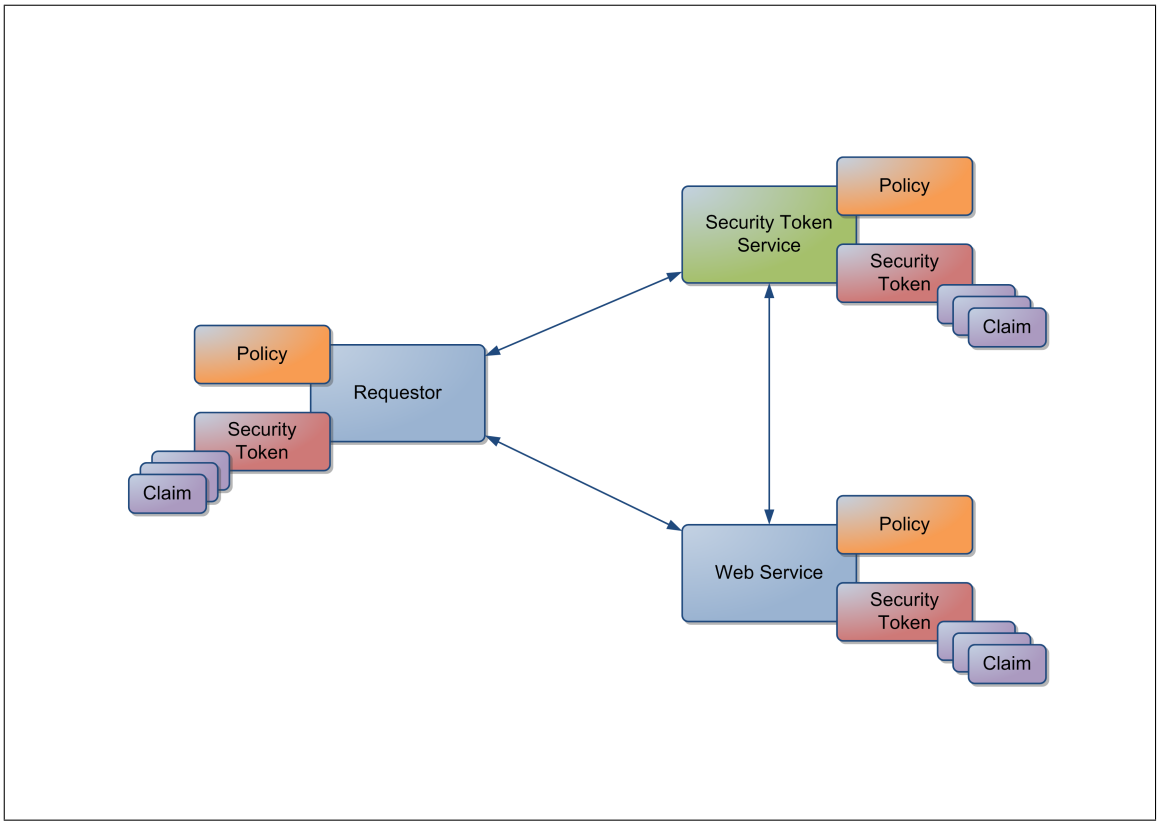

Abbildung 2.2: Vertrauensmodell von WS-Trust (Vgl. Lawrence und Kaler, 2006g)

WS-Trust definiert, wie Security-Tokens erstellt, ausgetauscht, erneuert und überprüft werden können. Eine wesentliche Rolle übernimmt dabei der Security Token Service (STS), der für das Ausstellen von Security-Tokens zuständig ist. Über den STS besteht die Möglichkeit Vertrauen über mehrere Vertrauensdomänen hinweg aufzubauen.

WS-Trust definiert nicht, welche Security-Tokens benötigt werden, um Vertrauen herzustellen (Vgl. Lawrence und Kaler, 2006g).

2.2.3 WS-Policy

WS-Policy definiert eine Sprache, die es ermöglicht, nicht-funktionale Anforderungen und Fähigkeiten eines Webservice zu beschreiben. Dabei kann eine Policy aus mehreren Policy-Alternativen bestehen, die wiederum aus einzelnen Assertions bestehen. Eine Assertion kann wiederum eine Policy enthalten. Die Auswahlmöglichkeiten aus den Policy-Alternativen werden mit den XML-Elementen *wsp:ExactlyOne* und *wsp:All* angegeben. Da es verschiedene Ausdrucksmöglichkeiten für komplexe Policies wie z.B. $(Assertion1 \wedge (Assertion2 \vee Assertion3))$ gibt, führt das zu Schwierigkeiten beim Vergleichen von Policies. Zu diesem Zweck legt die Normalform für Policies fest, dass diese aus einem *wsp:ExactlyOne*-Element bestehen müssen, das ausschließlich *wsp:All*-Elemente enthalten darf. In Listing 2.1 ist das genannte Beispiel in der Normalform zu sehen (Vgl. Bajaj u. a., 2006).

```
1  <wsp:Policy>
2    <wsp:ExactlyOne>
3      <wsp:All>
4        <Assertion1 />
5        <Assertion2 />
6      </wsp:All>
7      <wsp:All>
8        <Assertion1 />
9        <Assertion3 />
10     </wsp:All>
11   </wsp:ExactlyOne>
12 </wsp:Poilcy>
```

Listing 2.1: Schachtelung von Policies

2.2.4 WS-SecurityPolicy

WS-SecurityPolicy ist eine Erweiterung von WS-Policy, die bestimmte Assertions für Sicherheitsanforderungen definiert. Diese Assertions beziehen sich auf die Spezifikationen SOAP Message Security, WS-Trust und WS-SecureConversation und können verwendet werden, um vorzugeben, wie die Spezifikationen für einen Webservice einzusetzen sind (Vgl. Lawrence und Kaler, 2007).

2.2.5 WS-SecureConversation

WS-SecureConversation legt fest, wie Sicherheitskontexte erstellt werden können, die über mehrere Nachrichten hinweg verwendet werden können, so dass sich die Kommunikationspartner nicht mit jeder Nachricht neu authentifizieren müssen. Insbesondere wird definiert, wie ein solcher Sicherheitskontext mit WS-Trust erstellt und verteilt werden kann.

Um einen Sicherheitskontext zu erstellen, müssen die beteiligten Webservices ein gemeinsames Geheimnis austauschen, das an den Kontext gebunden wird. Dieses Geheimnis kann dann entweder direkt als Schlüssel für die Verschlüsselung und Signierung der Webservice-Nachrichten verwendet werden oder es können die benötigten Schlüssel daraus abgeleitet werden. Wie die Schlüssel abgeleitet werden können, wird ebenfalls durch WS-SecureConversation definiert (Vgl. Lawrence und Kaler, 2006f).

2.2.6 WS-Federation

In einem globalen Anwendungsszenario spielt FIM[11] eine zunehmend größer werdende Rolle. FIM beschäftigt sich mit der Verteilung von Benutzerinformationen[12] über mehrere Organisationen. WS-Federation legt fest, wie dies mit den WS-Security-Spezifikationen erreicht werden kann und wie ein organisationsübergreifendes Single-Sign-On (und Single-Sign-Off) realisiert werden kann (Vgl. Lockhart u. a., 2006).

2.2.7 XML-Encryption & XML-Signature

XML-Encryption ist keine Spezifikation, die nur für Webservices eingesetzt werden kann, sondern definiert im allgemeinen, wie XML-Dokumente oder Teile davon verschlüsselt werden können. Dazu wird festgelegt, wie verschlüsselte Daten in XML-Dokumenten dargestellt werden können und wie die Informationen, die zum Entschlüsseln der Daten benötigt werden (wie z.B. der verwendete Algorithmus), angegeben werden (Vgl. Eastlake und Reagle, 2002).

XML-Signature spezifiziert, wie XML-Dokumente oder Teile davon signiert werden können. Dazu wird festgelegt wie die Zertifikate in das XML-Dokument eingebunden werden und wie die Informationen, die notwendig sind um das Zertifikat zu überprüfen (wie z.B. die verwendeten Algorithmen), angegeben werden (Vgl. Eastlake u. a., 2002).

2.3 Kryptographie

Die Kryptographie ist die wichtigste Technologie, die für sichere Datenübertragung benötigt wird, insbesondere wenn ein nicht-abhörsicheres Netz (wie z.B. das Internet) für die Übertragung verwendet wird. Hier wird nur ein kurzer Überblick über die wesentlichen kryptographi-

[11] Federated Identity Management

[12] Abgeleitet von der Englischen Bezeichnung wird dies auch als „Föderieren von Entitäten“ bezeichnet.

schen Verfahren gegeben. Die detaillierten Funktionsweisen und Algorithmen können z.B. bei Eckert (2006) nachgelesen.

2.3.1 Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung wird eine Klartextnachricht mit einem geheimen Schlüssel verschlüsselt. Um die Nachricht entschlüsseln zu können, wird derselbe Schlüssel benötigt. Eine wichtige Aufgabe bei Verwendung symmetrischer Verschlüsselung ist daher der sichere Austausch des geheimen Schlüssels.

2.3.2 Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung erfolgen die Ver- und Entschlüsselung jeweils mit einem anderen Schlüssel. Dazu wird ein zueinander passendes Schlüsselpaar benötigt. Einer der beiden Schlüssel wird als „öffentlicher Schlüssel“ bezeichnet. Wie der Name schon sagt, ist dieser Schlüssel öffentlich bekannt und wird zum Verschlüsseln einer Nachricht verwendet. Um die Nachricht entschlüsseln zu können, wird der „private Schlüssel“ des Schlüsselpaares benötigt.

Der Vorteil der asymmetrischen Verschlüsselung ist, dass man keinen geheimen Schlüssel austauschen muss. Der Nachteil der asymmetrischen Verschlüsselung ist die deutlich höhere Rechenzeit, die asymmetrische Verschlüsselungsalgorithmen benötigen. In der Praxis werden die beiden Verfahren deshalb kombiniert eingesetzt. Dazu wird ein zufälliger symmetrischer Schlüssel generiert, der zum Verschlüsseln der Klartextnachricht verwendet wird. Dieser Schlüssel wird mit dem öffentlichen Schlüssel des Empfängers asymmetrisch verschlüsselt und zusammen mit der verschlüsselten Nachricht übertragen. Der verschlüsselte Schlüssel wird im Folgenden als Hybrid-Key bezeichnet. Der Empfänger kann mit seinem privaten Schlüssel den Hybrid-Key entschlüsseln und ist somit in der Lage den Rest der Nachricht zu entschlüsseln. Dieses Verfahren wird im Folgenden als Hybrid-Verfahren bezeichnet.

2.3.3 Elektronische Signaturen

Elektronische Signaturen dienen dem Beweis des Ursprungs einer Nachricht. Dazu wird z.B. die asymmetrische Verschlüsselung „in umgekehrter Reihenfolge“ verwendet. Aus der Klartextnachricht wird mit einem festgelegten Algorithmus zunächst ein Message Digest[13]

[13] Vergleichbar mit einem Hashwert

gebildet. Dieser Message Digest wird mit dem privaten Schlüssel des Absender verschlüsselt. Die Klartextnachricht wird dann zusammen mit dem verschlüsselten Message Digest an den Empfänger gesendet. Dieser bildet mit demselben Algorithmus, wie der Absender, den Message Digest der Nachricht. Dann entschlüsselt er den verschlüsselten Message Digest mit dem öffentlichen Schlüssel des Absenders. Stimmen beide Message Digests überein, ist sichergestellt, dass der Absender der wirkliche Verfasser der Klartextnachricht ist.

2.4 Authentifikation

In organisationsübergreifenden Szenarien reicht eine einfache Authentifizierung über Benutzername und Passwort meistens nicht aus, da nicht mehr jeder jeden kennen kann. Es werden daher Mechanismen benötigt, mit denen Angaben über die Authentizität eines Subjektes verteilt werden können. Die zwei bedeutendsten Konzepte dafür sind PKI[14] und Kerberos.

2.4.1 Public Key Infrastructure

Bei einer PKI belegen Zertifikate die Gültigkeit eines öffentlichen Schlüssels. Der Schlüssel wird damit an eine natürliche oder juristische Person gebunden. Die Zertifikate werden von einem Trust Center ausgestellt. Das Trust Center signiert das ausgestellte Zertifikat mit seinem privatem Schlüssel. Mit dem öffentlichen Schlüssel aus dem Zertifikat des Trust Centers kann diese Signatur überprüft werden. Um die Gültigkeit eines Zertifikates zu prüfen, muss also die Gültigkeit des Zertifikats des Trust Centers sichergestellt sein.

Damit nicht jeder die Zertifikate aller Trust Center kennen muss, sind die Trust Center in einer Hierarchie angeordnet (siehe Abbildung 2.3). Ausgangspunkt der Hierarchie ist die IPRA[15]. Die IPRA zertifiziert die PCAs[16], die jede eine eigene Strategie zum Vergeben von Zertifikaten definieren können. Die PCAs wiederum zertifizieren die CAs[17], die entweder noch weiter untergliedert sind oder direkt die Benutzer zertifizieren.

Um ein Zertifikat zu validieren, das aus einem anderen Vertrauensbereich der PKI stammt, muss ein Pfad durch die Hierarchie gefunden werden, der sicherstellt, dass das Zertifikat gültig ist. Dabei können auch Cross-Zertifikate verfolgt werden, mit denen eine CA die Vertrauenswürdigkeit einer anderen CA zertifiziert (Vgl. Eckert, 2006).

[14] Public Key Infrastructure
[15] Internet Policy Registration Authority
[16] Policy Certification Authorities
[17] Certification Auhorities

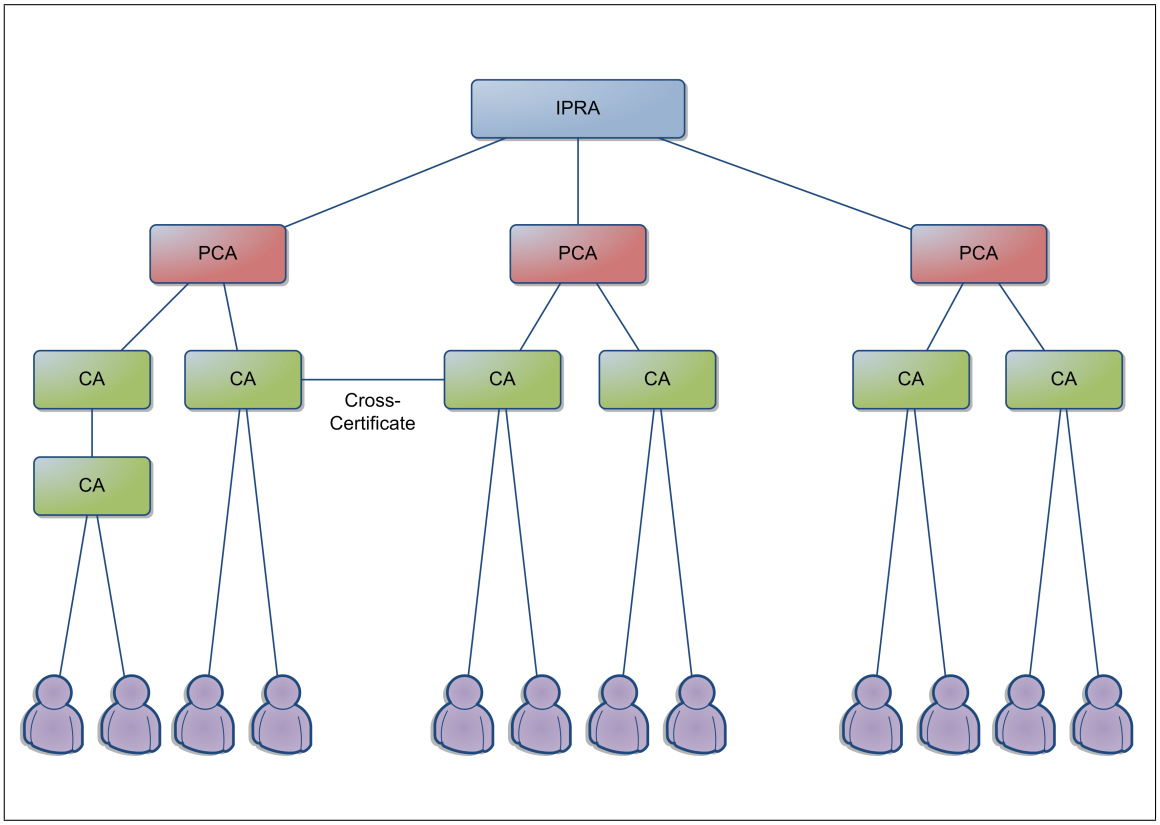

Abbildung 2.3: Darstellung einer PKI

Damit die Zertifikate organisationsübergreifend genutzt werden können, wurde ein Standard entwickelt, der festlegt, welche Daten in einem Zertifikat enthalten sein müssen. Der aktuell genutzte Standard sind X.509-Zertifikate in der Version v3. Eine schematische Darstellung eines X.509-Zertifikats ist in Abbildung 2.4 zu sehen (Vgl. International Telecommunication Union, 2005).

Eine besondere Rolle bei der Verwendung von Zertifikaten spielen Smart-Cards. Smart-Cards sind Chipkarten, die nicht nur Daten speichern können sondern zusätzlich eine eigene CPU[18] und ein eigenes Betriebssystem haben. Der Chip und das Betriebssystem weisen diverse Sicherheitsmerkmale auf, so dass Smart-Cards gut geeignet sind um schützenswerte Daten auf ihnen zu speichern.

Bei einer Authentifikation mit Smart-Cards werden das Zertifikat des Benutzers und dessen privater Schlüssel auf der Smart-Card gespeichert. Während das Zertifikat von der Smart-Card gelesen werden kann, wird der private Schlüssel nicht preisgegeben. Die Ver- und Entschlüsselung und die Signierung von Daten mit dem privaten Schlüssel findet auf der

[18] Central Processing Unit

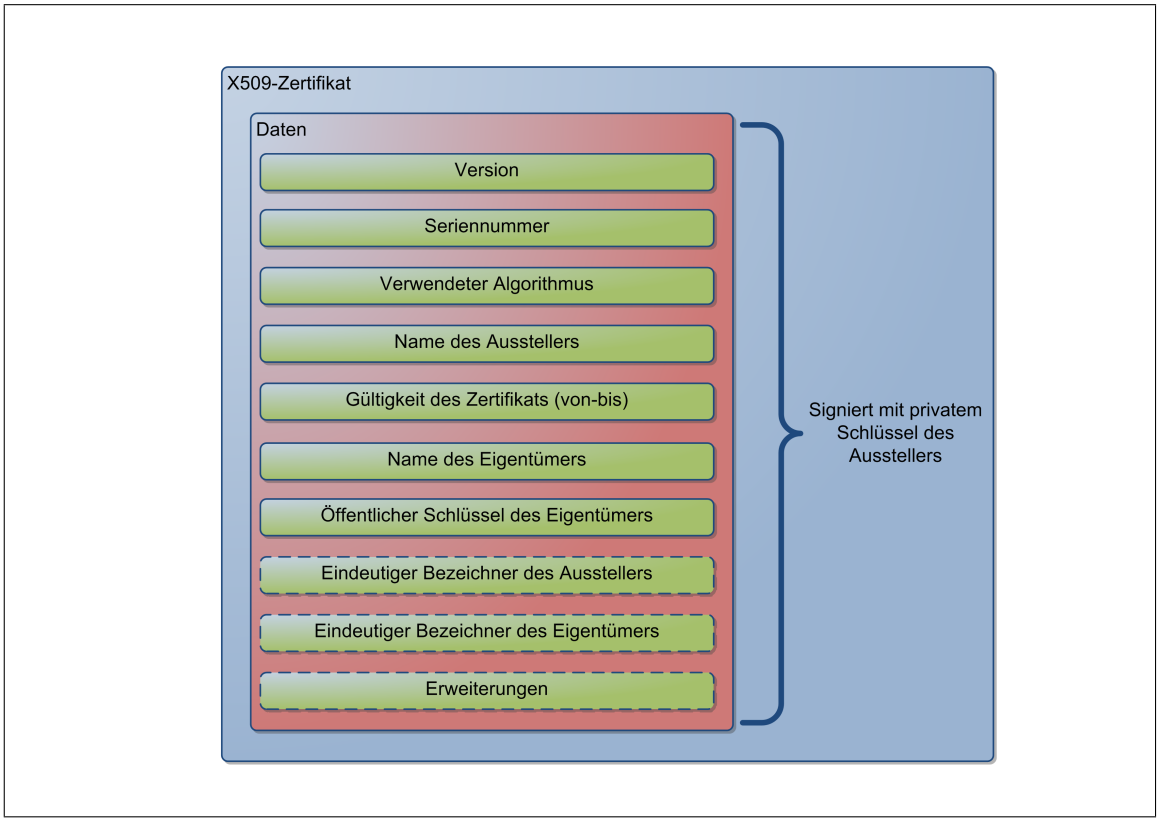

Abbildung 2.4: Schematische Darstellung eines X.509-Zertifikats.

Smart-Card selbst statt, die dafür eine Kryptographie-Schnittstelle zur Verfügung stellt. Damit die Smart-Card nicht von unberechtigten Benutzern verwendet werden kann, muss sie vor der Benutzung mit einer PIN[19] freigeschaltet werden. Alternativ zu der PIN kann auch eine Authentifikation über biometrische Merkmale verwendet werden.

2.4.2 Kerberos

Kerberos ist ein Konzept, das ursprünglich ohne asymmetrische Verschlüsselungsverfahren auskommt. Inzwischen gibt es aber auch Vorschläge, wie Kerberos mit asymmetrischen Verschlüsselungsverfahren kombiniert werden kann.

Die Basis in einem Kerberos-System stellt das KDC[20] dar (siehe Abbildung 2.5). Das KDC besteht aus dem AS[21] und dem TGS[22]. Meldet sich ein Benutzer am System an, muss er

[19] Personal Identification Number
[20] Key Distribution Center
[21] Authentication Server
[22] Ticket Granting Service

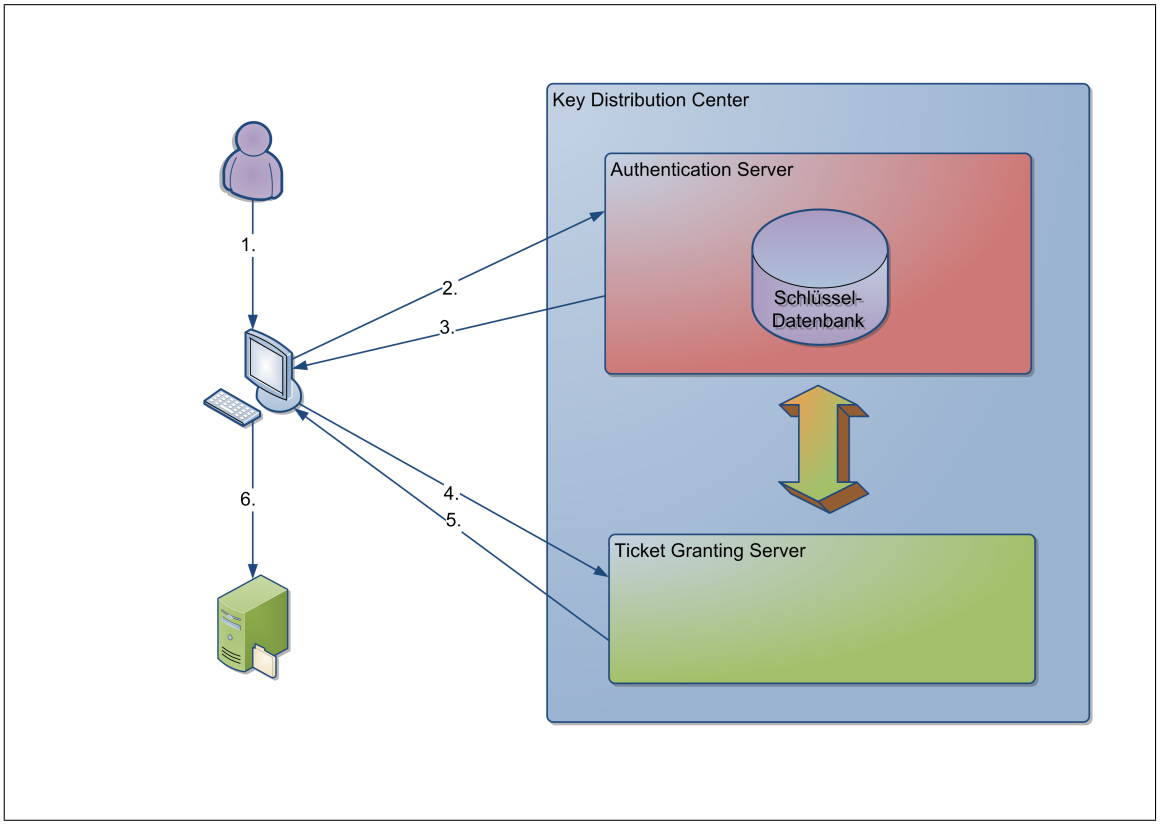

Abbildung 2.5: Authentifikation bei Kerberos

zunächst ein TGT[23] beim KDC anfordern, das ihn berechtigt, Tickets für einzelne Dienste (wie z.B. der Zugriff auf einen Dateiserver) beim TGS anzufordern (**1.+2.**). Der AS kennt den Schlüssel des Benutzers und stellt ihm das TGT aus. Das Ticket enthält einen Schlüssel, der zur Verschlüsselung der Kommunikation zwischen dem Benutzer und dem TGS verwendet wird. Das Ticket wird mit dem Schlüssel des Benutzers, der i.d.R. aus dessen Passwort generiert wird, verschlüsselt und an den Benutzer gesendet (**3.**). Durch die Verschlüsselung wird sichergestellt, dass nur der Benutzer in der Lage ist das Ticket zu verwenden. Damit ist der Benutzer gegenüber dem System authentifiziert.

Möchte der Benutzer z.B. die Dienste eines Dateiservers nutzen, muss er beim TGS ein Ticket dafür anfordern (**4.**). Der TGS erkennt die Gültigkeit der Anfrage daran, dass die Anfrage mit dem Schlüssel aus dem TGT verschlüsselt wurde. Der TGS stellt ein Ticket für den angeforderten Dienst aus. Das Ticket enthält einen Schlüssel, der zum Verschlüsseln der Kommunikation zwischen dem Benutzer und dem Dienst genutzt wird. Dieser Schlüssel wird doppelt an den Benutzer gesendet. Einmal mit dem Schlüssel aus dem TGT verschlüsselt, einmal mit dem Schlüssel des Dienstes verschlüsselt (**5.**).

[23] Ticket Granting Ticket

Mit diesem Ticket kann der Benutzer den Dienst in Anspruch nehmen. Die Anfragen an den Dienst werden mit dem Schlüssel aus dem Ticket verschlüsselt. Zusätzlich wird der Schlüssel, der mit dem Schlüssel des Dienstes verschlüsselt ist, mitgesendet (**6.**). Der Dienst erkennt die Gültigkeit der Anfrage daran, dass er in der Lage ist die Nachricht mit dem für ihn verschlüsselten Schlüssel zu entschlüsseln (Vgl. Eckert, 2006).

Mit der Version 5 von Kerberos wurde eine Möglichkeit eingeführt, Tickets über mehrere Vertrauensbereiche weiterzureichen. Dazu muss eine Vertrauensbeziehung zwischen den KDCs einzelner Bereiche bestehen, die dann in einer Hierarchie angeordnet werden können. So angeordnete KDCs können dann TGTs für die TGTs anderer Vertrauensbereiche ausstellen, mit denen sie direkt verbunden sind. Um Tickets für weiter entfernte Vertrauensbereiche auszustellen, müssen die TGTs entsprechend durchgereicht werden.

3 Anforderungsanalyse und Marktübersicht

In diesem Kapitel wird ein realistisches Anwendungsszenario entworfen. Aus diesem Szenario werden die Anforderungen für ein Sicherheitsframework abgeleitet. Es wird ein Überblick über vorhandene Frameworks gegeben und diese werden nach den ermittelten Anforderungen bewertet.

3.1 Anforderungen an das Szenario

Das entworfene Anwendungsszenario soll typische Sicherheitsprobleme enthalten, die in einer global verteilten Anwendung auftreten. Unter einer global verteilten Anwendung wird in dieser Arbeit eine Anwendung verstanden, die auf einer SOA basiert und deren Dienste als Webservices implementiert sind.

Wenn ein Webservice im Internet zur Verfügung gestellt wird, kann er von unterschiedlichen Subjekten genutzt werden, wenn der Anbieter des Dienstes dies zulässt. Bei Wiehler (2004) wird die Nutzung eines Webservice in vier Phasen eingeteilt:

- Phase 1: Nutzung des Dienstes nur organisationsintern
- Phase 2: Nutzung des Dienstes durch ausgewählte Partnerorganisationen
- Phase 3: Nutzung des Dienstes durch Partnerorganisationen im Rahmen eines Netzwerks
- Phase 4: Öffentliche Nutzung des Dienstes

Die verschiedenen Phasen haben erhebliche Auswirkung auf die Authentifikation und die Autorisierung der Subjekte. Während es in der ersten Phase problemlos möglich ist, jedes Subjekt in der eigenen Benutzerverwaltung zu speichern, wird dies bereits in der zweiten Phase schwierig, da sichergestellt werden muss, dass jeder Zu- bzw. Abgang oder veränderte Rechte bei den Mitgliedern einer Partnerorganisation bekannt gemacht werden. Spätestens in der dritten Phase, in der auch die partizipierenden Partnerorganisationen wechseln

können, scheint dieses Vorgehen nicht mehr geeignet zu sein. In der vierten Phase muss zusätzlich überlegt werden, wie unbekannte Subjekte authentifiziert und autorisiert werden können. Für die jeweilige Einordnung eines Subjektes ergeben sich die folgenden Konstellationen, die in dem Anwendungsszenario berücksichtigt werden müssen.

- Das Subjekt ist bekannt
- Das Subjekt ist einer Partnerorganisation bekannt
- Das Subjekt ist unbekannt

Darüber hinaus besteht die Möglichkeit, dass der Dienst von einem Subjekt genutzt wird, das im Auftrag eines anderen handelt. In diesem Fall muss gegebenenfalls neben der Authentizität und der Autorisierung des Subjektes auch geprüft werden, ob das Subjekt bevollmächtigt ist, den Dienst im Auftrag des anderen Subjekts zu nutzen.

3.2 Anwendungsszenario

Als Fallbeispiel für das Anwendungsszenario wird eine Autovermietung verwendet. Diese Autovermietung bietet einen Dienst an, über den man Autos mieten kann. Damit ein Anwender berechtigt werden kann, ein Auto zu mieten, muss dieser eine der folgenden Voraussetzungen erfüllen:

- Er wird als Stammkunde in der Kundendatenbank geführt.
- Er ist Mitarbeiter eines Partnerunternehmens, das einen Generalvertrag mit der Autovermietung hat.
- Er besitzt eine gültige Kreditkarte.

Die Autovermietung ist Mitglied bei einem Preisvergleichportal, über das der Dienst ebenfalls genutzt werden kann. Das Portal bietet seinerseits ebenfalls einen Dienst dafür an. In der Abbildung 3.1 wird ein Überblick über die verschiedene Aufrufarten des Dienstes gegeben:

- Aufruf des Dienstes durch einen Stammkunden
- Aufruf des Dienstes durch einen unbekannten Benutzer
- Aufruf des Dienstes durch einen Mitarbeiter eines Partnerunternehmens
- Aufruf des Dienstes über das Preisvergleichportal

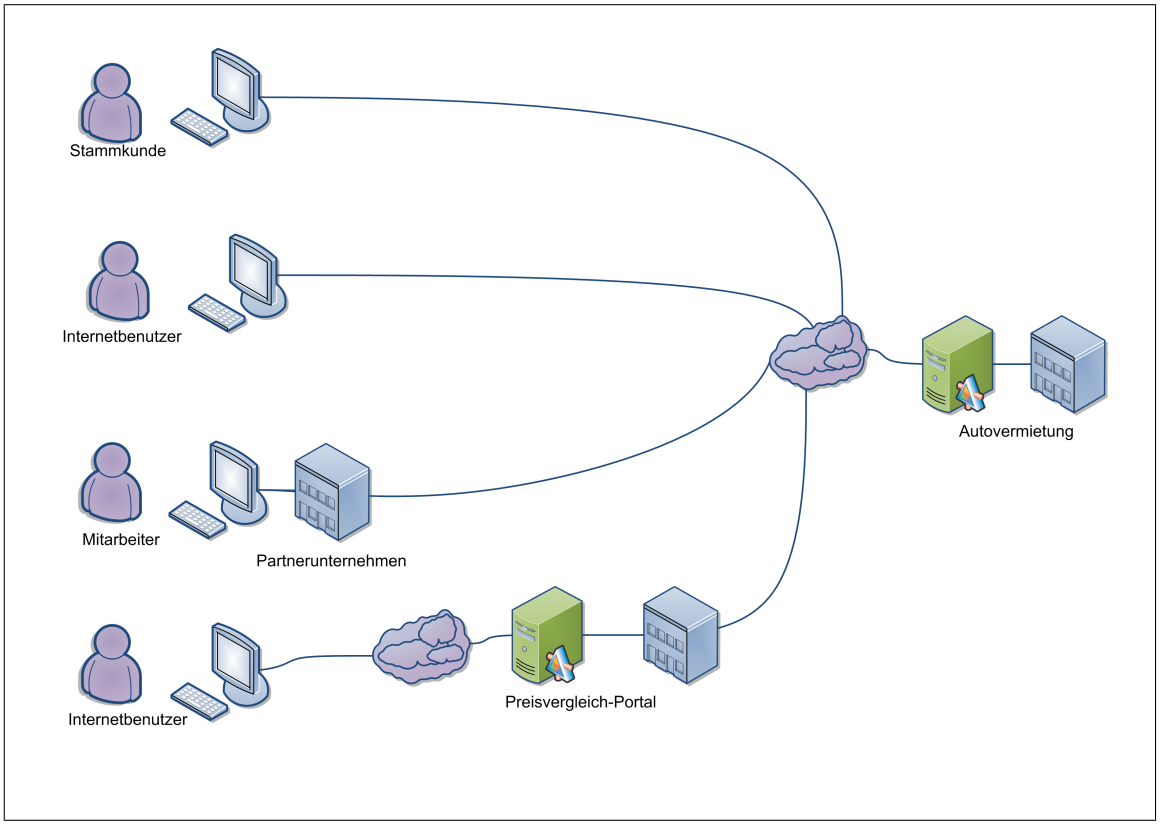

Abbildung 3.1: Übersicht des Anwendungsszenarios

3.2.1 Anwendungsfall: Stammkunde

Der Benutzer ruft den Dienst der Autovermietung direkt auf. Als Stammkunde ist der Benutzer in der Kundendatenbank gespeichert. In der Kundendatenbank sind auch seine Kontodaten gespeichert. Der Benutzer muss angeben, welches Auto er in welchem Zeitraum mieten möchte. Der Dienst sendet eine Bestätigung oder eine Fehlermeldung.

3.2.2 Anwendungsfall: Unbekannter Benutzer

Der Benutzer ruft den Dienst der Autovermietung direkt auf. Der Benutzer ist nicht in der Kundendatenbank gespeichert und muss daher seine Kredikarteninformation angeben. Der Benutzer muss angeben, welches Auto er in welchem Zeitraum mieten möchte. Der Dienst sendet eine Bestätigung oder eine Fehlermeldung.

3.2.3 Anwendungsfall: Mitarbeiter eines Partnerunternehmens

Der Mitarbeiter eines Partnerunternehmens ruft den Dienst der Autovermietung über eine Anwendung im Intranet seines Unternehmens auf. Das gemietete Fahrzeug wird von dem Partnerunternehmen bezahlt. Der Benutzer muss angeben, welches Auto er in welchem Zeitraum mieten möchte. Der Dienst sendet eine Bestätigung oder eine Fehlermeldung.

3.2.4 Anwendungsfall: Vermittler-Dienst

Der Benutzer sucht über einen Dienst des Preisvergleichportals den günstigsten Anbieter für ein bestimmtes Fahrzeug in einem bestimmten Zeitraum. Aus dem Ergebnis der Suche wählt der Benutzer einen Anbieter aus und übergibt dem Portal den Auftrag, das Fahrzeug bei diesem Anbieter zu mieten. Das Preisvergleichportal leitet den Aufruf an den Dienst der Autovermietung weiter. Der Benutzer bezahlt nur für den Dienst der Autovermietung und muss dafür seine Kredikarteninformation angeben. Das Preisvergleichportal erhält für die Vermittlung des Auftrags eine Provision von der Autovermietung.

3.3 Funktionale Anforderungen an das Framework

Die IT-Grundschutz-Kataloge (BSI, 2006) teilen die IT-Sicherheitsaspekte in fünf Schichten ein. In Abbildung 3.2 sind die einzelnen Schichten und deren Einteilung in die hier verwendeten Begriffe dargestellt. Die Sicherheitsmaßnahmen lassen sich in allgemeine Maßnahmen (Schichten 1+2) und in Maßnahmen, die direkt die IT-Systeme betreffen einteilen (Schichten 3-5).

Die Sicherheitsmaßnahmen in IT-Systemen lassen sich in den Bereich der Perimetersicherheit (Schichten 3+4) und den Bereich der Anwendungssicherheit (Schicht 5) einteilen. Unter Perimetersicherheit wird hier die Sicherheit der IT-Infrastruktur, auf der die Anwendung läuft, verstanden, d.h. die Absicherung des Netzwerks und der Server gegen Angriffe, die sich nicht direkt gegen die Anwendung richten (z.B. Denial-of-Service-Angriffe[1]). Die Maßnahmen zur Gewährleistung der Perimetersicherheit sind ein notwendiger Bestandteil einer Sicherheitsarchitektur für eine global verteilte Anwendung. Mögliche Maßnahmen sind z.B. der Einsatz von Firewalls und Virenscannern.

Aufgabe des zu entwerfenden Frameworks ist nicht die Gewährleistung der Perimetersicherheit, sondern die Gewährleistung der Anwendungssicherheit. Unter Anwendungssicherheit

[1] Bei einem Denial-of-Service-Angriff wird ein Server innerhalb kurzer Zeit mit so vielen Anfragen überhäuft, dass er nicht mehr reagieren kann.

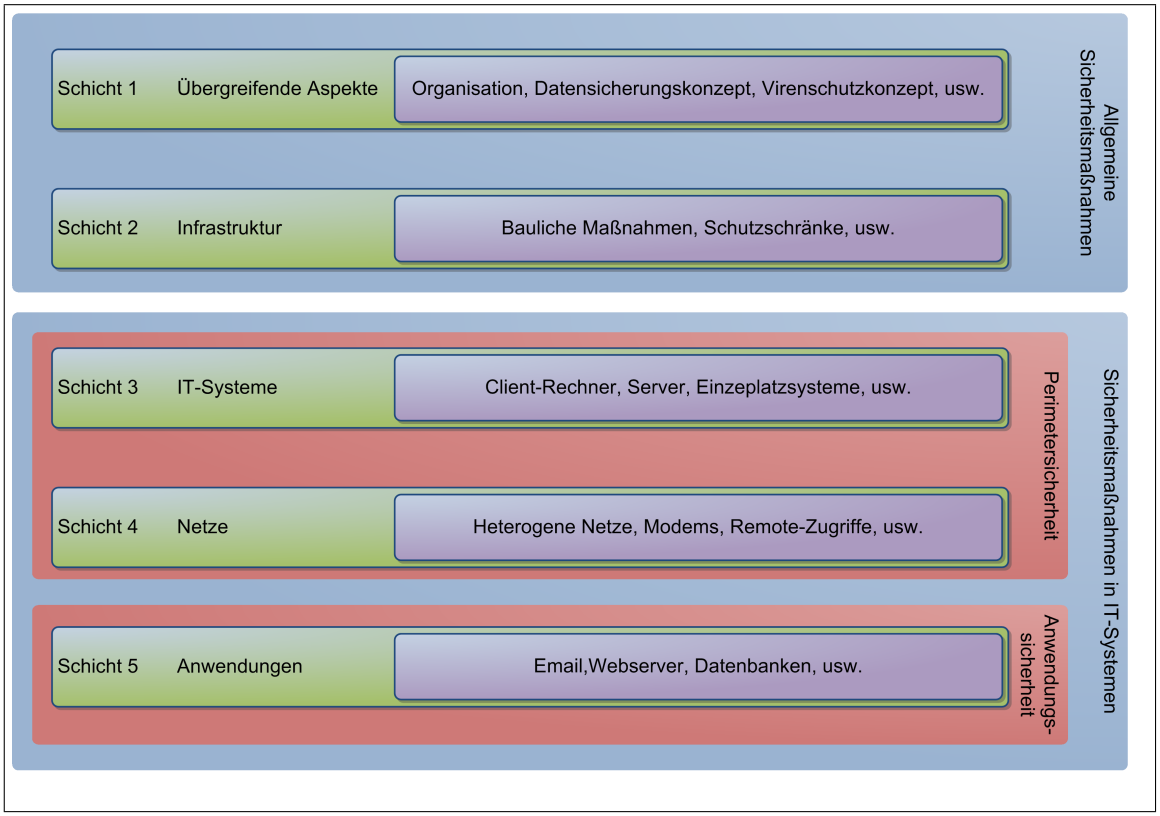

Abbildung 3.2: Schichten der IT-Grundschutz-Kataloge

werden hier die Maßnahmen verstanden, die auf Anwendungsebene getroffen werden, um die klassischen Sicherheitsziele wie Vertraulichkeit und Integrität zu erreichen (Vgl. Eckert, 2006).

3.3.1 Schutzbedarf der Anwendung

Bei der Ermittlung des Schutzbedarfs muss für verschiedene Schadenszenarien ermittelt werden, welche Auswirkung eine Verletzung der Grundziele Vertraulichkeit und Integrität auf das jeweilige Szenario hat. Dabei sind folgende Szenarien zu berücksichtigen (Vgl. Eckert, 2006):

- Verstoß gegen Gesetze/Vorschriften/Verträge
- Beeinträchtigung des informationellen Selbstbestimmungsrechts
- Beeinträchtigung der persönlichen Unversehrtheit
- Beeinträchtigung der Aufgabenerfüllung

- Negative Außenwirkung
- Finanzielle Auswirkungen

Für diese Szenarien muss ermittelt werden, wie hoch die entsprechenden Auswirkungen sind und wie hoch der daraus folgende Schutzbedarf ist. Der Schutzbedarf der gesamten Anwendung wird nach dem Maximumprinzip aus den Einzelwerten für die Szenarien gebildet.

Da das zu entwerfende Framework für beliebige Anwendungen einsetzbar ist, muss es in der Lage sein, einen maximalen Schutzbedarf zu erfüllen.

3.3.2 Sicherheitsmaßnahmen in den Anwendungsfällen

Um einen hohen Schutzbedarf zu erfüllen, müssen in den ausgetauschten Webservicenachrichten Vertraulichkeit, Integrität und Verbindlichkeit garantiert sein. Um diese Ziele zu garantieren, müssen alle Subjekte authentifiziert und autorisiert werden. Um die Vertraulichkeit und die Integrität der Daten in den Webservicenachrichten zu garantieren, müssen sämtliche Nutzdaten verschlüsselt werden.

Stammkunde Die Authentifizierung und Autorisierung von Stammkunden kann anhand einer eigenen Benutzerverwaltung geschehen. Der Benutzer kann sich mittels geheimem Wissen (z.B. Passwort), Besitz (z.B. Smart-Card) oder durch biometrische Merkmale der Autovermietung gegenüber ausweisen. Anhand der Benutzerverwaltung kann geprüft werden, welche Fahrzeuge der Kunde mieten darf.

Die Daten, die in den Webservice-Nachrichten zwischen dem Kunden und der Autovermietung ausgetauscht werden, müssen zum Schutz gegen Ausspionieren und Manipulation der Daten verschlüsselt werden.

Unbekannter Benutzer Ein unbekannter Benutzer muss sich über ein Zertifikat authentifizieren, dass von einer vertrauenswürdigen Zertifizierungsstelle ausgestellt wurde. Damit er autorisiert werden kann ein Fahrzeug zu mieten, muss er zusätzlich eine gültige Kreditkarte angeben. Die Daten, die in den Webservice-Nachrichten zwischen dem Kunden und der Autovermietung ausgetauscht werden, müssen wie beim Stammkunden verschlüsselt werden.

Mitarbeiter eines Partnerunternehmens Benutzer einer Partnerorganisation werden nicht von der Autovermietung authentifiziert, sondern von der Partnerorganisation selbst. Dazu muss der Benutzer der Autovermietung einen Nachweis erbringen, der belegt, dass er Mitglied der Partnerorganisation ist. Anhand dieses Nachweises kann die Autovermietung die Autorisierungsinformationen bei der Partnerorganisation erfragen. Die Daten, die in den Webservice-Nachrichten zwischen dem Kunden und der Autovermietung ausgetauscht werden, müssen wie bei den anderen Anwendungsfällen verschlüsselt werden.

Preisvergleich-Portal Das Preisvergleichportal muss sich der Autovermietung gegenüber mit einem Zertifikat authentifizieren. Außerdem muss es nachweisen, dass es bevollmächtigt ist, im Auftrag eines anderen Benutzers zu handeln. Dieser Benutzer muss sich wie ein unbekannter Benutzer, der den Dienst direkt nutzt, authentifizieren.

Die Daten, die in den Webservice-Nachrichten zwischen dem Kunden und der Autovermietung ausgetauscht werden, müssen wie bei den anderen Anwendungsfällen verschlüsselt werden. Während die allgemeinen Angaben über den Benutzer und das zu mietende Fahrzeug sowohl für das Preisvergleichportal, als auch für die Autovermietung lesbar sein müssen, dürfen die Kreditkarteninformationen nur von der Autovermietung lesbar sein. Für diese Daten muss deshalb ein anderer Schlüssel verwendet werden.

3.3.3 Funktionalität des Frameworks

Aus den Sicherheitsmaßnahmen, die in den Anwendungsfällen benötigt werden, lassen sich die funktionalen Anforderungen an das Framework ableiten.

Authentifikation Das Framework muss verschiedene Möglichkeiten der Authentifikation unterstützen. Während Stammkunden sich mittels Passwort oder biometrischer Merkmale authentifizieren können, können unbekannte Benutzer nur über Aussagen von vertrauenswürdigen Dritten authentifiziert werden (z.B. Zertifikate). Darüber hinaus muss die Nutzung von Smart-Cards möglich sein.

Verschlüsselung Das Framework muss die übertragenen Daten verschlüsseln. Da asymmetrische Verschlüsselung für große Datenmengen zu rechenaufwändig ist, müssen die Daten symmetrisch verschlüsselt werden. Dazu ist es notwendig, dass der symmetrische Schlüssel sicher zwischen den Kommunikationspartnern ausgetauscht werden kann.

Sicherheitskontexte Wenn der Webservice nicht direkt aufgerufen wird sondern über einen Vermittler-Dienst (z.B. das Preisvergleichportal), muss es möglich sein, bestimmte Daten für den Vermittler unzugänglich zu verschlüsseln. Dazu müssen mehrere Sicherheitskontexte in einer Nachricht existieren, die für unterschiedliche Teilnehmer sichtbar sind. Da der Vermittler gegenüber dem Webservice im Auftrag des ursprünglichen Benutzers handelt, muss er dazu bevollmächtigt werden können.

Föderieren von Identitäten Im Fall des Mitarbeiters einer Partnerorganisation kann die Autovermietung nicht selbst bestimmen, ob der Benutzer berechtigt ist, den Dienst zu nutzen oder nicht. Diese Information muss von der Partnerorganisation erfragt werden. Das Framework muss daher das Erfragen und Weitergeben von Benutzerinformationen unterstützen.

Dynamische Konfiguration Damit die Anwendung sich nicht um die Sicherheitsmaßnahmen kümmern muss, neue Dienste leicht integriert werden können und neue Sicherheitsanforderungen keine Auswirkung auf bestehende Anwendungen haben, muss das Framework dynamisch konfigurierbar sein. Die Konfiguration muss über einen universellen Mechanismus erfolgen, damit Dienste interoperabel verwendet werden können.

3.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen für eine SOA sind in hohem Maße von der damit realisierten Anwendung abhängig (Vgl. Bieberstein u. a., 2006). Ein Dienst, der auf der ganzen Welt genutzt wird, stellt wesentlich höhere Anforderungen an z.B. die Verfügbarkeit des Systems als ein Dienst, der ausschließlich im Intranet eines Unternehmens genutzt wird. Von hoher Bedeutung für die Anforderungen ist unter anderem auch wie geschäftskritisch der Dienst ist.

Ein Framework, das für beliebige Anwendungen einsetzbar ist, muss daher sehr hohen Anforderungen genügen. Es gelten daher die allgemeinen Qualitätsmerkmale für Softwareprodukte (Vgl. Kahlbrandt, 2001):

- Funktionserfüllung (hier: Sicherheit)
- Effizienz
- Zuverlässigkeit
- Benutzbarkeit
- Erweiterbarkeit

- Wartbarkeit
- Übertragbarkeit
- Wiederverwendbarkeit
- Sicherheit

Von besonderer Bedeutung sind für ein Framework die Punkte Erweiterbarkeit, Wiederverwendbarkeit und Sicherheit. Die Bedeutung der Wiederverwendbarkeit ergibt sich aus dem Zweck eines Frameworks, das in beliebig vielen Anwendung eingesetzt werden soll. Bei der Wiederverwendbarkeit dieses Frameworks muss insbesondere berücksichtigt werden, dass das Framework sowohl auf einem Anwendungsserver, als auch in einer Client-Anwendung einsetzbar sein muss. Da das Framework in Anwendungen eingesetzt werden soll, die global über mehrere Organisationen verteilt sein können, spielt die Interoperabilität mit anderen Frameworks für die Wiederverwendbarkeit des Frameworks ein große Rolle. Dies kann durch den Einsatz offener Standards erreicht werden.

Da unterschiedliche Anwendungen zusätzliche Anforderungen stellen können, muss es möglich sein, das Framework mit vertretbarem Aufwand zu erweitern, so dass diese Anforderungen abgedeckt werden.

Um die Sicherheit in Anwendungen sicherzustellen, muss das Framework selbst hohen Sicherheitsanforderungen genügen, da durch Kompromittieren des Frameworks die Sicherheit der ganzen Anwendung verloren geht. Im schlimmsten Fall könnte ein Angreifer in den Besitz von privaten Schlüsseln kommen, was sich auch auf andere Anwendungen auswirken würde, die von denselben Anwendern genutzt werden.

3.5 Marktübersicht

In diesem Kapitel soll ein Überblick über vorhandene Frameworks gegeben werden, die Sicherheit in Webservice-basierten Anwendungen durch Verwendung von offenen Standards ermöglichen. Die vorgestellten Frameworks sollen anhand der Anforderungen, die in diesem Kapitel formuliert wurden, bezüglich ihrer Eignung für den Einsatz in einer global verteilten Anwendung bewertet werden.

3.5.1 Vorhandene Frameworks

Die wichtigen Softwarehersteller und Open-Source-Projekte bieten bereits Implementierungen für WS-Security an, von denen sich die meisten auf SOAP Message Security beschränken. Obwohl die Implementierungen teilweise in der Lage sind, Policies zu verarbeiten und

teilweise auch die darin vorgegebenen Sicherheitsmaßnahmen umzusetzen, werden die Policies von den Implementierungen hauptsächlich dazu verwendet, auf der Serverseite zu überprüfen, ob die geforderten Sicherheitsmaßnahmen eingehalten sind. Um einen Eindruck des aktuellen Standes zu vermitteln, werden hier ein paar ausgewählte Implementierungen von WS-Security vorgestellt.

Microsoft .net Web Service Enhancements Microsoft bietet mit WSE[2] ein Framework an, mit dem eigene Handler in die .net-Webservice-Middleware eingebunden werden können. WSE bietet auch eigene Handler für WS-Security an, in denen die folgenden Spezifikationen implementiert sind:

- SOAP Message Security (WS-Security)
- WS-Policy, WS-SecurityPolicy
- WS-Trust
- WS-SecureConversation
- Username-, X509-, und Kerberos-Tokens

WSE konzentriert sich auf einige häufige Szenarien, die mit Policies und den WSE-Handlern leicht umgesetzt werden können, verzichtet dabei aber nicht auf proprietäre Erweiterungen. Wenn eigene Protokolle umgesetzt werden sollen, muss in den Anwendungscode selbst eingegriffen werden. Die Interoperabilität mit Java-Frameworks kann erreicht werden, wenn diese sich an die Anforderungen von WSE anpassen (Vgl. Microsoft, 2007).

Apache Rampart Apache bietet mit Rampart ein Modul für die Webservice-Middleware Axis 2(siehe Apache, 2007a) an, das auf WSS4J[3] (siehe Apache, 2007c) basiert. WSS4J ist ein Java-API für WS-Security, das die folgenden Spezifikationen implementiert:

- SOAP Message Security (WS-Security)
- WS-Policy, WS-SecurityPolicy
- Username- und X509-Tokens

Einfache Standard-Protokolle (wie z.B. Public-Key-Verschlüsselung und Signierung) können durch Policies definiert werden, in denen aber Rampart-eigene Policies zur Konfiguration benötigt werden. Diese Policies können zu Interoperabilitätsproblemen führen, wenn es darum geht die Policies von Server und Client zu vergleichen. Um eigene Protokolle umsetzen zu können, muss in den Anwendungscode eingegriffen werden.

[2] Web Service Enhancements
[3] Web Service Security for Java

JBoss Web Services Das Webservice-Framework von JBoss unterstützt die folgenden Spezifikationen:

- SOAP Message Security (WS-Security)
- Username- und X509-Tokens

Es können keine Sicherheitsprotokolle über Policies definiert werden. In einer Konfigurationsdatei für die Anwendung kann angegeben werden, wie ein- und ausgehende Webservice-Nachrichten verschlüsselt und signiert werden sollen. Es kann dabei nicht zwischen Diensten unterschieden werden.

IBM WebSphere Application Server Der WAS[4] von IBM (Vgl. IBM, 2007) unterstützt die folgenden Spezifikationen:

- SOAP Message Security (WS-Security)
- Username- und X509-Tokens

Für Webservices können ähnlich wie bei WS-Policy Anforderungen formuliert werden, die über einen Wizard in der Entwicklungsumgebung generiert werden können. Für die Definition der Anforderungen wird jedoch nicht WS-Policy sondern eine proprietäre Sprache verwendet.

3.5.2 Bewertung der Frameworks

In Tabelle 3.1 ist dargestellt, in welchem Umfang die betrachteten Frameworks die in diesem Kapitel formulierten Anforderungen abdecken. Die Standardfunktionen Authentifikation, Verschlüsselung und Signierung werden von allen Frameworks angeboten, wobei keine der Spezifikationen alle in WS-Security definierten Security-Tokens unterstützt. Das Föderieren von Entitäten wird nur ansatzweise von Microsofts WSE durch Implementierung von WS-Trust unterstützt.

Einer der wichtigsten Punkte für den Einsatz in einer global verteilten Anwendung, die Konfigurierbarkeit der Sicherheitsfunktionen, wird von keinem Framework in zufriedenstellendem Umfang unterstützt. Die einzigen Frameworks, die WS-Policy unterstützen (Microsoft WSE und Apache Rampart), kommen nicht ohne proprietäre Erweiterungen aus.

Da Frameworks prinzipiell wiederverwendbar sind, wurden die Frameworks danach bewertet, wie universell sie einsetzbar sind oder ob sie nur in bestimmten Szenarien eingesetzt werden können.

[4] WebSphere Application Server

Framework	Authentifikation	Verschlüsselung	Signierung	Föderierte Entitäten	Konfiguration der Sicherheit	Konfiguration universell	Erweiterbarkeit	Wiederverwendbarkeit	Interoperabilität
Microsoft .net WSE	●	●	●	◐	●	◐[1]	●	◐[2]	◐[3]
Apache Axis 2 mit Rampart	●	●	●	○	◐[4]	◐[1]	●	●[5]	●
JBoss Web Services	●	●	●	○	◐[6]	○	●	?	●
IBM WebSphere Application Server	●	●	●	○	●	○	?	◐[7]	●

[1] WS-Policy wird unterstützt, aber es werden proprietäre Erweiterungen verwendet
[2] Nur in .net-Anwendungen einsetzbar
[3] Nur ausgewählte Funktionalitäten der Spezifikation umgesetzt, proprietäre Erweiterungen
[4] Nur einfache Protokolle über Policies konfigurierbar
[5] Rampart kann auch unabhängig von Axis 2 verwendet werden
[6] Konfiguration gilt für alle Webservices einer Anwendung
[7] Einsatz nur im IBM WAS möglich
●= erfüllt, ○= nicht erfüllt, ◐= teilweise erfüllt, ? = unbekannt

Tabelle 3.1: Bewertung vorhandener Sicherheitsframeworks

Wie aus der Tabelle 3.1 deutlich hervorgeht, werden die Anforderungen aus diesem Kapitel von keinem der Frameworks zufriedenstellend erfüllt. In Kapitel 4 wird deshalb anhand des vorgestellten Anwendungsszenarios ein Framework entworfen, dass die gestellten Anforderungen erfüllt.

4 Konzeption eines Sicherheitsframeworks

Für die Absicherung von Webservice-basierten Anwendungen gibt es zusätzlich zu den im Grundlagenkapitel vorgestellten WS-Security-Spezifikationen die SAML[1]-Spezifikation und das Liberty Alliance Project, das sich die Erstellung eines offenen Standards für FIM[2] zur Aufgabe gemacht hat. SAML und Liberty Alliance konkurrieren mit den WS-Security-Spezifikationen. Da sich die Spezifikationen jeweils mit derselben Problemstellung beschäftigen, sind die daraus folgenden Schwachstellen in allen Spezifikationen vorhanden (Vgl. Hommel und Reiser, 2005):

- Beschränkung auf Webservices
- Keine Verfahren zum Cachen / Verteilen von Änderungen von verteilten Benutzerinformationen
- Sicherheitsaspekte beschränken sich überwiegend auf die Kommunikation
- Kein allgemeines Datenmodell für Benutzer- und Berechtigungsinformationen

Da die WS-Security-Spezifikationen zusammen mit den weiteren WS-*-Spezifikationen ein breiteres Spektrum abdecken und somit einen einheitlichen Quality-of-Service-Ansatz für Webservices bieten, werden in dieser Arbeit die WS-Security-Spezifikation verwendet, um die Anwendung sicher zu machen. Die Vor- und Nachteile gegenüber SAML und dem Liberty Alliance Project sollen in dieser Arbeit nicht betrachtet werden.

4.1 Einsatz der Spezifikationen

Der zentrale Ansatz der WS-Security-Spezifikationen, um Webservices sicher zu machen, ist die Anreicherung der Webservicenachrichten um „Security Tokens“. Diese Tokens können bestimmte Aussagen über ein Objekt oder ein Subjekt machen und dazu verwendet werden,

[1] Security Assertion Markup Language
[2] Federated Identity Management

um Schlüssel auszutauschen. Jeder Webservice, der Security-Tokens erwartet, benötigt daher einen STS[3], dem er vertraut. Nutzer des Webservice können sich die geforderten Tokens von dem STS erstellen lassen oder der Webservice fordert die Tokens selbst an und übergibt dabei die benötigten Daten des Nutzers.

Ein STS kann z.B. die Aufgabe des KDC[4] in einem Kerberos-System übernehmen. Das Konzept des STS ist allerdings allgemeiner gehalten, so dass ein STS beliebige Tokens ausstellen, validieren oder erneuern kann.

4.1.1 Anwendungsfall bekannter Benutzer

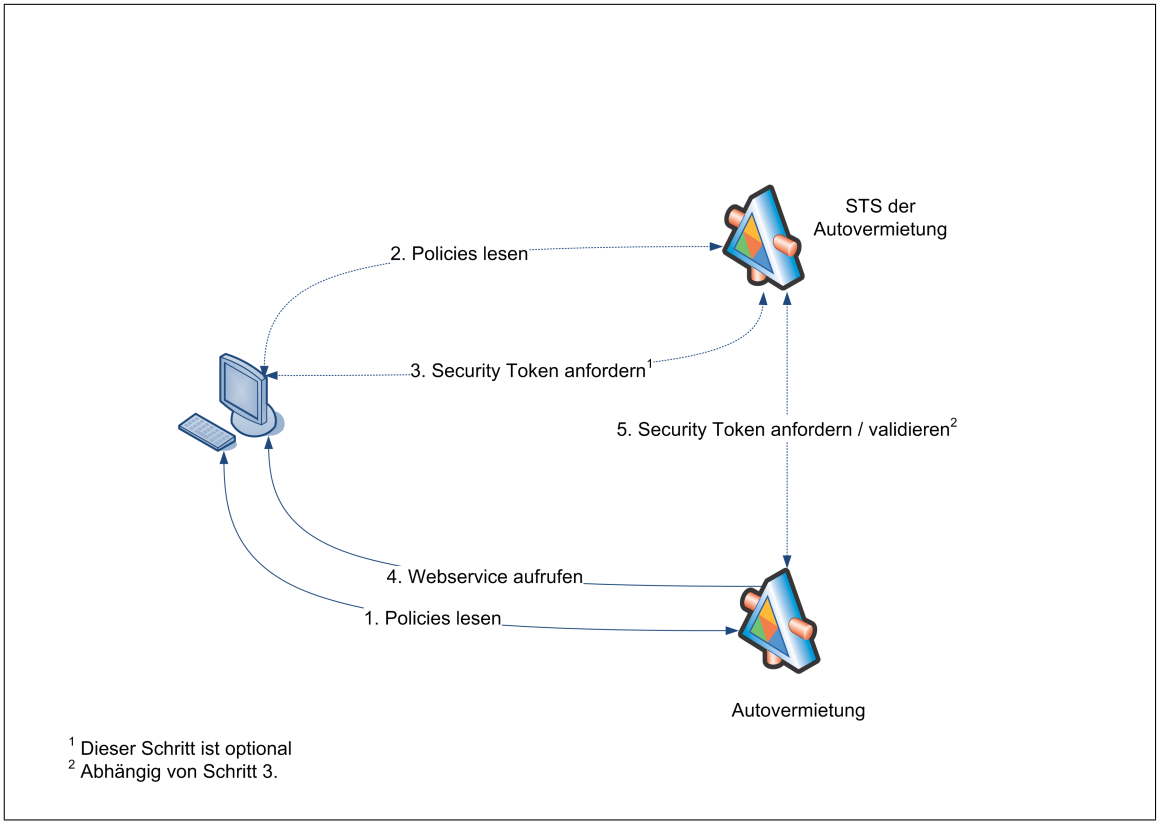

Abbildung 4.1: Grobkonzept: Bekannter Benutzer

Abbildung 4.1 zeigt den prinzipiellen Ablauf des Webservice-Aufrufs durch einen bekannten Benutzer. Bevor der eigentliche Webservice aufgerufen wird, werden zunächst die Metadaten des Webservices abgefragt (**1.**). In den Metadaten sind die Policies enthalten. In den

[3] Security Token Service
[4] Key Distribution Center

Policies ist definiert, welche Angaben benötigt werden, um den Webservice aufzurufen und wie die Nutzdaten gesendet werden sollen, d.h. welche Teile der SOAP-Nachricht verschlüsselt oder signiert werden sollen.

Ein bekannter Benutzer kann sich mit seiner Kundennummer und seinem Passwort gegenüber der Autovermietung authentifizieren. Die Authentifikation und die darauffolgende Autorisierung des Benutzers wird jedoch nicht von der Anwendung durchgeführt, sondern von dem STS der Autovermietung, der einen Security-Token erstellt, anhand dessen der Autovermietungsdienst feststellen kann, ob die Webservice-Anfrage berechtigt ist oder nicht.

Dieser Security-Token kann auf zwei unterschiedlichen Wegen erstellt werden. Zum einen können die Policies des Autovermietungsdienstes verlangen, dass der Aufrufer des Dienstes diesen Token bereits mitsendet. In diesem Fall muss der Aufrufer selbst den Security-Token-Service der Autovermietung aufrufen, dessen Policies seine Kundennummer und sein Passwort erwarten (**2. + 3.**). Nach Überprüfung dieser Daten wird der Token ausgestellt und an den Aufrufer gesendet, der diesen dann mit seiner eigentlichen Anfrage an den Autovermietungsdienst sendet (**4.**). Bevor der Autovermietungsdienst die Anfrage dann bearbeitet, muss er den mitgesendeten Token validieren. Ist der Autovermietungsdienst nicht selbst in der Lage den Token zu validieren, kann dies vom STS übernommen werden (**5.**).

Bei dem zweiten Weg verlangen die Policies des Autovermietungsdienstes die Kundennummer und das Passwort des Benutzers. Dieser sendet diese Daten zusammen mit seiner Anforderung an den Dienst (**4.**). In diesem Fall fordert der Autovermietungsdienst selbst den Security Token beim STS an (**5.**), bevor er die eigentliche Anfrage verarbeitet.

4.1.2 Unbekannter Benutzer

Der Aufruf durch einen unbekannten Benutzer läuft prinzipiell genauso ab, wie der Aufruf durch einen bekannten Benutzer. Der Unterschied besteht dabei in den Benutzerinformationen, die benötigt werden. Während sich ein bekannter Benutzer mit seiner Kundennummer und seinem Passwort authentifizieren kann, benötigt ein unbekannter Benutzer ein Zertifikat, das von einer vertrauenswürdigen CA ausgestellt wurde.

4.1.3 Benutzer eines Partnerunternehmens

In Abbildung 4.2 zeigt den prinzipiellen Ablauf des Webservice-Aufrufes durch einen Mitarbeiter eines Partnerunternehmens. Wie bei dem bekannten Benutzer müssen zunächst die Policies des Webservice und danach die Policies des STS der Autovermietung gelesen werden (**1. + 2.**). Der STS der Autovermietung erwartet in diesem Fall nicht Benutzernamen und

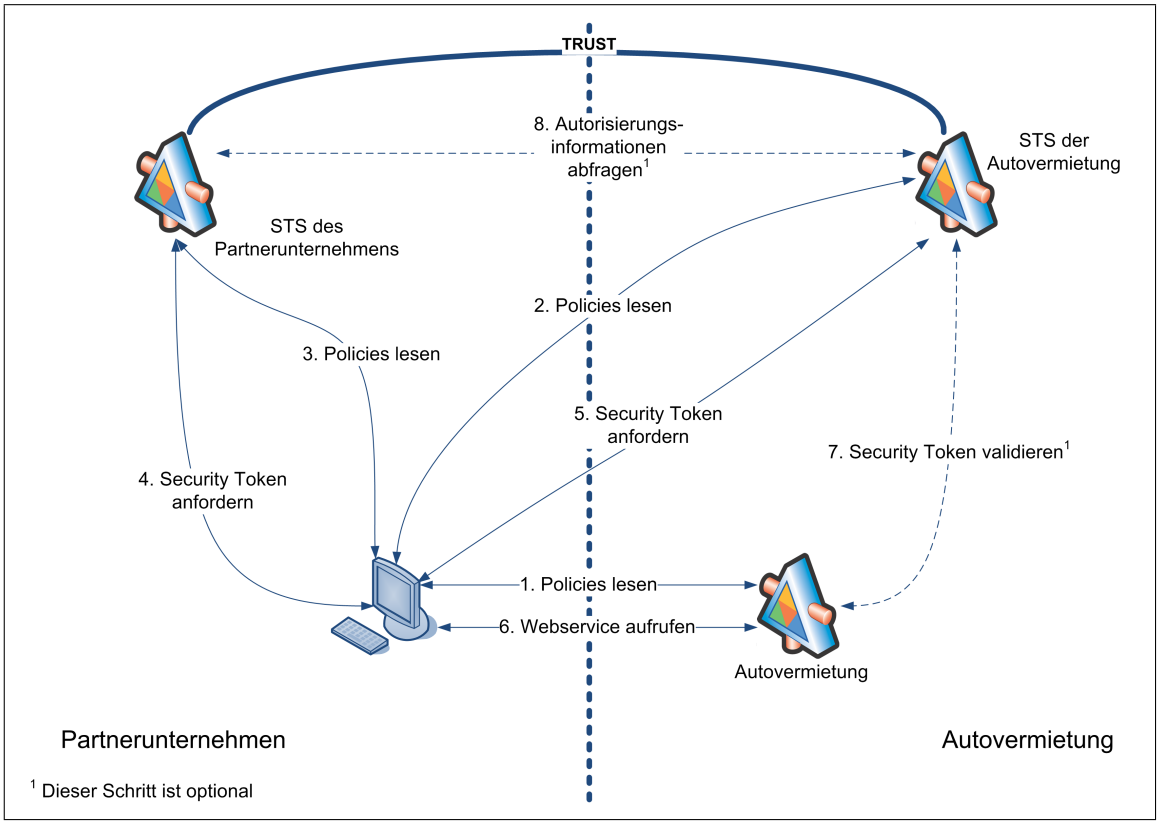

Abbildung 4.2: Grobkonzept: Föderierte Entitäten

Passwort, sondern einen Security Token des Partnerunternehmens (**4.**). Erst mit diesem Token, kann der Benutzer den eigentlichen Security Token anfordern (**5.**), der ihn zur Nutzung des Dienstes berechtigt.

Mit diesem Token kann der Aufruf des eigentlichen Dienstes erfolgen (**6.**). Falls der Dienst den Token nicht selbst validieren kann, kann er den Token von seinem eigenen STS überprüfen lassen (**7.**). Dabei müssen eventuell Autorisierungsinformationen vom STS des Partnerunternehmens erfragt werden (**8.**).

4.1.4 Aufruf über das Preisvergleichportal

Abbildung 4.3 zeigt den prinzipiellen Ablauf des Webservice-Aufrufes über das Preisvergleichportal. Wie bei den anderen Fällen, werden am Anfang die Policies des Dienstes gelesen (**1.**). Vom STS des Preisvergleichportals muss ggfs ein benötigter Security-Token angefordert werden (Schritt **3.**). Theoretisch ist auch denkbar, dass ein weiterer Security-Token

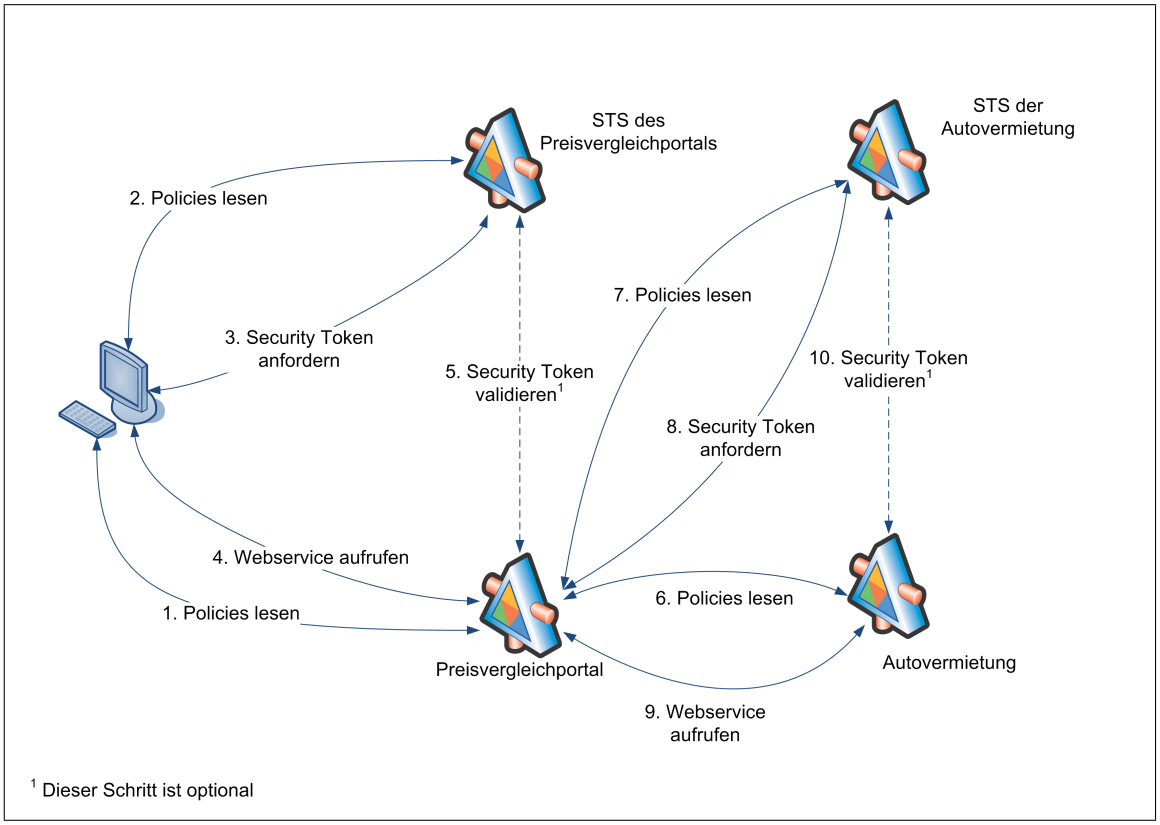

Abbildung 4.3: Grobkonzept: Preisvergleichportal

benötigt wird, der beim STS der Autovermietung angefordert werden muss (z.B. zum Verschlüsseln der Kreditkarteninformationen, so dass sie nur von der Autovermietung gelesen werden können).

Mit diesen Security Tokens kann der Aufruf des Preisvergleichportals geschehen (**4.**). Das Preisvergleichportal validiert ggfs „seinen“ Security Token (**5.**) und fordert selbst einen Security Token für den Dienst der Autovermietung an (**8.**), nachdem es die Policies des Webservice zur Fahrzeugreservierung und des STS der Autovermietung gelesen hat (**6.+7.**). Mit dem angeforderten Security Token und dem zusätzlichen Token des Benutzers kann der Dienst der Autovermietung aufgerufen werden (**9.**). Der Dienst der Autovermietung prüft anhand der Tokens, ob die Kreditkarteninformationen des Benutzers korrekt sind und ob das Portal berechtigt ist, die Autoreservierung für den Benutzer vorzunehmen, und lässt die Tokens ggfs durch seinen STS validieren (**10.**).

4.2 Entwurf eines Sicherheitsprotokolls

Die WS-*-Spezifikationen stellen lediglich bestimmte Funktionen zur Verfügung, die zum Sichern einer Anwendung benötigt werden. Damit die Spezifikationen eingesetzt werden können, muss ein Protokoll entworfen werden, das festlegt, wie die Anwendung mit den durch die Spezifikationen bereitgestellten Funktionen gesichert werden kann. Im Folgenden wird ein solches Protokoll für das Anwendungsszenario aus Kapitel 3.2 entworfen.

4.2.1 Sicherung des Webservice

Um den Webservice zum Reservieren eines Fahrzeugs nutzen zu können, wird ein Security-Token benötigt, der vom STS der Autovermietung ausgestellt wurde. Die Authentifizierung von Benutzern wird dabei komplett an den STS delegiert. Dieser Security-Token enthält einen Hybrid-Key, der zur Verschlüsselung und Signierung der Nachrichten verwendet wird, die zwischen dem Benutzer und dem Webservice zur Fahrzeugreservierung ausgetauscht werden. Der symmetrische Schlüssel in dem Hybrid-Key wird mit dem öffentlichen Schlüssel der Autovermietung verschlüsselt, und der Hybrid-Key wird mit dem privaten Schlüssel des STS signiert. Dadurch wird sichergestellt, dass nur Security-Tokens des STS der Autovermietung akzeptiert werden und der Security-Token nur von der Autovermietung verwendet werden kann.

Ein Benutzer des Webservice muss vor dem Aufruf des Webservice den benötigten Security-Token des Webservice beim STS der Autovermietung anfordern. Um den Hybrid-Key aus dem Security-Token verwenden zu können, sendet der STS zusätzlich zu dem Security-Token denselben Schlüssel an den Benutzer, diesmal so verschlüsselt, dass der Benutzer ihn entschlüsseln kann. Dieser zweite Hybrid-Key wird in WS-Trust als „Proof-of-Possession-Token“ bezeichnet, da der Benutzer durch diesen Schlüssel nachweisen kann, dass er berechtigt ist den Security-Token zu verwenden.

Dadurch, dass die Kommunikation zwischen dem Benutzer und dem Webservice der Autovermietung mit dem vom STS erstellten Schlüssel verschlüsselt und signiert wird, ist sichergestellt, dass kein unberechtigter die Kommunikation zwischen den beiden ausspionieren kann. Die Authentizität der beiden Kommunikationspartner ist ebenfalls sichergestellt, vorausgesetzt, dass beide dem STS vertrauen, dass nur ein berechtigter Benutzer und der Webservice den symmetrischen Schlüssel erhalten haben.

Der Security-Token entspricht von seiner Funktionalität her einem Service-Ticket bei Kerberos. Im Gegensatz zu Kerberos wird hier anstelle des Service-Keys ein Public-Key-Verfahren zum Übertragen des Sitzungsschlüssels an den Dienst verwendet. Während bei Kerberos ein Ticket-Granting-Ticket benötigt wird, um ein Service-Ticket zu erhalten, kann der STS

beliebige Möglichkeiten anbieten um den Security-Token anzufordern. Im Folgenden wird beschrieben, wie der Security-Token in den einzelnen Anwendungsfällen ausgestellt wird.

4.2.2 Stammkunde

Der Stammkunde kann sich der Autovermietung gegenüber mit seiner Kundennummer und seinem Passwort authentifizieren. Um eine maximale Sicherheit zu gewährleisten, soll das Passwort nicht über die Leitung gesendet werden, sondern ähnlich wie bei Kerberos zur Signierung und Verschlüsselung des Datenverkehrs zwischen dem STS und dem Benutzer genutzt werden.

Aus dem Passwort des Benutzers muss nach einem festgelegten Algorithmus ein symmetrischer Schlüssel generiert werden. Mit diesem Schlüssel wird die Nachricht, die zur Anforderung des Security-Tokens an den STS gesendet wird, signiert und verschlüsselt. Zusätzlich muss die Nachricht einen Username-Token enthalten, an dem der STS erkennt, von welchem Benutzer die Anforderung kommt. Aus der Benutzerdatenbank der Autovermietung kann der STS das Passwort des Benutzers lesen. Der STS generiert aus dem Passwort des Benutzers nach dem festgelegten Algorithmus den symmetrischen Schlüssel. Wenn die Nachrichtensignatur mit diesem Schlüssel überprüft werden kann, ist der Benutzer authentifiziert und der Security-Token kann ausgestellt werden.

Der ausgestellte Security-Token wird zusammen mit dem Proof-of-Possession an den Benutzer gesendet. Die Antwortnachricht wird ebenfalls mit dem aus dem Passwort generierten Schlüssel signiert und verschlüsselt. Daran erkennt der Benutzer, dass er tatsächlich mit dem STS der Autovermietung kommuniziert, da das Passwort nur dem Benutzer und der Autovermietung bekannt ist.

Der Webservice kann die Gültigkeit des Security-Tokens anhand der Signatur des STS überprüfen. Diese Überprüfung kann der Webservice selbst vornehmen, so dass Schritt **5.** aus Abbildung 4.1 entfallen kann.

Schematisch stellt sich das Protokoll für die Schritte **3. + 4.** aus Abbildung 4.1 wie folgt dar. Die Principals A und B stehen für den Benutzer und den Webservice der Autovermietung. Der Principal S steht für den STS der Autovermietung.

$$(3a)\ A \rightarrow S : \langle \{\text{Request Token}\}_{K_{AS}} \rangle_{K_{AS}}$$

$$(3b)\ S \rightarrow A : \langle \{ \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token} , \underbrace{\{K_{AB}\}_{K_{AS}}}_{Proof-of-Possession} \}_{K_{AS}}, \rangle_{K_{AS}}$$

$$(4a)\ A \rightarrow B : \langle \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token} , \{\text{Reserviere Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

$$(4b)\ B \rightarrow A : \langle \{\text{Bestätige Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

4.2.3 Unbekannter Benutzer

Da der Benutzer der Autovermietung nicht bekannt ist, muss er ein Zertifikat einer vertrauenswürdigen CA[5] vorzeigen, das seine Identität bestätigt. Zusätzlich muss er eine gültige Kreditkarte nachweisen.

Der Webserviceaufruf zum Anfordern des Security-Tokens muss mit dem privaten Schlüssel des Benutzers signiert werden. Zusätzlich muss das Zertifikat des Benutzers mitgesendet werden, in dem die Identifikationsmerkmale und der öffentliche Schlüssel des Benutzers enthalten sind. Wenn das Zertifikat gültig ist und die Signatur der Webservice-Nachricht mit dem öffentlichen Schlüssel des Benutzers überprüft werden kann, ist sichergestellt, dass der Benutzer derjenige ist, für den er sich ausgibt. Um sicherzustellen, dass die Nachricht nicht ausspioniert werden kann, muss sie verschlüsselt werden. Dazu wird ein zufälliger Schlüssel generiert, mit dem die Nachricht symmetrisch verschlüsselt wird. Dieser Schlüssel wird mit dem öffentlichen Schlüssel des STS verschlüsselt und mit der Nachricht mitgesendet, so dass nur der STS in der Lage ist, die Nachricht zu entschlüsseln.

Wenn der Benutzer authentifiziert werden kann und die Nachricht zusätzlich gültige Kreditkarteninformationen enthält, erstellt der STS den Security-Token, den der Benutzer benötigt, um den Webservice zur Fahrzeugreservierung aufzurufen. Der Security-Token wird zusammen mit dem Proof-of-Possession-Token an den Benutzer gesendet. Damit der Benutzer sicher sein kann, dass die Antwort wirklich von dem STS kommt, wird sie mit dem privaten Schlüssel des STS signiert. Damit die Antwort nicht ausspioniert werden kann, muss sie verschlüsselt werden. Dazu wird abermals ein symmetrischer Schlüssel generiert, mit dem die Nachricht verschlüsselt wird. Dieser Schlüssel wird mit dem öffentlichen Schlüssel des Benutzers verschlüsselt und mit der Antwort mitgesendet, so dass nur der Benutzer in der Lage ist, die Nachricht zu entschlüsseln und damit in den Besitz des Security-Tokens und des Proof-of-Possession-Tokens zu gelangen.

[5] Certification Authority

Der Webservice kann die Gültigkeit des Security-Tokens anhand der Signatur des STS überprüfen. Diese Überprüfung kann der Webservice selbst vornehmen, so dass Schritt **5.** aus Abbildung 4.1 entfallen kann.

Schematisch stellt sich das Protokoll für die Schritte **3. + 4.** aus Abbildung 4.1 wie folgt dar. Die Principals A und B stehen für den Benutzer und den Webservice der Autovermietung. Der Principal S steht für den STS der Autovermietung.

$$(3a)\ A \rightarrow S : \langle \{\text{Request Token, Kreditkarte}\}_{K1}, \{K1\}_{K_S} \rangle_{K_A^{-1}}$$

$$(3b)\ S \rightarrow A : \langle \{ \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token}, \underbrace{\{K_{AB}\}_{K_A}}_{Proof-of-Possession} \}_{K2}, \{K2\}_{K_A} \rangle_{K_S^{-1}}$$

$$(4a)\ A \rightarrow B : \langle \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token}, \{\text{Reserviere Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

$$(4b)\ B \rightarrow A : \langle \{\text{Bestätige Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

4.2.4 Partnerunternehmen

Die Mitarbeiter des Partnerunternehmens authentifizieren sich innerhalb des Unternehmens mit Zertifikaten, die von der internen CA des Unternehmens ausgestellt werden. Dieses Zertifikat kann jedoch nicht verwendet werden, um den Dienst der Autovermietung zu nutzen. Das Zertifikat kann den Benutzer zwar authentifizieren, wenn die Autovermietung der internen CA des Partnerunternehmens vertraut, dadurch ist aber nicht sichergestellt, dass der Benutzer auch berechtigt ist, ein Fahrzeug für sein Unternehmen zu mieten.

Damit das Zertifikat von der Autovermietung als gültig akzeptiert werden kann, muss es durch den STS des Partnerunternehmens bestätigt werden. Dies kann durch einen SAML-Token geschehen. SAML-Tokens bieten zwei unterschiedliche Formen der Bestätigung an. Mit der „Holder-Of-Key“-Bestätigung kann der STS den Benutzer berechtigen, mit seinem firmeninternen Zertifikat den Dienst der Autovermietung zu nutzen.

Der SAML-Token enthält das Zertifikat des Benutzers, das mit dem privaten Schlüssel des STS des Partnerunternehmens signiert ist. Der STS der Autovermietung muss das Zertifikat des STS des Partnerunternehmens kennen und dem STS vertrauen. Dadurch kann die Signatur in dem SAML-Token überprüft werden und das Zertifikat des Benutzers akzeptiert werden.

Zum Verschlüsseln und Signieren der Nachrichten zwischen dem Benutzer und beiden STS werden die Zertifikate des Benutzers und des jeweiligen STS verwendet. Der Benutzer erhält wie bei den anderen Anwendungsfällen den Security-Token, mit dem er den Webservice zur Fahrzeugreservierung aufrufen kann. Die Schritte **7. + 8.** aus Abbildung 4.2 können deshalb

entfallen, da der Webservice den Security-Token selbst validieren kann. Die schematische Darstellung der Nachrichten der Schritte **4. - 6.** ergibt sich wie folgt. Dabei stehen die Principals A, B und S wie bei den vorigen Anwendungsfällen für den Benutzer, den Webservice der Autovermietung und den STS der Autovermietung. Für den STS des Partnerunternehmens wird zusätzlich der Principal T eingeführt.

$$(4a)\ A \rightarrow T : \langle \{\text{Request Token}\}_{K1}, \{K1\}_{K_T} \rangle_{K_A^{-1}}$$

$$(4b)\ T \rightarrow A : \langle \{ \underbrace{\langle K_A \rangle_{K_T^{-1}}}_{SAML-Token} \}_{K2}, \{K2\}_{K_A} \rangle_{K_T^{-1}}$$

$$(5a)\ A \rightarrow S : \langle \{ \underbrace{\langle K_A \rangle_{K_T^{-1}}}_{SAML-Token} , \text{Reserviere Auto} \}_{K3}, \{K3\}_{K_S} \rangle_{K_A^{-1}}$$

$$(5b)\ S \rightarrow A : \langle \{ \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token} , \underbrace{\{K_{AB}\}_{K_A}}_{Proof-of-Possession} \}_{K4}, \{K4\}_{K_A} \rangle_{K_S^{-1}}$$

$$(6a)\ A \rightarrow B : \langle \underbrace{\langle \{K_{AB}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token} , \{\text{Reserviere Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

$$(6b)\ B \rightarrow A : \langle \{\text{Bestätige Auto}\}_{K_{AB}} \rangle_{K_{AB}}$$

4.2.5 Preisvergleichportal

Das Preisvergleichportal kann sich gegenüber der Autovermietung wie ein unbekannter Benutzer über ein Zertifikat authentifizieren. Da das Preisvergleichportal das Fahrzeug nicht selbst mieten möchte, muss sich zusätzlich der Benutzer gegenüber der Autovermietung authentifizieren und seine Kreditkarteninformationen angeben. Die Kreditkarteninformationen sollen jedoch nicht für das Preisvergleichportal sichtbar sein.

Der Benutzer ruft nicht den Webservice der Autovermietung, sondern den Webservice des Preisvergleichportals auf. Das Preisvergleichportal führt keine Authentifizierung oder Autorisierung des Benutzers durch, so dass es nicht sinnvoll ist, diese Funktionalität in einen STS auszulagern. Der STS aus Abbildung 4.3 und die Schritte **2.**, **3.** und **5.** entfallen somit.

Beim Aufrufen des Webservice muss der Benutzer seine Kreditkarteninformationen angeben, die mit dem öffentlichen Schlüssel der Autovermietung im Hybrid-Verfahren verschlüsselt und mit dem privaten Schlüssel des Benutzers signiert sind. Um Wiedereinspielungsangriffe zu verhindern, müssen die Kreditkarteninformationen des Benutzers mit einem Timestamp angereichert sein. Die Kommunikation zwischen dem Benutzer und dem Preisvergleichportal wird in bekannter Weise mit deren Zertifikaten geschützt.

Der Webservice des Preisvergleichportals kann mit den Kreditkarteninformationen des Benutzers und seinem eigenen Zertifikat beim STS der Autovermietung den Security-Token anfordern, der von dem Webservice zur Fahrzeugreservierung benötigt wird. Die schematische Darstellung der ausgetauschten Nachrichten der Schritte **4.**, **8.** und **9.** aus Abbildung 4.3 ergibt sich wie folgt. Die Principals A, B, und C stehen dabei für den Benutzer, die Autovermietung und das Preisvergleichportal. Der Principal S steht für den STS der Autovermietung. Die generierten Hybrid-Keys werden zur besseren Unterscheidung nummeriert ($K1$ und $K2$). Das Validieren des Security-Tokens (Schritt **10.**) kann wie bei den bisherigen Anwendungsfällen entfallen.

$$(4a)\ A \rightarrow C : \langle \{\text{Kreditkarte}\}_{K1}, \{K1\}_{K_S}, \text{Timestamp} \rangle_{K_A^{-1}}, \langle \{\text{Reserviere Auto}\}_{K2}, \{K2\}_{K_C} \rangle_{K_A^{-1}}$$

$$(8a)\ C \rightarrow S : \langle \{\text{Kreditkarte}\}_{K1}, \{K1\}_{K_S}, \text{Timestamp} \rangle_{K_A^{-1}}, \langle \{\text{Request Token}\}_{K3}, \{K3\}_{K_S} \rangle_{K_C^{-1}}$$

$$(8b)\ S \rightarrow C : \langle \{ \underbrace{\langle \{K_{BC}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token}, \underbrace{\{K_{BC}\}_{K_C}}_{Proof-of-Possession} \}_{K4}, \{K4\}_{K_C} \rangle_{K_S^{-1}}$$

$$(9a)\ C \rightarrow B : \langle \underbrace{\langle \{K_{BC}\}_{K_B} \rangle_{K_S^{-1}}}_{Security-Token}, \{\text{Reserviere Auto}\}_{K_{BC}} \rangle_{K_{BC}}$$

$$(9b)\ B \rightarrow C : \langle \{\text{Bestätige Auto}\}_{K_{BC}} \rangle_{K_{BC}}$$

$$(4b)\ C \rightarrow A : \langle \{\text{Bestätige Auto}\}_{K5}, \{K5\}_{K_A} \rangle_{K_C^{-1}}$$

4.3 Sicherheitsarchitektur

Das Framework, das in dieser Arbeit entworfen wird, ist für die Sicherstellung der Anwendungssicherheit zuständig. Neben der Anwendungssicherheit muss in einem globalen Szenario zusätzlich die Perimetersicherheit berücksichtigt werden, mit der die IT-Infrastruktur, in der die Anwendung läuft, gegen Angriffe abgesichert wird.

4.3.1 Perimetersicherheit

Ein wesentlicher Bestandteil heutiger Sicherheitsarchitekturen sind Firewalls, die in der Regel dazu eingesetzt werden, ein privates Netzwerk vor Angriffen aus einem öffentlichen Netzwerk, wie z.B. dem Internet, zu schützen, in dem der komplette Datenverkehr zwischen den Netzen über die Firewall erfolgen muss. Die Firewall hat dadurch die Möglichkeit, gefährliche Datenpakete zu löschen. Firewalls können auch intern eingesetzt werden, um verschiedene Bereiche eines Netzwerks voneinander zu trennen oder die Zugriffskontrolle für technische

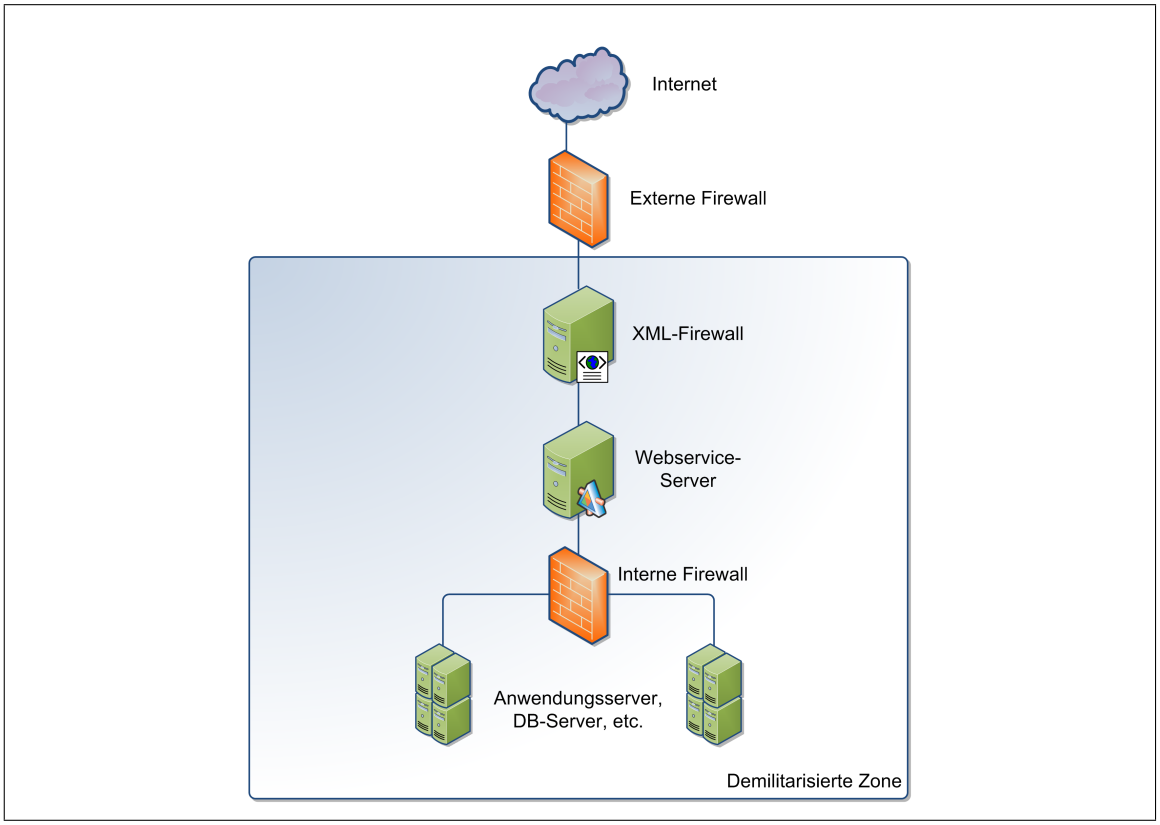

Abbildung 4.4: Konzeptionelle Sicherheitsarchitektur (Vgl. Steel u. a., 2006)

Dienste (z.B. FTP[6]- oder Druckdienste) durchzuführen. Firewalls lassen sich in die folgenden Klassen einteilen (Vgl. Eckert, 2006):

Paketfilter Prüft anhand von definierten Regeln, ob ein Paket weitergeleitet werden darf oder nicht. Bei den Regeln werden beispielsweise die IP-Adressen des Senders und des Empfängers, der verwendete Port und das verwendete Protokoll berücksichtigt.

Proxy-Firewall Die Firewall stellt einen generischen Proxy zur Verfügung, an den die Datenpakete eines Benutzers gesendet werden. Die Firewall kann dann z.B. Zugriffskontrollen durchführen und bei Erfolg das Paket an den tatsächlichen Dienst weiterleiten.

Applikationsfilter Im Gegensatz zur Proxy-Firewall stellt ein Applikationsfilter einen dedizierten Proxy zur Verfügung, der nicht nur allgemeine Prüfungen durchführen kann, sondern

[6] File Transfer Protocol

auch anwendungsspezifisches Wissen hat und somit auch spezielle Prüfungen durchführen kann. Ein SMTP-Filter kann z.B. eingehende Emails auf schädliche Anhänge wie z.B. Viren und „Zip-Bomben“[7] überprüfen.

In der Abbildung 4.4 ist dargestellt an welchen Stellen Firewalls eingesetzt werden sollten.

Webservices sind darauf ausgelegt, herkömmliche Firewalls zu umgehen, in dem sie den Standard-Port für HTTP verwenden, der in der Regel bei jeder Firewall freigeschaltet ist. Da das SOAP-Protokoll auf XML basiert, können Webservice-Nachrichten problemlos die Firewalls passieren. XML und SOAP bieten jedoch diverse Angriffsmöglichkeiten (Vgl. Zedlitz, 2006):

XML-Angriffspunkte

- Zu große XML-Dokumente können ein DOM-basierten XML-Parser „in die Knie zwingen“
- Rekursive Entity-Deklarationen, die bei Nichterkennen dazu führen, dass der XML-Parser sämtlichen Hauptspeicher verbraucht. Solche Deklarationen können auch über mehrere Rechner verteilt werden, so dass alle beteiligten Rechner beeinträchtigt werden.
- Entity-Referenzen auf das lokale Dateisystem, die es ermöglichen anhand von Fehlermeldungen Informationen über die Existenz von bestimmten Dateien zu erlangen.
- Durch das Verwenden von „unendlich“ langen Namen für Elemente und Attribute kann der XML-Parser dazu gebracht werden, extrem viel Hauptspeicher zu verbrauchen.
- Extrem lange Attributwerte können nicht erkannt werden, weil der Attributtyp erst am Ende des Attributs definiert wird.

SOAP-Angriffspunkte

- Bestimmte SOAP-Elemente (z.B. Header, Envelope und Fault) können beliebige unbekannte Elemente enthalten, die normalerweise nicht verarbeitet werden, aber dazu genutzt werden können, beliebig große XML-Dokumente zu erzeugen.
- SOAP-Routing ist in der Praxis noch kaum erprobt und enthält daher wahrscheinlich viele Schwachstellen.

[7] Als Zip-Bomben werden Zip-Dateien bezeichnet, die zwar sehr klein sind, beim Entpacken aber beliebig große Dateien enthalten.

- Wenn die WSDL-Beschreibung eines Webservice beliebig lange Sequenzen in der Eingabe erlaubt (maxOccurs='unbounded'), können beliebig große XML-Dokumente erzeugt werden.

Um diese Angriffspunkte zu verhindern, sollte eine XML- bzw. Webservice-Firewall eingesetzt werden (Vgl. Zedlitz, 2006). XML-Firewalls können die SOAP-Nachrichten selbst parsen und auf syntaktische Korrektheit überprüfen. Zusätzlich ist es möglich, bestimmte Restriktionen festzulegen, durch die bestimmte Angriffe verhindert werden. Das Basic Profile der WSI[8](Vgl. Ballinger u. a., 2006) ist ein Standard, der Restriktionen für Webservices festlegt, die solche Angriffe verhindern und die Interoperabilität von Webservices, die diesen Standard unterstützen, plattformübergreifend sicherstellen.

4.3.2 Anwendungssicherheit

Da das zu entwerfende Framework für die Anwendungssicherheit zuständig ist, muss es selbst dicht an der Anwendungsschicht sein. Es gibt vorhandene Ansätze die vorschlagen die Sicherheitsprotokolle in einem „Message Interceptor Gateway“ abzuhandeln, das vor der eigentlichen Webservice-Infrastruktur sitzt (Vgl. Steel u. a., 2006, Kapitel 11). Diese Ansätze sind für Szenarien geeignet, in denen Dienste anderer Unternehmen in den eigenen Workflow eingebaut werden und die benötigten Benutzerinformationen zentral verfügbar sind. Wenn ein Webservice über eine Client-Anwendung direkt von einem Endanwender aufgerufen wird, muss es möglich sein, das Framework direkt in die Anwendung zu integrieren. Auch wenn für die Public-Key-Verfahren z.B. Smart-Cards eingesetzt werden sollen, muss die Verschlüsselung der Daten direkt auf dem Rechner des Anwenders vorgenommen werden und kann nicht von einem zentralen Gateway übernommen werden.

In Abbildung 4.5 ist dargestellt, wie sich das Framework in die konzeptionelle Sicherheitsarchitektur einordnet. Das Framework muss als Bestandteil in die Webservice-Middleware integriert werden, oder als Decorator (Vgl. Gamma u. a., 1995, Decorator-Pattern) um die Middleware gesetzt werden. Ein solches Framework kann sowohl auf einem zentralen Server eingesetzt und z.B. an ein zentrales LDAP[9]-System angeschlossen werden, als auch lokal in eine Client-Anwendung eingebunden werden. Damit das Framework die unterschiedlichen Anforderungen, die sich aus diesen unterschiedlichen Betriebsarten ergeben, erfüllen kann, muss sein Verhalten was z.B. das Ermitteln von Benutzerinformationen (Zertifikate, Passwörter, etc.) hoch-konfigurierbar sein.

[8]Web Services-Interoperability Organization
[9]Lightweight Directory Access Protocol

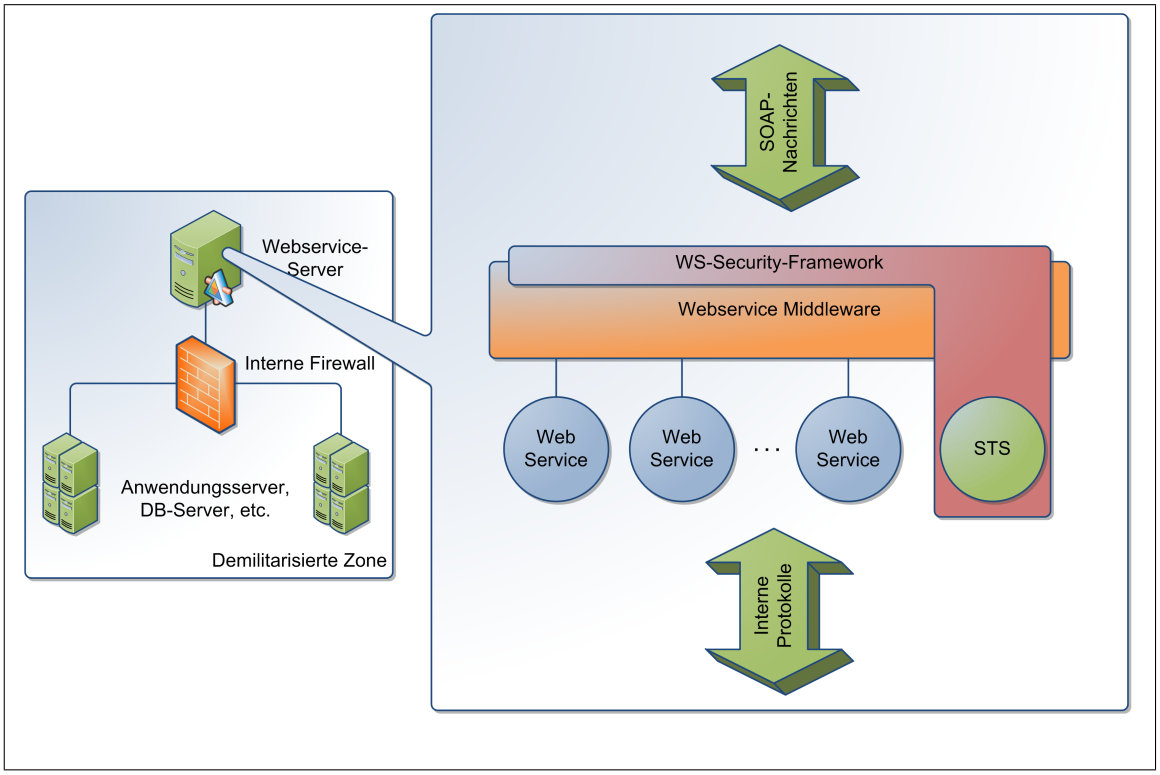

Abbildung 4.5: Einordnung des Frameworks in die Sicherheitsarchitektur

4.4 Entwurf des Frameworks

Das Framework ist unabhängig von dem verwendeten Sicherheitsprotokoll für jede Anwendung, die über Webservices kommuniziert, einsetzbar. Die unterschiedlichen Protokolle werden von dem Framework generisch verarbeitet. Dazu müssen die Protokolle in den Policies der Webservices spezifiziert werden. Aus den Policies kann das Framework die notwendigen Maßnahmen ableiten und auf der Seite des Aufrufers durchführen bzw. auf der Seite des aufgerufenen Webservice überprüfen.

Die Verarbeitung auf der Seite des Aufrufers lässt sich in folgende Schritte einteilen:

1. Ermitteln der Policies
2. Benötigte Security-Tokens erstellen bzw. anfordern
3. Nachricht signieren und verschlüsseln

Auf der Seite des aufgerufenen Webservice sind folgende Schritte notwendig:

1. Ermitteln der Policies

2. Überprüfen der benötigten Security-Tokens
3. Überprüfen der geforderten Verschlüsselungs- und Signaturverfahren
4. Entschlüsseln der Nachricht
5. Überprüfen der Signaturen

4.4.1 Komponenten des Frameworks

Entsprechend den Spezifikationen lässt sich das Framework in die folgenden Komponenten einteilen:

- WS-Security
- WS-Policy
- WS-Trust
- SOAP Message Security
- Security Token Service
- XML Encryption
- XML Signature

In Abbildung 4.6 sind die Abhängigkeiten der Komponenten untereinander und zu der Anwendung dargestellt, die im Folgenden kurz beschrieben werden. Bei den Komponenten Security-Adapter, den Token- und den Callback-Handlern handelt es sich um individuelle Komponenten, die teilweise anwendungsspezifisch sind. Das Framework bietet für diese Komponenten Standardbausteine an, die gängige Szenarien unterstützen.

WS-Security Diese Komponente ist die Schnittstelle des Frameworks, über die das Framework in eine Anwendung eingebunden werden kann. An diese Komponente werden noch nicht gesicherte ausgehende Nachrichten zur Verschlüsselung und Signierung und gesicherte eingehende Nachrichten zur Validierung und Entschlüsselung übergeben.

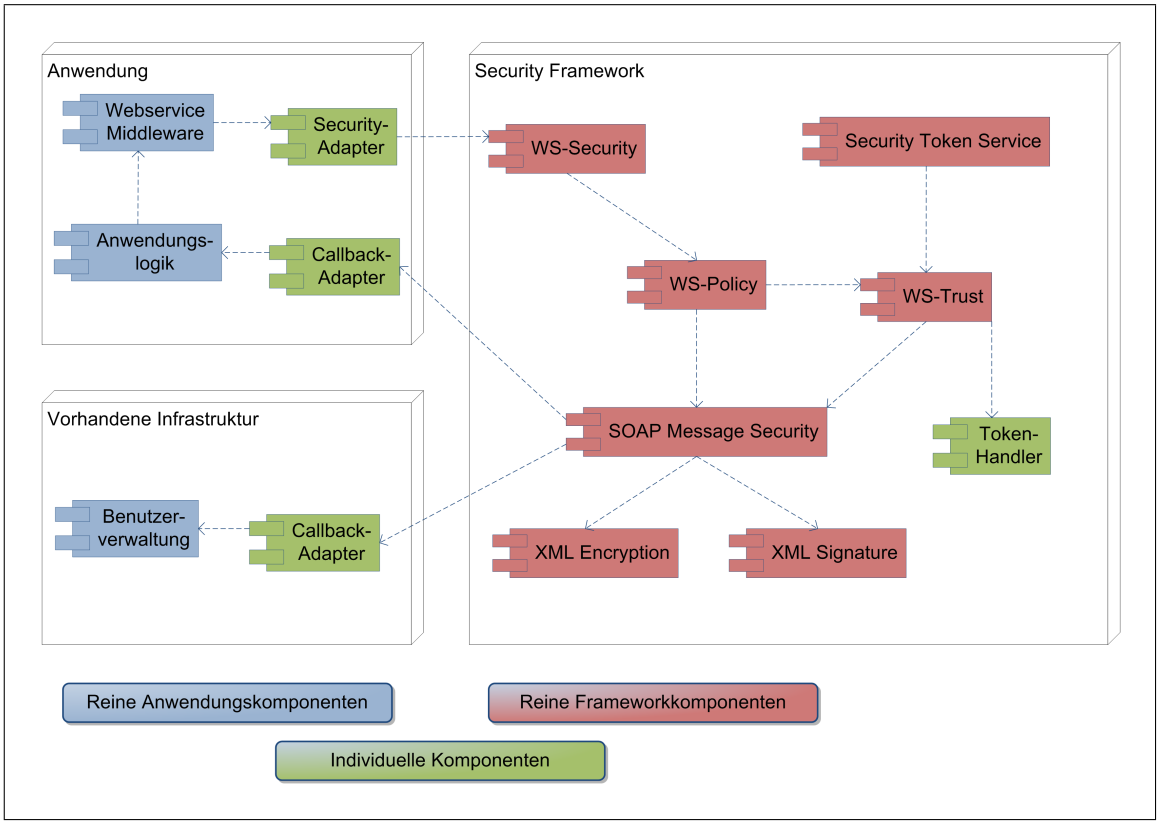

Abbildung 4.6: Komponentendiagramm des Framework

WS-Policy Da die benötigten Sicherheitsmaßnahmen in den Policies eines Webservice definiert sind, ist diese Komponente der Koordinator für die Verarbeitung der Sicherheitsmaßnahmen. Bei ausgehenden Nachrichten sorgt diese Komponente dafür, dass die geforderten Maßnahmen umgesetzt werden. Dazu ruft sie für jede Maßnahme die jeweils zuständige Komponente auf. Bei eingehende Nachrichten prüft diese Komponente, ob alle geforderten Sicherheitsmaßnahmen umgesetzt sind. Zusätzlich beauftragt sie die zuständigen Komponenten, die Security-Tokens und die Signaturen zu überprüfen und verschlüsselte Daten zu entschlüsseln.

WS-Trust Diese Komponente ist für das Ausstellen, Validieren und Verlängern von Security-Tokens zuständig. Ggfs muss dazu der STS einer anderen Anwendung aufgerufen werden.

Token-Handler Damit das Framework in der Lage ist, beliebige Tokens auszustellen, werden die Tokens nicht direkt von der WS-Trust-Komponente ausgestellt, sondern von Token-

Handlern. So ist es leicht möglich das Framework um einen neuen Token-Typ zu erweitern, in dem ein neuer Token-Handler erstellt wird.

Security Token Service Die STS-Komponente dient als Webservice-Schnittstelle für das WS-Trust-Modul.

SOAP Message Security Diese Komponente ist für das Einbinden von Security-Tokens in die SOAP-Nachrichten, die Ver- und Entschlüsselung und die Signierung von Daten zuständig. Die Verschlüsselung und Signierung wird dabei an die Komponenten **XML Encryption** und **XML Signature** delegiert. Des Weiteren ist diese Komponente auch für das Ermitteln von Authentifikationsdaten (wie z.B. Benutzerinformationen oder Zertifikate) zuständig.

Callback-Handler Damit das Ermitteln der Authentifikationsdaten an das jeweilige Szenario, in dem das Framework eingesetzt wird, angepasst werden kann, werden dafür Callback-Handler eingesetzt, die wie die Token-Handler dazu verwendet werden können, das Framework an individuelle Anforderungen anzupassen. Wird das Framework in einer Client-Anwendung verwendet, wird beispielsweise ein Callback-Handler benötigt, der einen Dialog zum Eingeben des Benutzernamens und des Passworts anzeigt. Wird das Framework in einem Anwendungsserver eingesetzt, könnte ein Callback-Handler beispielsweise Benutzerinformationen aus der Benutzerverwaltung des Unternehmens auslesen. Für Standardkomponenten (wie z.B. ein LDAP[10]-System) kann das Framework fertige Callback-Handler anbieten.

Security-Adapter Der Security-Adapter wird benötigt um das Framework in eine vorhandene Webservice-Middleware (wie z.B. Axis 2) einzubinden. Dadurch wird das Framework unabhängig von der verwendeten Middleware und kann in beliebigen Umgebungen verwendet werden. Denkbar wäre hier z.B. in einer Java EE-Umgebung[11] anstelle eines Adapters für eine Webservice-Middleware ein Servlet-Filter, der als Decorating-Filter (Vgl. Bien, 2003) vor das Webservice-Servlet gehängt wird und alle ein- und ausgehenden Webservice-Nachrichten zur Verarbeitung an das Framework übergibt. Des Weiteren ist der Security-Adapter dafür zuständig, die geltenden Policies zu ermitteln.

[10] Lightweight Directory Access Protocol
[11] Java Enterprise Edition

4.4.2 Konkretisierung der Komponenten

Nachdem im vorherigen Abschnitt die Komponenten des Frameworks vorgestellt wurden, werden die einzelnen Bestandteile der Komponenten näher betrachtet und deren Funktionsweise erklärt. In Abbildung 4.7 ist das Klassendiagramm für die Komponente WS-Security dargestellt.

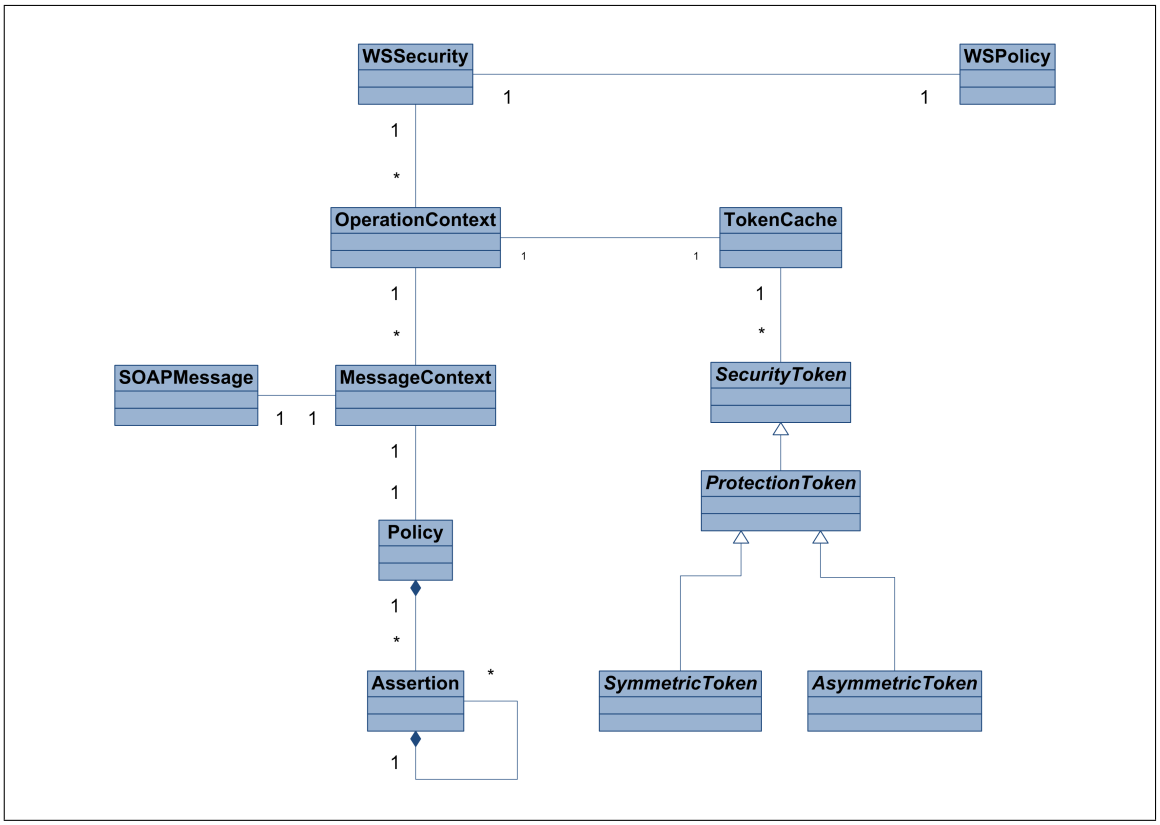

Abbildung 4.7: Klassendiagramm für die Komponente WS-Security

Die zentrale Klasse des WS-Security-Moduls ist die Klasse `WSSecurity`. Diese Klasse wird von dem Security-Adapter der Anwendung aufgerufen und bekommt eine SOAP-Nachricht und deren Policies übergeben. Dabei werden Aufrufe von ein- und ausgehenden Nachrichten unterschieden. Bei ausgehenden Nachrichten müssen die in den Policies geforderten Sicherheitsmaßnahmen umgesetzt werden. Bei eingehenden Nachrichten müssen die Sicherheitsmaßnahmen überprüft werden und verschlüsselte Daten müssen entschlüsselt werden.

Eine weitere Unterscheidung muss zwischen Request- und Response-Nachrichten gemacht werden, da bei der Verarbeitung von Response-Nachrichten ggfs Daten aus der Verarbeitung der Request-Nachricht benötigt werden, wie z.B. Security-Tokens, die in der Request-

Nachricht mitgesendet wurden und zum Sichern der Response-Nachricht benötigt werden. Damit diese Daten beim Verarbeiten der Response-Nachricht zur Verfügung stehen, wird der `OperationContext` benötigt. In dieser Klasse werden alle Informationen gespeichert, die zum Verarbeiten einer Nachricht notwendig sind. Die einzelnen Nachrichten, die zu einer Operation gehören (i.d.R. die Request- und die Response-Nachricht), werden als `MessageContext` im `OperationContext` abgelegt. Im `MessageContext` sind die `SOAPMessage` selbst und die `Policy` enthalten, die aus mehreren `Assertion`s besteht, die wiederum mehrere `Assertion`s enthalten können.

`SecurityToken`s, die beim Verarbeiten der Request-Nachricht auf der Client-Seite erstellt bzw. auf der Server-Seite empfangen werden, können im `TokenCache` des `OperationContext` gespeichert werden und sind somit beim Verarbeiten der Response-Nachricht verfügbar. Es können beliebige Tokens gespeichert werden. Zum Verschlüsseln und Signieren werden im wesentlichen `ProtectionToken`s benötigt. Dabei wird zwischen `SymmetricToken`s (wie z.B. Kerberos-Tokens) und `AsymmetricToken`s (wie z.B. X509-Tokens) unterschieden.

Die Verarbeitung der SOAP-Nachrichten wird von der Komponente WS-Policy gesteuert. Dazu ruft die Klasse `WSSecurity` die Klasse `WSPolicy` auf und übergibt ihr den `OperationContext`. Die Klassen der Komponente WS-Policy und der weiteren WS-*-Komponenten sind in Abbildung 4.8 dargestellt.

Die Klasse `WSPolicy` durchläuft die `Policy` der SOAP-Nachricht und ruft für die Verarbeitung der einzelnen `Assertions` den jeweils zuständigen `AssertionHandler` auf. Die beiden wichtigsten `AssertionHandler` sind die `TokenAssertionHandler` und die `BindingAssertionHandler`. `TokenAssertionHandler` sind dafür zuständig, dass benötigte Tokens zur Verfügung gestellt oder bei eingehenden Nachrichten validiert werden. Sollte ein benötigter Token nicht im `TokenCache` gespeichert sein, muss er über die Klasse `WSTrust` angefordert werden, die den Token von dem zuständigen `TokenHandler` erstellen lässt oder bei einem STS anfordert.

Die `BindingAssertionHandler` sorgen dafür, dass die für das Binding benötigten Tokens erstellt werden und führen die im Binding geforderten Maßnahmen wie die Verschlüsselung und Signierung von Daten durch. Die Verschlüsselung und Signierung wird dabei an die Klasse `SOAPMessageSecurity` delegiert. Wie die Daten verschlüsselt oder signiert werden, wird durch die `EncryptionHandler` und die `SignatureHandler` festgelegt. Dadurch ist es möglich die Verschlüsselung oder Signierung von einer Smart-Card durchführen zu lassen. Die Klassen `XMLEncryption` und `XMLSignature` stellen die korrekte XML-Darstellung der verschlüsselten Daten bzw. der Signaturen sicher. Da zum Ver- oder Entschlüsseln ggfs weitere Informationen benötigt werden, als in den Security-Tokens der Nachricht vorhanden sind, wie z.B. der private Schlüssel, der zu einem Zertifikat gehört,

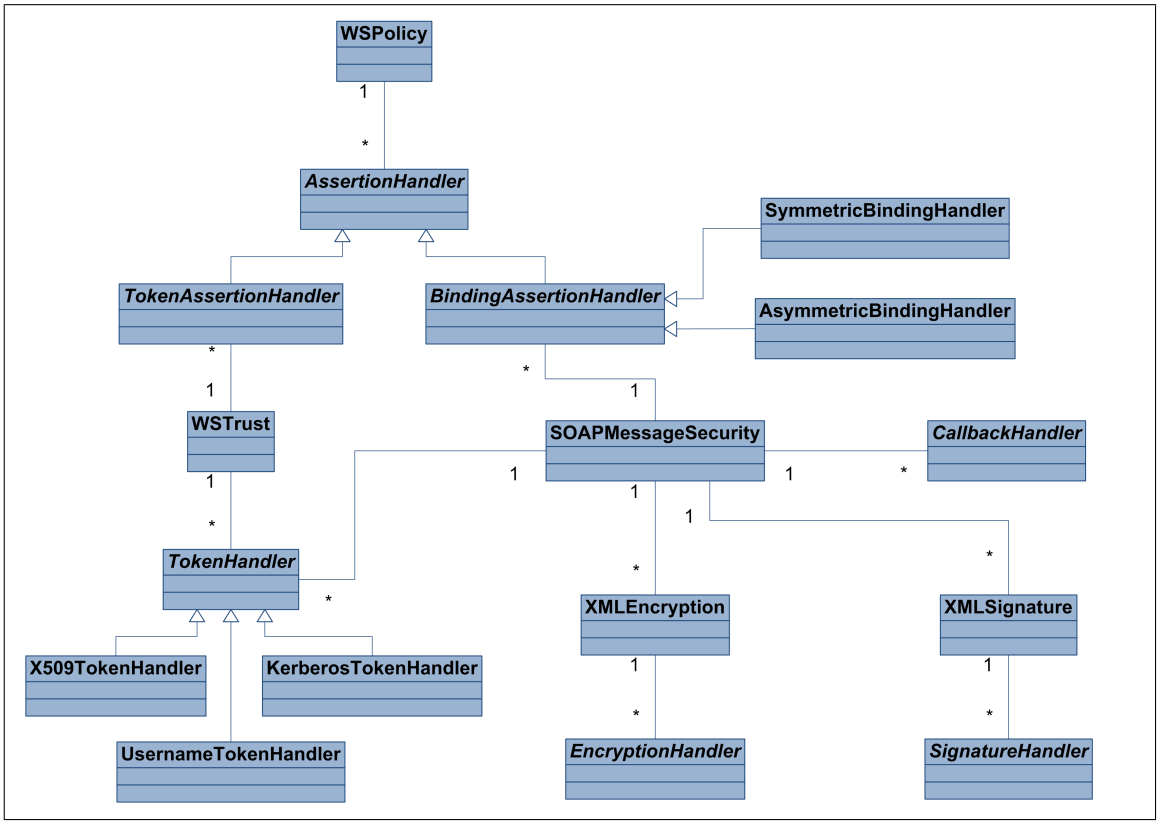

Abbildung 4.8: Klassendiagramm für die WS-*-Komponenten

benötigt die Klasse `SOAPMessageSecurity` den Zugriff auf die `CallbackHandler`, über die diese Informationen ermittelt werden können.

`TokenHandler` benötigen Zugriff auf die Klasse `SOAPMessageSecurity` um Daten in Security-Tokens verschlüsseln zu lassen oder um auf Zertifikate (z.B. zum Erstellen eines `X509Token`s) oder andere Authentifikationsdaten zuzugreifen.

Welche Handler zur Laufzeit eingesetzt werden, hängt von den Policies des Webservice und von der Konfiguration des Frameworks ab. Durch Einsetzen des Factory-Pattern (Vgl. Gamma u. a., 1995) kann die jeweilige Implementierung der benötigten Handler flexibel ausgetauscht werden.

4.4.3 Verarbeitung einer Request-Nachricht

Als Beispiel für die Verarbeitung der SOAP-Nachrichten durch das Framework wird ein Aufruf genommen, bei dem die Nachrichten asymmetrisch verschlüsselt und signiert werden,

wie z.B. bei dem Aufruf des STS der Autovermietung durch den unbekannten Benutzer. In Abbildung 4.9 ist dargestellt, wie eine ausgehende Nachricht verarbeitet wird.

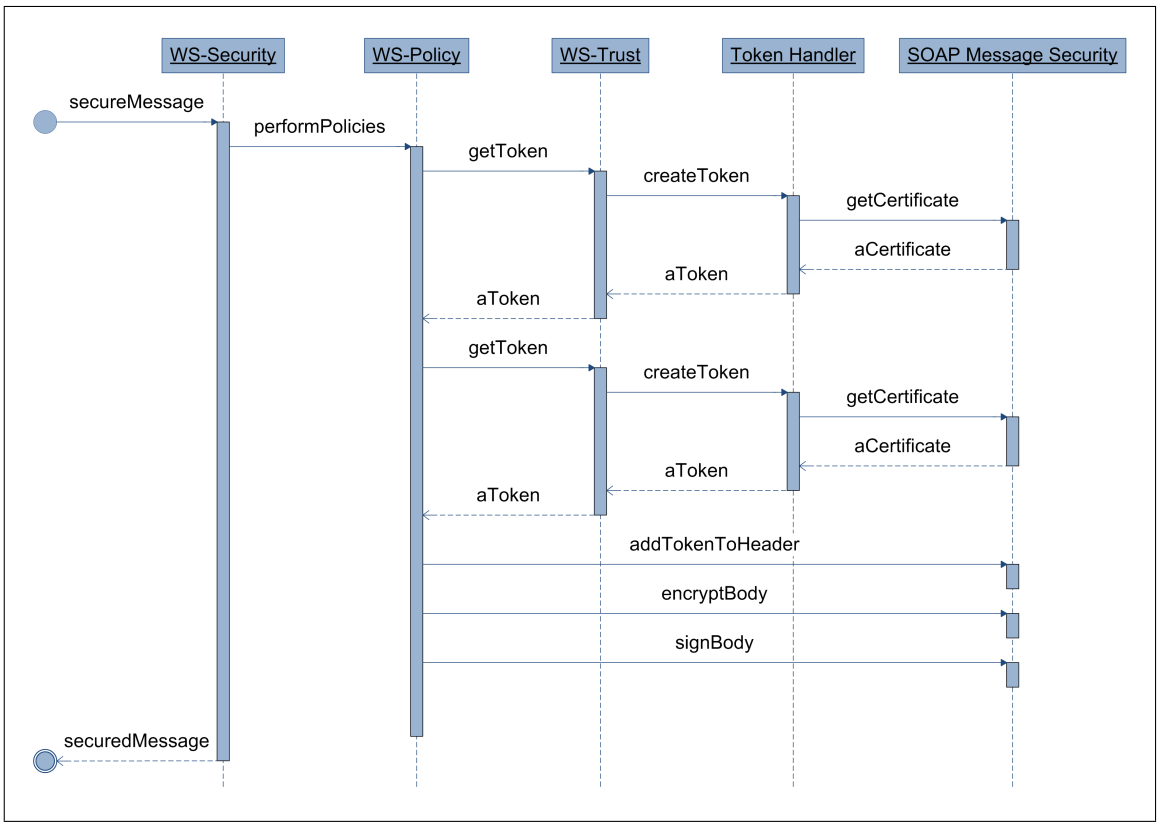

Abbildung 4.9: Verarbeitung einer ausgehenden Nachricht

Das Framework bekommt die zu sichernde Nachricht übergeben und startet die Verarbeitung der Policies. Bei ausgehenden Nachrichten besteht die Aufgabe des Frameworks darin, die in den Policies geforderten Sicherheitsmaßnahmen umzusetzen. Im ersten Schritt müssen die erforderlichen Security-Tokens erstellt werden. Dazu wird die WS-Trust-Komponente aufgerufen. Die WS-Trust-Komponente ruft zum Erstellen des Tokens den zuständigen Token-Handler auf. In diesem Beispiel werden zwei X.509-Tokens benötigt. Zum Erstellen eines X.509-Tokens wird das entsprechende Zertifikat benötigt. Das Zertifikat kann von der SOAP-Message-Security-Komponente erfragt werden, die über den Zugriff auf den benötigten Callback-Handler verfügt[12].

Damit der aufgerufene Webservice ermitteln kann, von wem er aufgerufen wurde, muss das Zertifikat des Aufrufers mit in die SOAP-Nachricht geschrieben werden. Dazu wird die Komponente SOAP-Message-Security aufgerufen, die für das Einfügen von Security-Tokens in SOAP-Nachrichten zuständig ist. Zum Verschlüsseln der Daten und zum Signieren der Daten

[12]Der Callback-Handler ist aus Platzgründen nicht im Sequenzdiagramm dargestellt.

wird ebenfalls die Komponente SOAP-Message-Security aufgerufen. Die gesicherte SOAP-Nachricht wird zurück gegeben und kann versendet werden. Die Verarbeitung der Request-Nachricht auf der Serverseite ist in Abbildung 4.10 dargestellt.

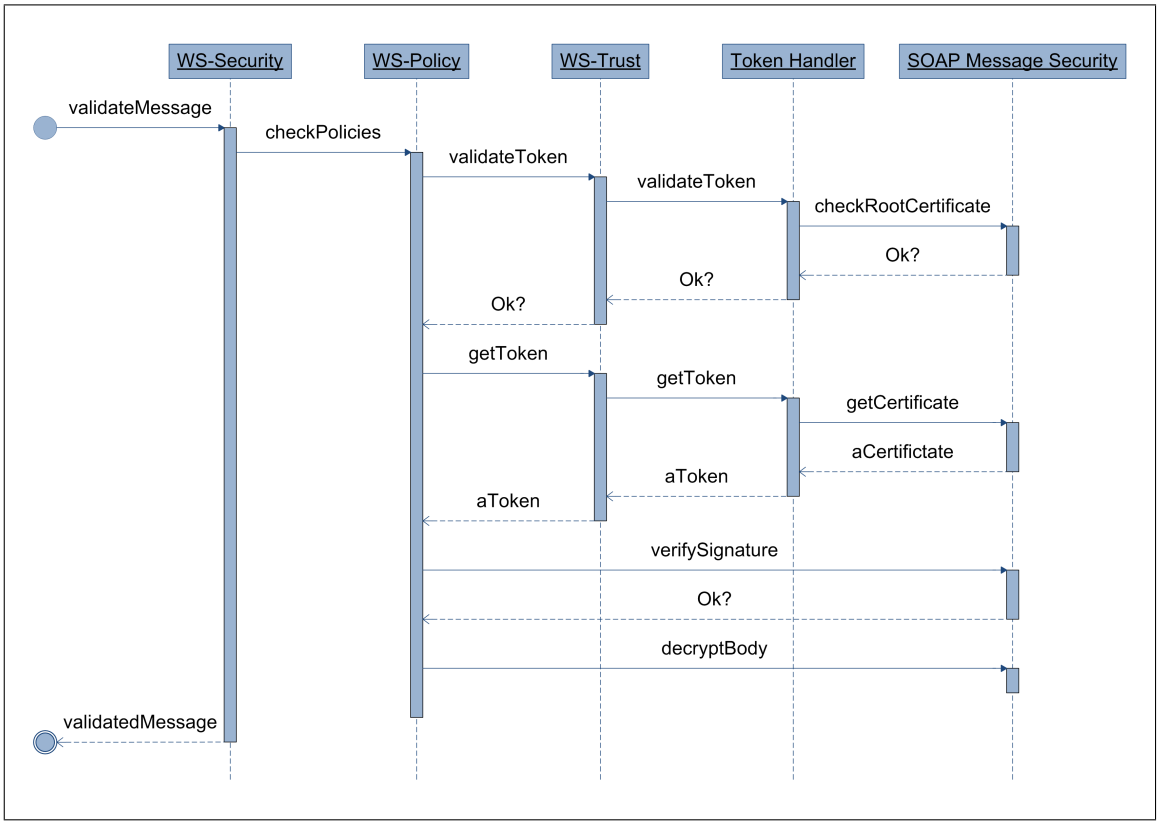

Abbildung 4.10: Verarbeitung einer eingehenden Nachricht

Das Framework bekommt die gesicherte SOAP-Nachricht übergeben und startet die Verarbeitung der Policies. Bei eingehenden Nachrichten müssen die in den Policies geforderten Sicherheitsmaßnahmen überprüft werden und verschlüsselte Daten müssen entschlüsselt werden. Im ersten Schritt müssen die mitgesendeten Security-Tokens geprüft werden. In dem Beispiel ist ein X509-Token in der Nachricht enthalten. Der Token wird zur Überprüfung an die WS-Trust-Komponente übergeben, die den entsprechenden Token-Handler aufruft, um das Zertifikat zu validieren. Um ein Zertifikat zu validieren, muss dessen Root-Zertifikat, also das Zertifikat der ausstellenden CA bekannt und als vertrauenswürdig eingestuft sein. Um das Root-Zertifikat zu überprüfen, wird die Komponente SOAP-Message-Security aufgerufen, die über den entsprechenden Callback-Handler das Root-Zertifikat überprüfen kann.

Das X.509-Zertifikat des Webservice selbst wurde nicht mitgesendet. Daher muss es wie auf der Client-Seite von der WS-Trust-Komponente erstellt werden. Sind alle benötigten

Security-Tokens überprüft und als gültig eingestuft worden, können die Signatur der Nachricht überprüft und die verschlüsselten Daten entschlüsselt werden. Dazu wird abermals die Komponente SOAP-Message-Security aufgerufen. Wenn die Signatur gültig ist und die Daten entschlüsselt wurden, kann die überprüfte Nachricht zur weiteren Verarbeitung zurückgegeben werden.

Die Verarbeitung einer Response-Nachricht läuft im wesentlichen identisch mit der Verarbeitung der Request-Nachricht ab. Der wesentliche Unterschied besteht darin, dass die benötigten Security-Tokens nicht über die WS-Trust-Komponente erstellt werden müssen, sondern aus dem Token-Cache gelesen werden können.

4.5 Implementierung des Sicherheitsprotokolls

Das Framework ist in der Lage, beliebige Sicherheitsprotokolle abzuhandeln, die den WS-Security-Spezifikationen entsprechen. Solange keine individuellen Erweiterungen (wie z.B. eigene Security-Tokens) verwendet werden, muss das Framework auch bei einer Protokolländerung nicht angepasst werden.

Um das in Kapitel 4.2 entworfene Protokoll mit dem Framework umzusetzen, muss das Protokoll in Form von Policies beschrieben werden. Aus den Policies kann das Framework dynamisch zur Laufzeit ermitteln welche Sicherheitsmaßnahmen für die einzelnen Nachrichten notwendig sind und diese umsetzen.

Die XML-Listings in diesem Kapitel sind aus Gründen der Übersichtlichkeit verkürzt dargestellt[13].

4.5.1 Policy für den Webservice

Das in Kapitel 4.2 entworfene Protokoll sieht vor, dass die Nachrichten, die mit dem Webservice zur Fahrzeugreservierung ausgetauscht werden, symmetrisch verschlüsselt werden. Der Schlüssel dafür soll von dem STS der Autovermietung erzeugt werden und in einen Security-Token verpackt werden, der von dem Benutzer angefordert und mit dem eigentlichen Webservice-Aufruf zur Fahrzeugreservierung mitgesendet wird. Eine vereinfachte Darstellung der entsprechenden Policy ist in Listing 4.1 zu sehen.

[13] Im Wesentlichen wurden verschachtelte *wsp:Policy*-Elemente weggelassen, um die Verschachtelung der Daten möglichst flach zu halten.

```
1  <wsp:Policy>
2    <sp:SymmetricBinding>
3        <sp:ProtectionToken>
4        <sp:IssuedToken>
5          <sp:Issuer>
6            <wsa:Address>http://www.car24.com/STS</wsa:Address>
7          </sp:Issuer>
8
9          <sp:RequestTokenTemplate>
10           <wst:TokenType>car24:sessionToken</wstTokenType>
11         </sp:RequestTokeTemplate>
12       </sp:IssuedToken>
13     </sp:ProtectionToken>
14
15     <sp:EncryptedParts>
16       <sp:Body/>
17     </sp:EncryptedParts>
18
19     <sp:SignedParts>
20       <sp:Body/>
21     </sp:SignedParts>
22
23     <sp:AlgorithmSuite>
24       <sp:Basic256 />
25     </sp:AlgorithmSuite>
26   </sp:SymmetricBinding>
27 </wsp:Policy>
```

Listing 4.1: Policy für den Webservice zur Fahrzeugreservierung

Dass die Nachricht symmetrisch verschlüsselt werden soll, wird durch die Assertion *sp:SymmetricBinding* angegeben (Zeilen 2-26). Der Schlüssel für die Verschlüsselung und die Signierung wird durch den Parameter *sp:ProtectionToken* angegeben (Zeilen 3-13). In diesem Fall wird als Protection-Token ein Security-Token benötigt, der von dem STS der Autovermietung ausgestellt wird. Dies wird mit der Assertion *sp:IssuedToken* angegeben (Zeilen 4-12). Mit der Angabe *sp:Issuer* kann die Adresse des STS angegeben werden, der in der Lage ist den benötigten Security-Token zu erstellen (Zeilen 5-7). Des Weiteren muss der Typ des benötigten Tokens angegeben werden (Zeilen 9-11).

Der Benutzer benötigt kein Wissen über den Aufbau des Security-Tokens, da dieser den Token nur weiterleitet. Wenn der Benutzer den ausgestellten Token zum Verschlüsseln und Signieren der Nachricht verwenden soll, kann er dies durch den „Proof-of-Possession-Token“, den der STS beim Ausstellen des Security-Tokens mitsendet. Der in diesem Beispiel verwendete Security-Token, wird unten genauer beschrieben.

WS-Trust bietet noch weitere Möglichkeiten den benötigten Security-Token genauer zu spezifizieren, z.B. für welchen Dienst er benötigt wird. In diesem Beispiel wird jedoch davon ausgegangen, dass der STS nur Security-Tokens für den Autovermietungsservice ausstellt und daher weiß, wie der Token zu erstellen ist.

Für den Benutzer muss noch angegeben werden, welche Teile der Nachricht verschlüsselt und signiert werden sollen und welche Algorithmen dafür verwendet wird. Dies wird mit den Parametern *sp:EncryptedParts* (Zeilen 15-17), *sp:SignedParts* (Zeilen 19-19) und *sp:AlgorithmSuite* (Zeilen 23-25) gemacht. Für die verwendeten Algorithmen stellt WS-SecurityPolicy bereits einige vordefinierte Konstanten zur Verfügung, die für die verschiedenen Sicherheitsmechanismen wie die Verschlüsselung und die Signierung entsprechende Algorithmen und Schlüssellängen festlegen.

Der Security-Token, der vom STS ausgestellt wird, beinhaltet den Hybrid-Key, der zum Verschlüsseln der Nachricht verwendet wird, und eine Signatur, die belegt, dass der Schlüssel vom STS ausgestellt wurde. Der Aufbau des Security-Tokens ist in Listing 4.2 zu sehen.

```
1  <car24:sessionToken>
2    <xenc:EncryptedKey wsu:Id="Car24SessionToken">
3      <xenc:EncryptionMethod Algorithm="#RSA-1 5" />
4
5      <ds:KeyInfo>
6        <wsse:KeyIdentifier ValueType="#X509v3" EncodingType="#Base64">
7          M874hds87fznvhv...
8        </wsse:KeyIdentifier>
9      </ds:KeyInfo>
10
11     <xenc:CipherData>
12       <xenc:CipherValue>
13         Hni7uzg5K87dGjgusd...
14       </xenc:CipherValue>
15     </xenc:CipherData>
16   </xenc:EncrptedKey>
17
18   <ds:Signature>
19     <ds:SignedInfo>
20       <ds:CanonicalizationMethod Algorithm="xml-exc-c14n#"/>
21       <ds:SignatureMethod Algorithm="#rsa-sha1"/>
22       <ds:Reference URI="#Car24SessionToken">
23         <ds:DigestMethod Algorithm="#sha1"/>
24         <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
25       </ds:Reference>
26     </ds:SignedInfo>
27
28     <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
29
30     <ds:KeyInfo>
31       <wsse:KeyIdentifier ValueType="#X509v3" EncodingType="#Base64">
32         jkhgH76gHf7u...
33       </wsse:KeyIdentifier>
34     </ds:KeyInfo>
35   </ds:Signature>
36 </car24:sessionToken>
```

Listing 4.2: Aufbau des Security-Tokens

Der symmetrische Schlüssel muss in dem Security-Token so verschlüsselt sein, dass er nur von der Autovermietung entschlüsselt werden kann. Er ist deshalb als *xenc:EncryptedKey*

in dem Token enthalten (Zeilen 2-16). Damit der symmetrische Schlüssel entschlüsselt werden kann, muss bekannt sein, mit welchem Algorithmus und mit welchem Schlüssel er verschlüsselt wurde. Dazu dienen die Angaben *xenc:EncryptionMethod* (Zeile 3) und *ds:KeyInfo* (Zeilen 5-9). Als *ds:KeyInfo* wird das X.509-Zertifikat der Autovermietung angegeben.

Damit die Autovermietung sicher sein kann, dass der Security-Token von ihrem STS ausgestellt wurde, muss der Schlüssel signiert werden. Die Signatur ist in dem *ds:Signature*-Element enthalten (Zeilen 18-35). Damit die Signatur überprüft werden kann, werden einige Informationen über die Signatur benötigt, die in dem *ds:SignedInfo*-Element enthalten sind (Zeilen 19-26). In dem *ds:Reference*-Element wird angegeben, welche Teile des Tokens signiert wurde (Zeilen 22-25).

Damit die Benutzerdaten über den Security-Token einer Fahrzeugreservierung zugeordnet werden können, sollten zusätzlich entweder die Benutzerdaten selbst oder eine ID, die auf z.B. in einer Datenbank gespeicherten Daten verweist, in dem Token enthalten sein. Des Weiteren sollte der Security-Token mit einem Timestamp oder mit einer Nonce versehen werden, damit er nicht für Wiedereinspielungsangriffe missbraucht werden kann. Da diese beiden Punkte trivial umzusetzen sind, werden sie hier aus Gründen der Vereinfachung nicht berücksichtigt.

4.5.2 Stammkunde

Die Authentifizierung eines Stammkunden wird über dessen Passwort erreicht. Damit das Passwort nicht ausspioniert werden kann, wird es nicht übertragen sondern zum Signieren und Verschlüsseln der Nachrichten zwischen dem Benutzer und dem STS verwendet. Dazu muss aus dem Passwort ein symmetrischer Schlüssel abgeleitet werden. Wie abgeleitete Schlüssel verwendet werden, ist in WS-SecureConversation (Lawrence und Kaler, 2006f) beschrieben. Die Policy des STS für den Stammkunden ist in Listing 4.3 beschrieben.

```
1  <wsp:Policy>
2    <sp:SymmetricBinding>
3      <sp:ProtectionToken>
4        <sp:UserNameToken IncludeToken="IncludeToken/AlwaysToRecipient">
5          <sp:NoPassword />
6          <sp:RequireDerivedKeys />
7        </sp:UserNameToken>
8      </sp:ProtectionToken>
9
10     <sp:EncryptedParts>
11       <sp:Body/>
12     </sp:EncryptedParts>
13
14     <sp:SignedParts>
15       <sp:Body/>
16     </sp:SignedParts>
17
18     <sp:AlgorithmSuite>
```

```
19        <sp:Basic256 />
20      </sp:AlgorithmSuite>
21    </sp:SymmetricBinding>
22  </wsp:Policy>
```

Listing 4.3: Policy für den STS (Stammkunde)

Die *sp:SymmetricBinding*-Assertion gibt wie in der Policy für den Webservice zur Fahrzeugreservierung an, dass die Nachricht symmetrisch verschlüsselt und signiert wird. Der Unterschied zu der Policy des Webservice besteht in dem Security-Token, der für das Signieren und für die Verschlüsselung verwendet wird. In diesem Fall wird ein *UsernameToken* verwendet (Zeilen 4-7). Durch die Properties *NoPassword* (Zeile 5) und *RequireDerivedKeys* (Zeile 6) wird angegeben, dass das Passwort nicht übertragen werden soll, sondern ein Schlüssel aus dem UserNameToken abgeleitet werden soll.

In der Spezifikation WS-SecureConversation wird spezifiziert, wie in Webservice-Nachrichten die Regeln für das Ableiten von Schlüsseln angegeben werden. Wie diese Regeln in einer Policy definiert werden, wird nicht spezifiziert. Auch in WS-SecurityPolicy und in WS-Policy wird keine Möglichkeit beschrieben, wie diese Regeln angegeben werden können. Um abgeleitete Schlüssel nutzen zu können, müssen sich die Kommunikationspartner außerhalb der Policies auf ein anzuwendendes Verfahren verständigen. Dies widerstrebt dem Ansatz des Frameworks, unabhängig von dem eingesetzten Sicherheitsprotokoll zu sein. Die Authentifizierung über das Passwort ist deshalb nicht mit aus dem Passwort abgeleiteten Schlüsseln realisierbar.

4.5.3 Unbekannter Benutzer

Damit ein unbekannter Benutzer den Security-Token beim STS der Autovermietung anfordern kann, muss er sich nach dem Protokoll aus Kapitel 4.2 mit seinem X.509-Zertifikat authentifizieren. Dazu wird die Nachricht an den STS mit dem privaten Schlüssel des Benutzers signiert, und die Antwort des STS mit dem öffentlichen Schlüssel aus dem Zertifikat im Hybrid-Verfahren verschlüsselt, so dass sichergestellt ist, dass nur der Benutzer die Antwort lesen kann.

Damit der STS ebenfalls authentifiziert werden kann, wird die Anfrage an den STS mit dessen öffentlichen Schlüssel im Hybrid-Verfahren verschlüsselt, und der STS signiert seine Antwort mit seinem privaten Schlüssel. Die entsprechende Policy für diese Anforderung ist in Listing 4.4 zu sehen.

```
1  <wsp:Policy>
2    <sp:RequiredElements>
3      <sp:xPath>
4        /Envelope/Header/CreditCard
5      </sp:xPath>
```

```
6   </sp:RequiredElements>
7
8   <sp:AsymmetricBinding>
9     <sp:InitiatorToken>
10      <wsp:Policy>
11        <sp:X509V3Token IncludeToken="IncludeToken/AlwaysToRecipient" />
12      </wsp:Policy>
13    </sp:InitiatorToken>
14
15    <sp:RecipientToken>
16      <wsp:Policy>
17        <sp:X509V3Token IncludeToken="IncludeToken/Never">
18          <wst:Claims>
19            <ds:X509SubjectName>CN=Car24STS</ds:X509SubjectName>
20          </wstClaims>
21        </sp:X509V3Token>
22      </wsp:Policy>
23    </sp:RecipientToken>
24
25    <sp:AlgorithmSuite>
26      <sp:Basic256 />
27    </sp:AlgorithmSuite>
28
29    <sp:EncryptedParts>
30      <sp:Body/>
31    </sp:EncryptedParts>
32
33    <sp:EncryptedElements>
34      <sp:xPath>
35        /Envelope/Header/CreditCard
36      </sp:xPath>
37    </sp:EncryptedElements>
38
39    <sp:SignedParts>
40      <sp:Body/>
41    </sp:SignedParts>
42
43    <sp:SignedElements>
44      <sp:xPath>
45        /Envelope/Header/CreditCard
46      </sp:xPath>
47    </sp:SignedElements>
48  </sp:AysmmetricBinding>
49 </wsp:policy>
```

Listing 4.4: Policy für den STS (Unbekannter Benutzer)

Die *sp:AsymmetricBinding*-Assertion (Zeilen 8-48) besagt, dass die Verschlüsselung und die Signierung der Nachricht mit einem Public-Key-Verfahren durchgeführt wird. Dazu müssen verschiedene Schlüssel angegeben werden. Prinzipiell werden vier Schlüsselpaare benötigt, da zum Verschlüsseln und zum Signieren in der Regel unterschiedliche Schlüsselpaare verwendet werden. Zur Vereinfachung wird hier davon ausgegangen, dass ein Schlüsselpaar für beide Zwecke verwendet wird. Durch das *sp:AsymmetricBinding* ist festgelegt, dass für die Verschlüsselung ein symmetrischer Schlüssel generiert wird, der mit dem öffentlichen Schlüssel des Empfängers verschlüsselt wird.

Die benötigten Schlüssel werden als *sp:InitiatorToken* (Zeilen 9-13) und *sp:RecipientToken* (Zeilen 15-23) angegeben. Angegeben werden jeweils die X.509-Zertifikate, die den öffentlichen Schlüssel an den jeweiligen Benutzer binden. Mit dem *IncludeToken*-Parameter wird festgelegt, dass der Benutzer sein Zertifikat bei jeder Nachricht an den STS mitsenden muss (Zeile 11), da der STS von beliebig vielen Benutzern aufgerufen werden kann. Das Zertifikat des STS ist fest vorgegeben und ist öffentlich bekannt (Zeilen 17-21). Da es dem STS selbst ebenfalls bekannt ist, muss es nie mitgesendet werden. Wie auch bei der symmetrischen Verschlüsselung muss angegeben werden, welche Algorithmen zu verwenden sind, und welche Teile der Nachricht verschlüsselt und signiert werden müssen.

Zusätzlich zu den Angaben über die notwendige Verschlüsselung muss auch angegeben werden, dass die Kreditkarteninformationen des Benutzers notwendig sind. Dies wird mit der Assertion *sp:RequiredElements* getan (Zeilen 2 - 6). Das XML-Schema für die Kreditkarteninformation ist in Listing 4.5 zu sehen. Die Assertion *sp:RequiredElements* erfordert, dass die Kreditkarteninformationen in den Header der SOAP-Nachricht geschrieben werden. Daher muss mit den Assertion *sp:EncryptedElements* (Zeilen 33-37) und *sp:SignedElements* (Zeilen 43-47 angegeben werden, dass diese Daten zusätzlich zum Body der SOAP-Nachricht verschlüsselt und signiert werden müssen.

```
1  <xs:schema>
2    <xs:complexType name="CreditCard">
3      <xs:sequence>
4        <xs:element minOccurs="1" maxOccurs="1"
5          name="CardNumber" type="xs:string" />
6
7        <xs:element minOccurs="1" maxOccurs="1"
8          name="ExpirationMonth" type="xs:int" />
9
10       <xs:element minOccurs="1" maxOccurs="1"
11         name="ExpirationYear" type="xs:int" />
12
13       <xs:element minOccurs="0" maxOccurs="1"
14         name="VCode" type="xs:string" />
15     </xs:sequence>
16   </xs:complexType>
17 </xs:schema>
```

Listing 4.5: XML-Schema für Kreditkarteninformationen

4.5.4 Partnerunternehmen

Bevor der Mitarbeiter des Partnerunternehmens den Security-Token von dem STS der Autovermietung anfordern kann, muss er zunächst einen SAML-Token von dem STS seines Unternehmens anfordern. Der Aufruf des STS seines Unternehmens wird über die Verwendung von Zertifikaten gesichert. Die Policy für den STS gleicht deshalb im Wesentlichen der Policy des STS für einen unbekannten Benutzer (siehe Listing 4.6).

```
1  <wsp:Policy>
2    <sp:AsymmetricBinding>
3      <sp:InitiatorToken>
4        <wsp:Policy>
5          <sp:X509V3Token IncludeToken="IncludeToken/AlwaysToRecipient" />
6        </wsp:Policy>
7      </sp:InitiatorToken>
8
9      <sp:RecipientToken>
10       <wsp:Policy>
11         <sp:X509V3Token IncludeToken="IncludeToken/Never" />
12         <wst:Claims>
13           <ds:X509SubjectName>CN=PartnercompanySTS</ds:X509SubjectName>
14         </wstClaims>
15       </wsp:Policy>
16     </sp:RecipientToken>
17
18     <sp:AlgorithmSuite>
19       <sp:Basic256 />
20     </sp:AlgorithmSuite>
21
22     <sp:EncryptedParts>
23       <sp:Body/>
24     </sp:EncryptedParts>
25
26     <sp:SignedParts>
27       <sp:Body/>
28     </sp:SignedParts>
29   </sp:AysmmetricBinding>
30 </wsp:policy>
```

Listing 4.6: Policy für den STS des Partnerunternehmens

Einziger Unterschied ist der *X509SubjectName* des Zertifikats des *sp:RecipientToken*s, da hier das Zertifikat des STS des Partnerunternehmens und nicht des STS des Autovermietung verwendet werden muss (Zeile 13).

Mit dem vom STS seines Unternehmens ausgestellten SAML-Token kann der Benutzer den Security-Token für den Webservice der Autovermietung bei dessen STS anfordern. Die Sicherung der Webservice-Aufrufe findet dabei wie bei dem unbekannten Benutzer über Zertifikate statt. Die Policy für den STS der Autovermietung gleicht deshalb im Wesentlichen der Policy für den unbekannten Benutzer (siehe Listing 4.7).

```
1  <wsp:Policy>
2    <sp:AsymmetricBinding>
3      <sp:InitiatorToken>
4        <wsp:Policy>
5          <sp:X509V3Token IncludeToken="IncludeToken/AlwaysToRecipient" />
6        </wsp:Policy>
7      </sp:InitiatorToken>
8
9      <sp:RecipientToken>
10       <wsp:Policy>
11         <sp:X509V3Token IncludeToken="IncludeToken/Never" />
12         <wst:Claims>
```

```
13            <ds:X509SubjectName>CN=Car24STS</ds:X509SubjectName>
14          </wstClaims>
15        </wsp:Policy>
16      </sp:RecipientToken>
17
18      <sp:AlgorithmSuite>
19        <sp:Basic256 />
20      </sp:AlgorithmSuite>
21
22      <sp:EncryptedParts>
23        <sp:Body/>
24      </sp:EncryptedParts>
25
26      <sp:SignedParts>
27        <sp:Body/>
28      </sp:SignedParts>
29
30      <sp:SignedEncryptedSupportingTokens>
31        <sp:SamlToken IncludeToken="IncludeToken/AlwaysToRecipient">
32          <sp:WssSamlV11Token11 .../>
33
34          <wsp:AppliesTo>
35            <wsaAddress>http://www.car24.com/STS</wsa:Address>
36          </wsp:AppliesTo>
37
38          <wst:Claims>
39            <saml:ConfirmationMethod>
40              urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
41            </saml:ConfirmationMethod>
42          </wst:Claims>
43        </sp:SamlToken>
44      </sp:SignedEncryptedSupportingTokens>
45    </sp:AysmmetricBinding>
46  </wsp:policy>
```

Listing 4.7: Policy für den STS der Autovermietung (Partnerunternehmen)

Zusätzlich zu der Policy für den unbekannten Benutzer enthält die Policy für den Benutzer des Partnerunternehmens die Assertion *sp:SignedEncryptedSupportingTokens* (Zeilen 30-44). Diese Assertion stellt sicher, dass ein gültiger SAML-Token erstellt werden kann. In Zeile 31 wird definiert, dass in Nachrichten an den STS immer ein SAML-Token mitgesendet werden muss. In Zeile 32 wird die unterstützte SAML-Version angegeben. Mit der Assertion AppliesTo (Zeilen 34-36) wird angegeben, dass der SAML-Token für den STS der Autovermietung gültig sein muss. Daran kann der STS des Partnerunternehmens feststellen, ob der Benutzer berechtigt ist, den SAML-Token zu erhalten. Als zusätzliche Claims (Zeilen 38-42) wird angegeben, dass der SAML-Token die „Holder-of-Key“-Bestätigung enthalten muss.

4.5.5 Preisvergleichportal

Der Webservice des Preisvergleichportals wird wie der STS der Autovermietung von einem unbekannten Benutzer aufgerufen, mit dem Unterschied, dass die Kreditkarteninformationen

mit einem anderen Schlüssel verschlüsselt werden müssen, als die Nutzdaten aus dem Body der SOAP-Nachricht. Die Policy für das Preisvergleichportal ist in Listing 4.8 zu sehen.

```
<wsp:Policy>
  <sp:RequiredElements>
    <sp:xPath>
      /Envelope/Header/CreditCard
    </sp:xPath>
  </sp:RequiredElements>

  <sp:AsymmetricBinding>
    <sp:InitiatorToken>
      <wsp:Policy>
        <sp:X509V3Token IncludeToken="IncludeToken/AlwaysToRecipient" />
      </wsp:Policy>
    </sp:InitiatorToken>

    <sp:RecipientToken>
      <wsp:Policy>
        <sp:X509V3Token IncludeToken="IncludeToken/Never">
          <wst:Claims>
            <ds:X509SubjectName>CN=Price24</ds:X509SubjectName>
          </wstClaims>
        </sp:X509V3Token>
      </wsp:Policy>
    </sp:RecipientToken>

    <sp:AlgorithmSuite>
      <sp:Basic256 />
    </sp:AlgorithmSuite>

    <sp:EncryptedParts>
      <sp:Body/>
    </sp:EncryptedParts>

    <sp:SignedParts>
      <sp:Body/>
    </sp:SignedParts>
  </sp:AysmmetricBinding>

  <sp:AsymmetricBinding>
    <sp:InitiatorToken>
      <wsp:Policy>
        <sp:X509V3Token IncludeToken="IncludeToken/AlwaysToRecipient" />
      </wsp:Policy>
    </sp:InitiatorToken>

    <sp:RecipientToken>
      <wsp:Policy>
        <sp:X509V3Token IncludeToken="IncludeToken/Never">
          <wst:Claims>
            <ds:X509SubjectName>CN=Car24STS</ds:X509SubjectName>
          </wstClaims>
        </sp:X509V3Token>
      </wsp:Policy>
    </sp:RecipientToken>

    <sp:AlgorithmSuite>
      <sp:Basic256 />
    </sp:AlgorithmSuite>
```

```
58
59      <sp:EncryptedElements>
60        <sp:xPath>
61          /Envelope/Header/CreditCard
62        </sp:xPath>
63      </sp:EncryptedElements>
64
65      <sp:SignedElements>
66        <sp:xPath>
67          /Envelope/Header/CreditCard
68        </sp:xPath>
69      </sp:SignedElements>
70    </sp:AsymmetricBinding>
71  </wsp:policy>
```

Listing 4.8: Policy für das Preisvergleichportal

Um die zwei Sicherheitskontexte für die unterschiedliche Verschlüsselung der Nutzdaten und der Kreditkarteninformationen zu realisieren, werden zwei *sp:AsymmetricBinding*-Assertions verwendet. Das erste Binding (Zeilen 8-36) gibt die Verschlüsselung und Signierung für die Nutzdaten vor. Die *sp:RecipientToken*-Assertion (Zeilen 15-23) gibt dabei an, dass zum Verschlüsseln das Zertifikat des Preisvergleichportals verwendet werden muss. Die Assertions *sp:EncryptedParts* (Zeilen 29-31) und *sp:SignedParts* (Zeilen 33-35) geben an, dass die Nutzdaten, also der SOAP-Body, verschlüsselt und signiert werden sollen.

Das zweite Binding (Zeilen 38-70) gibt die Verschlüsselung und die Signierung für die Kreditkarteninformationen vor. Dabei soll das Zertifikat des STS der Autovermietung zum Verschlüsseln der Daten verwendet werden (Zeile 49). Durch die Assertions *sp:EncryptedElements* (Zeilen 59-63) und *sp:SignedElements* (Zeilen 65-69) wird angegeben, dass die Kreditkarteninformationen verschlüsselt werden müssen.

Die Policy für den STS der Autovermietung muss ebenfalls zwei *sp:AsymmetricBinding*-Assertions enthalten. Der einzige Unterschied zu der Policy des Preisvergleichportals besteht darin, dass als zum Verschlüsseln der Nutzdaten ebenfalls das Zertifikat des STS der Autovermietung verwendet werden muss. Im Gegensatz zur Policy beim unbekannten Benutzer werden zwei Bindings benötigt, da die Kreditkarteninformationen und die Nutzdaten mit unterschiedlichen Schlüsseln signiert sind.

5 Implementierung eines technischen Prototyps

In diesem Kapitel wird der Prototyp beschrieben, der die Machbarkeit des in Kapitel 4.4 entworfenen Framework belegt. Der Prototyp ist in Java implementiert und wird in die Webservice-Middleware Axis 2 (siehe Apache, 2007a) integriert.

5.1 Funktionsumfang

Der Funktionsumfang des Prototyps beschränkt sich auf die Funktionalität, die für die Verarbeitung der Anwendungsfälle „Unbekannter Benutzer“ und „Mitarbeiter eines Partnerunternehmens“ benötigt wird. Da die Verwendung von abgeleiteten Schlüsseln in WS-SecureConversation und WS-SecurityPolicy nicht ausreichend spezifiziert ist, kann der Anwendungsfall „Stammkunde“ nicht umgesetzt werden (Vgl. Kapitel 4.5.2). Im Anwendungsfall „Preisvergleichportal“ müsste das Portal als SOAP-Intermediary agieren. Bei den SOAP-Intermediaries weist die SOAP-Spezifikation selbst Lücken auf (Vgl. Ballinger u. a., 2006, Appendix B):

> SOAP-Intermediaries sind ein unterspezifizierter Mechanismus aus SOAP 1.1, deren Nutzung zusätzliche Vereinbarungen vorraussetzt, die über das SOAP-Protokoll hinausgehen.

Für die Umsetzung der genannten Anwendungsfälle sind folgende Funktionen notwendig:

- Symmetrisches Ver- und Entschlüsseln von Daten
- Erstellen symmetrischer Schlüssel
- Ver- und Entschlüsseln von Daten im Hybrid verfahren
- Symmetrisches Signieren von Daten und Überprüfung der Signatur
- Asymmetrisches Signieren von Daten und Überprüfung der Signatur
- Erstellen und Verarbeiten von X.509-Token

- Erstellen und Verarbeiten von SAML-Token
- Erstellen und Verarbeiten von Car24-Token

5.2 Einbindung in Axis 2

Axis 2 bietet eine Art „Plug-In“-Konzept, mit dem die Webservice-Middleware beliebig erweitert werden kann (Vgl. Frotscher u. a., 2007). Die SOAP-Nachrichten werden direkt vor dem Versenden bzw. direkt nach dem Empfangen durch den „OutFlow“ bzw. den „InFlow“ der AxisEngine geleitet. Der „Out-“ und „InFlow“ sind Pipelines (Vgl. Hohpe und Woolf, 2004, "Pipes and Filters"-Pattern), in denen die SOAP-Nachricht von unterschiedlichen Handlern bearbeitet wird (siehe Abbildung 5.1). Durch Einfügen eines eigenen Handlers, kann die Funktionalität der Pipelines erweitert werden[1].

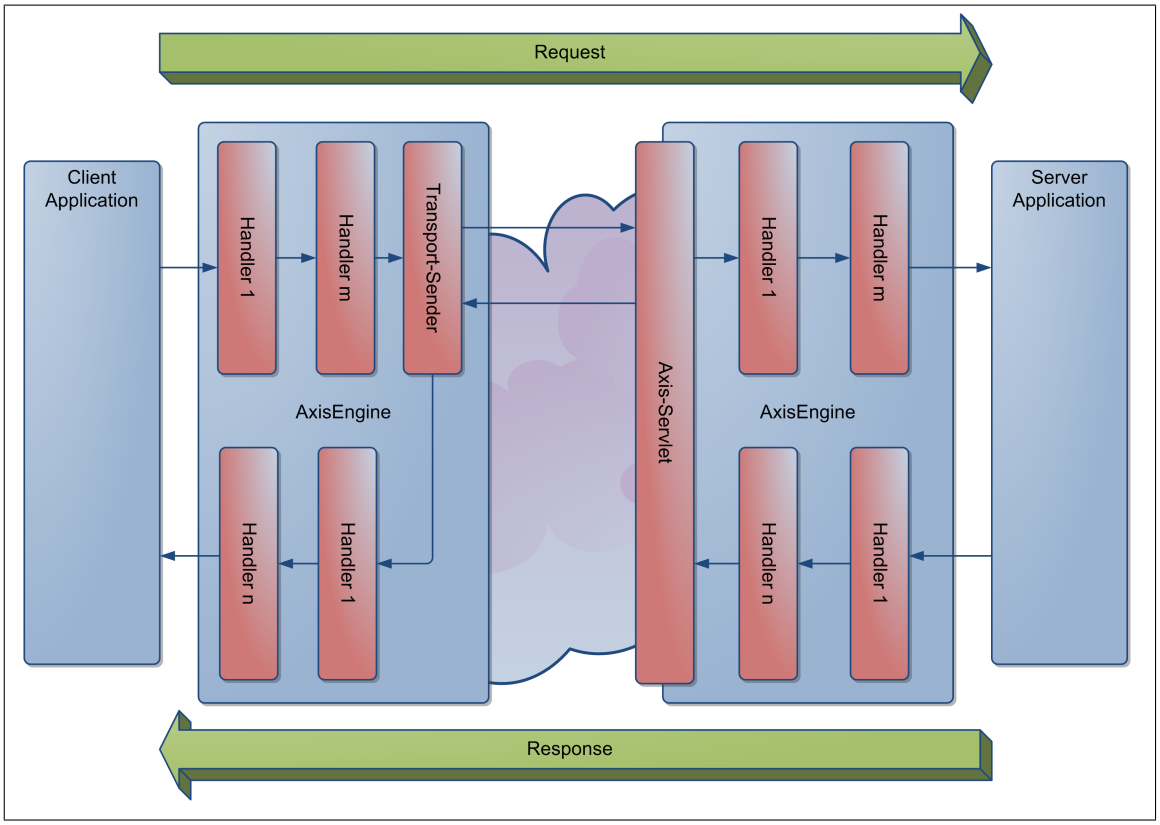

Abbildung 5.1: Vereinfachte Darstellung der AxisEngine

[1] Diese Möglichkeit der Erweiterung wird von den meisten Middleware-Produkten für Webservices angeboten.

Um das Framework in Axis 2 einzubinden wird ein Handler benötigt, der die SOAP-Nachrichten an das Framework übergibt. Dieser Handler übernimmt die Aufgabe des Security-Adapters, der in Kapitel 4.4 vorgestellt wurde. Bei Request-Nachrichten, übergibt der Handler die SOAP-Nachricht zusammen mit deren Policy an das Framework. Das Framework gibt die gesicherte Nachricht und eine Referenz auf den OperationContext zurück. Bei Response-Nachrichten muss der Handler zusätzlich zu der SOAP-Nachricht und deren Policy auch die Referenz auf den OperationContext mit an das Framework übergeben, damit dieses bei Response-Nachrichten auf Tokens aus der Request-Nachricht zugreifen kann.

5.3 Implementierung

Für die Implementierung des Frameworks werden vorhandene Open-Source-Produkte eingesetzt, die im Folgenden kurz vorgestellt werden.

5.3.1 Verarbeitung der SOAP-Nachrichten

Da SOAP-Nachrichten auf XML basieren, wird zum Verarbeiten der Nachrichten ein XML-Parser benötigt. XML-Parser lassen sich in die folgenden Kategorien einteilen:

- Baumbasierte Parser
- Ereignisbasierte Parser
  - Push-Parser
  - Pull-Parser

Bei baumbasierten Parsern wird der komplette XML-Baum im Speicher des Rechners aufgebaut, so dass auf die einzelnen Elemente des Dokuments komfortabel zugegriffen werden kann. Während dieser Ansatz für kleine Dokumente gut geeignet ist, kann es bei großen XML-Dokumenten zu Speicher- und Performanceproblemen kommen.

Die ereignisbasierten Parser verarbeiten das XML-Dokument nicht komplett, sondern arbeiten die Elemente des Dokuments nacheinander ab und übergeben sie der Anwendung, die für jedes Element einen Handler bereitstellen muss, der die anwendungsspezifische Verarbeitung des Elementes vornimmt. Bei den ereignisbasierten Parsern wird zwischen den Push- und den Pull-Parsern unterschieden. Während die Push-Parser das komplette Dokument durchlaufen, kann die Anwendung bei den Pull-Parsern bestimmen, welche Elemente

verarbeitet werden soll und welche nicht. Ein wichtiges API[2] für Pull-Parser ist StAX[3], das durch JSR173 standardisiert wurde (siehe JSR173, 2003).

Da der Prototyp des Frameworks in Axis 2 eingebunden wird, bietet es sich an zum Verarbeiten der SOAP-Nachrichten den XML-Parser zu verwenden, der auch in Axis 2 zum Einsatz kommt. Axis 2 verwendet einen StAX-Parser, der von AXIOM[4], einem eigenen XML-Objektmodell, gekapselt wird (siehe Abbildung 5.2). Durch AXIOM wird der Nachteil, dass bei einem Streaming-API keine Navigation in der Baumstruktur des XML-Dokuments möglich ist, aufgehoben. Bei der Entwicklung von AXIOM wurde stark darauf geachtet, dass die Performance und der geringe Speicherbedarf eines Streaming-APIs erhalten bleiben.

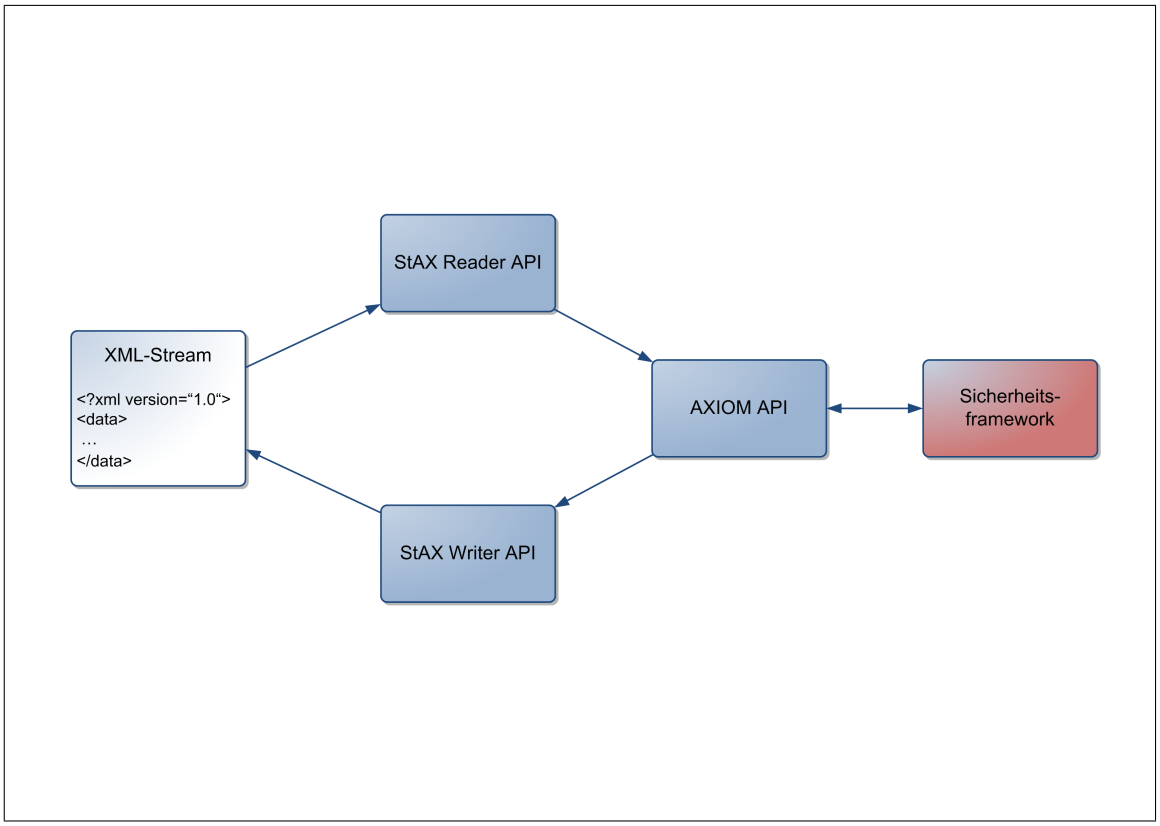

Abbildung 5.2: Funktionsweise von AXIOM (Vgl. Frotscher u. a., 2007)

[2] Application Programming Interface
[3] Streaming API for XML
[4] Apache aXIs Object Model

5.3.2 WS-Policy

Für die Verarbeitung von Policies gibt es von Apache das Projekt Neethi (siehe Apache, 2007b). Neethi bietet ein API für die Verarbeitung, die Serialisierung und Deserialisierung von Policies an. Für die Verarbeitung von Policies im Framework wird das API von Neethi eingesetzt.

5.3.3 WS-Security

Mit WSS4J (siehe Apache, 2007c) bietet Apache ein API für WS-Security an, das folgende Funktionalität zur Verfügung stellt:

- Signieren von SOAP-Nachrichten
- Verschlüsseln von SOAP-Nachrichten
- Erstellen von UserName-Tokens
- Erstellen von SAML-Assertions
- Hinzufügen und Validieren von Timestamps
- Verwalten von Zertifikaten

Für die Verschlüsselung und Signierung verwendet WSS4J das Apache Projekt XML-Security (siehe Apache, 2007d). XML-Security ist eine Implementierung der Standards XML-Encryption und XML-Signature.

Das Framework verwendet WSS4J für die Verschlüsselung und Signierung der SOAP-Nachrichten. Die Verwaltung der Zertifikaten übernimmt das Framework selbst. Durch die Verwendung von WSS4J und die damit verbundene Verwendung von XML-Security können die in Kapitel 4.4.2 vorgestellten Encryption- und Signature-Handler nicht eingesetzt werden, so dass der Prototyp keine SmartCard-Unterstützung anbieten kann.

6 Fazit

In diesem Kapitel wird das entworfene Framework bewertet und es wird ein Ausblick darüber gegeben, welche Entwicklungen notwendig sind, um die Sicherheit in global verteilten Anwendungen „Out-of-the-box“ anbieten zu können.

6.1 Bewertung des entworfenen Frameworks

Hier wird ein Überblick über die Ziele gegeben, die mit dem entworfenen Framework erreicht wurden, und es werden die offenen Punkte festgehalten, die zur Vervollständigung des Frameworks notwendig sind.

6.1.1 Erreichte Ziele

In Kapitel 3 wurden die Anforderungen an das Framework definiert. Für die einzelnen Anforderungen wird geprüft, inwieweit diese von dem Framework erfüllt werden.

Authentifikation Das entworfene Framework ist in der Lage beliebige Authentifizierungsmechanismen zu realisieren. Damit ein konkreter Mechanismus realisiert werden kann, muss dieser mit WS-Policy beschreibbar sein. Durch die flexible Architektur des Frameworks kann der Mechanismus mit entsprechenden Handlern umgesetzt werden.

Verschlüsselung Das Framework unterstützt sämtliche von den Spezifikationen definierten Verschlüsselungsmechanismen und kann diese umsetzen.

Signierung Das Framework unterstützt sämtliche von den Spezifikationen definierten Signierungsmechanismen und kann diese umsetzten.

Föderierte Entitäten Durch die Unterstützung von WS-Trust und das Anbieten des STS werden föderierte Entitäten ansatzweise unterstützt. Da die Autorisierung aufgrund der fehlenden Spezifikation nicht betrachtet wurde, sind zusätzliche Mechanismen, wie z.B. die Verteilung von Benutzerrechten, nicht berücksichtigt worden. Durch Erweiterung des STS um den in WS-Federation (Vgl. Lockhart u. a., 2006) definierten Pseudonym- und Attribut-Provider kann das Framework um die vollständige Unterstützung von föderierten Entitäten ergänzt werden.

Konfiguration der Sicherheit Das Framework führt die notwendigen Sicherheitsmaßnahmen und deren Überprüfung ausschließlich anhand der Policies des betroffenen Webservice durch. Die Policies müssen WS-Policy-konform sein. Damit ist die Interoperabilität des Frameworks mit anderen Implementierungen der WS-Security-Spezifikationen sichergestellt.

Erweiterbarkeit Die konkreten Sicherheitsmaßnahmen, die durch spezielle Policy-Assertions gefordert werden, werden nicht von dem Framework durchgeführt, sondern von entsprechenden Handlern. Das Framework selbst steuert im Wesentlichen die Verarbeitung der Policies und bietet Standard-Handler für die in den Spezifikationen definierten Sicherheitsmechanismen an. Durch individuelle Handler kann das Framework um beliebige Funktionalitäten erweitert werden, die von den Spezifikationen definiert werden.

Wiederverwendbarkeit Durch die hohe Erweiterbarkeit des Frameworks durch die Encryption-, Signature- und Callback-Handler des Moduls SOAP-Message-Security, kann das Framework in beliebigen Szenarien eingesetzt werden. Durch die unabhängige Schnittstelle des Moduls WS-Security kann das Framework in beliebige Webservice-Middleware-Produkte integriert werden, die eine Handler-Architektur wie die von Axis 2 zur Verfügung stellen. Durch einen entsprechenden Security-Adapter ist es auch möglich das Framework unabhängig von einer Webservice-Middleware in einem vorgeschalteten Gateway laufen zu lassen.

6.1.2 Offene Punkte

Die folgenden Punkte wurden bei dieser Arbeit nicht berücksichtigt und müssen noch bearbeitet werden, um die Funktionalität des Frameworks zu vervollständigen.

Autorisierung Da die Spezifikation WS-Authorization noch nicht existiert, wurde die Autorisierung in dieser Arbeit nicht betrachtet. Es gibt Ansätze, die Autorisierungsinformationen mit XACML[1] zu beschreiben (z.B. Hommel, 2005). Die Entwicklung eines Standards zeichnet sich aber noch nicht ab. Bis ein solcher Standard existiert, müssen die Anwendungen selbst die Autorisierung der Webservice-Zugriffe vornehmen.

Vergleichen von Policies In dieser Arbeit wurde davon ausgegangen, dass automatisch die vom Server vorgegebene Policy für einen Webservice verwendet wird. In der Praxis muss es möglich sein, dass der Benutzer ebenfalls vorgeben kann, ob und wie bestimmte Daten geschützt werden. Dazu müssen die Policies von Client und Server miteinander verglichen und ggfs zusammengeführt werden. Wie der Client und der Server die Policies austauschen können, wurde in dieser Arbeit ebenfalls nicht betrachtet.

Auswahl der Policies Wenn für ein Service (wie z.B. für den STS der Autovermietung) unterschiedliche Policies existieren, muss ein Benutzer die für ihn zutreffende Policy auswählen. Dazu besteht zum einen die Möglichkeit die Policies so zu priorisieren, dass der Benutzer durch Auswahl der höchstpriorisierten Policy aus den Policies, die zu seiner eigenen Policy passen, die für ihn zutreffende Policy ermitteln kann. Ist dies nicht möglich, besteht zusätzlich die Möglichkeit, den Webservice unter unterschiedlichen Adressen anzubieten, denen unterschiedliche Policies zugeordnet werden.

Verwenden von abgeleiteten Schlüsseln In WS-SecureConversation wird definiert, wie abgeleitete Schlüssel verwendet werden können, um z.B. Nachrichten mit dem Passwort des Benutzers zu verschlüsseln. In WS-SecurityPolicy wird jedoch nicht beschrieben, wie die notwendigen Parameter, die dazu notwendig sind, angegeben werden. Damit abgeleitete Schlüssel verwendet werden können, muss die Spezifikation entsprechend erweitert werden.

Sicherheitskontexte über mehrere Aufrufe Die unterschiedlichen Sicherheitskontexte beim Anwendungsfall „Preisvergleichportal“ wurden durch die unterschiedlichen asymmetrischen Bindings erreicht. Wenn eine Aktion mehrere Webservice-Aufrufe zwischen den gleichen Kommunikationspartnern erfordert, bietet es sich an, einen Sicherheitskontext aufzubauen, der einen oder mehrere (evtl auch abgeleitete) Schlüssel enthält, die

[1] eXtensible Access Control Markup Language

zur Verschlüsselung und Signierung der Daten verwendet werden. Die Spezifikation WS-SecureConversation definiert, wie ein solcher Sicherheitskontext aufgebaut und verteilt werden kann. Soll ein Sicherheitskontext für mehr als zwei Kommunikationspartner zugänglich sein, muss in den Policies definiert werden, an welche Kommunikationspartner die geheime Information (d.h. der Schlüssel oder das Datum, aus dem der Schlüssel abgeleitet wird) aus dem Kontext weitergegeben werden darf. Dies wird von der Spezifikation WS-SecurityPolicy derzeit nicht unterstützt. Sollen, wie in WS-SecureConversation empfohlen, abgeleitete Schlüssel verwendet werden, ergibt sich abermals das Problem, wie die benötigten Parameter in den Policies angegeben werden können.

6.2 Ausblick

Die vorhandenen Frameworks für WS-Security umfassen nur einen kleinen Teil denkbarer Sicherheitsprotokolle und arbeiten größtenteils mit proprietären Erweiterungen. Damit das Einbinden von Webservices anderer Organisationen in einem globalen Szenario funktioniert, müssen die Frameworks in der Lage sein, beliebige Sicherheitsprotokolle auszuführen, die mit den WS-Security-Spezifikationen definiert werden können. Darüber hinaus dürfen sich die proprietären Erweiterungen der Frameworks nur auf Implementierungsmaßnahmen beschränken, damit die Interoperabilität zwischen den Frameworks gewährleistet werden kann. Dies ist bei den bestehenden Frameworks nicht gegeben.

Da das Entwerfen eines Sicherheitsprotokolls keine triviale Aufgabe ist und bei sicherheitskritischen Anwendungen gewährleistet sein muss, dass die definierten Sicherheitsmaßnahmen greifen, wird ein Pattern-Katalog für Sicherheitsprotokolle benötigt, damit die Sicherheit der Anwendungen durch bewährte Mechanismen garantiert werden kann. Dies widerspricht nicht der Forderung nach umfassenden Frameworks, da es dennoch möglich sein muss, neue Mechanismen zu entwickeln, ohne dass eine Änderung des Frameworks oder Eingriffe in die Anwendungslogik notwendig ist. Damit auch bei der Implementierung der Protokolle gewährleistet ist, dass diese korrekt umgesetzt werden, wird zusätzlich eine entsprechende Tool-Unterstützung beim Erstellen der Policies für die Webservices benötigt.

Diese Arbeit hat gezeigt, dass es möglich ist, mit den bestehenden Spezifikationen ein Framework zu entwerfen, das in der Lage ist, die wesentlichen Sicherheitsziele wie Vertraulichkeit und Integrität von Anwendungen zu gewährleisten. Beim Implementieren des entworfenen Sicherheitsprotokolls hat sich jedoch gezeigt, dass die WS-*-Spezifikationen noch Lücken aufweisen. Darüber hinaus sind wichtige Bereiche, wie z.B. die Autorisierung noch nicht durch die WS-*-Spezifikationen abgedeckt. Insbesondere die Spezifikation WS-Authorization wird dringend benötigt, damit die Security-Frameworks in der Lage sind sämtliche Sicherheitsanforderungen abzudecken.

Damit das Einbinden von Diensten anderer Organisationen in einer globalen Umgebung im Hinblick auf die Sicherheit ohne Eingriffe in die eigenen Anwendungen möglich ist, muss die Weiterentwicklung der WS-Security-Spezifikation noch ein deutliches Stück vorangetrieben werden. Darüber hinaus müssen die eingesetzten Frameworks den Anforderungen genügen, die in dieser Arbeit diskutiert wurden. Bis Anwendungen, wie die Autovermietung aus dem hier verwendeten Beispielszenario, ohne individuelle Entwicklung für die Sicherheitsmaßnahmen erstellt werden können, werden wahrscheinlich noch ein paar Jahre vergehen.

Glossar

Apache Software Foundation Apache ist eine ehrenamtliche Organisation, die die Entwicklung von Open-Source-Softwareprojekten unterstützt.

API siehe **Application Programming Interface**

Application Programming Interface (API) Ein API ist eine Programmbibliothek, die ihre Funktionalität über eine beschriebene Schnittstelle für andere Anwendungen zur Verfügung stellt.

Axis 2 Axis 2 ist eine Webservice-Middleware für Java, die von Apache als Open-Source-Projekt angeboten wird.

CA siehe **Certification Authority**

Certification Authority (CA) Eine CA stellt Zertifikate aus und signiert diese. Die CA garantiert damit die Identität des Zertifikatbesitzers.

Demilitarisierte Zone Als demilitariserte Zone wird der Bereich eines Unternehmensnetzwerks bezeichnet, der vor dem Zugriff von außen z.B. durch Firewalls geschützt wird.

Denial-of-Service-Angriff Bei einem Denial-of-Service-Angriff wird ein Server innerhalb kurzer Zeit mit so vielen Anfragen überhäuft, dass er nicht mehr reagieren kann.

Document Object Model (DOM) DOM ist ein **API** für den Zugriff auf **XML**-Dokumente und wurde vom **W3C** spezifiziert.

DOM siehe **Document Object Model**

DoS siehe **Denial-of-Service-Angriff**

eXtensible Access Control Markup Language (XACML) XACML ist eine Sprache für die Darstellung und Verarbeitung von Autorisierungspolicies, die auf XML basiert.

eXtensible Markup Language (XML) XML ist eine Sprache, mit der hierarchisch strukturierte Daten dargestellt werden können. Zur Definition der Datentypen können die Elemente der Sprache beliebig erweitert werden.

Föderierte Entitäten siehe **Federated Identity Management**

Federated Identity Management (FIM) FIM beschäftigt sich mit der Verteilung von Identitäten in organisationsübergreifenden Anwendungsszenarios.

FIM siehe **Federated Identity Management**

HTTP siehe **Hypertext Transfer Protocol**

Hypertext Transfer Protocol (HTTP) HTTP ist ein einfaches Protokoll zum Übertragen von Texten, wir z.B. Internetseiten oder XML-Daten. Es kann mit wenig Aufwand auf beliebigen Hardwaresystemen implementiert werden und ist dadurch sehr gut für die Verwendung in heterogenen IT-Landschaften geeignet.

Internet Policy Registration Authority (IPRA) Als IPRA wird die **CA** in der obersten Hierarchiestufe einer **PKI** bezeichnet.

IPRA siehe **Internet Police Registration Authority**

Kerberos Kerberos ist ein Authentifizierungsprotokoll für offene und unsichere Netze (wie z.B. das Internet). Zur Authentifizierung werden Passwörter verwendet, die nicht über das Netz übertragen werden, sondern zum Verschlüsseln der ausgetauschten Daten verwendet werden.

LDAP siehe **Lightweigth Directory Access Protocol**

Lightweigth Directory Access Protocol (LDAP) LDAP ist ein Protokoll, das die Abfrage und Änderung von Daten eines Verzeichnisdienstes ermöglicht.

OASIS siehe **Organization for the Advancement of Structured Information Standards**

Organization for the Advancement of Structured Information Standards (OASIS) OASIS ist eine gemeinnützige Organisation, deren Aufgabe die Entwicklung und Standardisierung von Spezifikationen im Bereich E-Business ist.

PCA Siehe **Policy Certification Authority**

PKI siehe **Public Key Infrastructure**

Policy Certification Authority (PCA) Eine PCA berechtigt **CA**s in einer **PKI** zum Ausstellen von Zertifikaten.

Public Key Infrastructure (PKI) Eine PKI besteht aus einer Hierarchie von **CA**s, die Zertifikate ausstellen. Die **CA** in der obersten Hierarchie-Stufe ist die **IPRA**. Die **CA**s auf der zweiten Hierarchie-Stufe sind die **PCA**s. Ein Zertifikat von einer unbekannten **CA** kann in einer PKI auch als gültig eingestuft werden, wenn ein Pfad zu einer vertrauenswürdigen **CA** in der Hierarchie existiert.

Remote Procedure Call (RPC) RPC ermöglicht Funktionsaufrufe, die über ein Netzwerk auf entfernten Rechnern durchgeführt werden.

RPC siehe **Remote Procedure Call**

SAML siehe **Security Assertion Markup Language**

Security Assertion Markup Language (SAML) SAML ist eine Sprache zum beschreiben von Sicherheitsbestätigungen, die auf **XML** basiert. SAML definiert unter anderem Protokolle für ein verteiltes Single-Sign-On und verteilte Autorisierungsdienste.

Service Oriented Architecture SOA ist ein aktueller Architekturansatz, bei dem die Anwendungen aus einer Komposition von einzelnen Diensten bestehen. Die Dienste bieten in sich geschlossene Funktionen auf Ebene der Geschäftslogik an und können somit leicht wiederverwendet werden.

Simple Object Access Protocol (SOAP) SOAP ist ein Netzwerkprotokoll zum Übertragen von Daten und zum Durchführen von **RPC**s. SOAP basiert auf Standardtechniken wie **XML** und **HTTP**.

Smart-Card Smart-Cards sind Chipkarten, auf denen Daten gespeichert werden können. Darüber hinaus haben Smart-Cards eine eigene CPU und ein eigenes Betriebssystem, so dass sie Anwendungen mit individuellen Schnittstellen zur Verfügung stellen können.

SOA siehe **Service Oriented Architecture**

SOAP siehe **Simple Object Access Protocol**

UDDI siehe **Universal Description, Discovery and Integration**

Universal Description, Discovery and Integration (UDDI) UDDI ist ein universeller Verzeichnisdienst für Webservices.

Virtual Private Network (VPN) Ein VPN bietet die Möglichkeit einen sicheren Kommunikationskanal über ein unsicheres Netz aufzubauen, in dem die Daten verschlüsselt übertragen werden. Die kommunizierenden Anwendungen müssen sich dabei nicht

um die Verschlüsselung kümmern, da diese von einem so genannten VPN-Client sichergestellt wird.

VPN siehe **Virtual Private Network**

W3C siehe **World Wide Web Consortium**

Web Service Description Language (WSDL) WSDL definiert eine universelle Beschreibungssprache für Webservices.

Web Services Interoperability Organisation (WS-I) WS-I ist eine unabhängige Organisation, deren Ziel die Erstellung von Protokollen ist, die einen interoperablen Datenaustausch zwischen Webservices ermöglichen.

World Wide Web Consortium (W3C) Das W3C ist ein internationales Konsortium, in dem Mitgliedsorganisationen, ein fest angestelltes Team, und die Öffentlichkeit gemeinsam daran arbeiten, Web-Standards zu entwickeln. Ziel des W3C ist es, dem Internet dadurch seine vollen Möglichkeiten zu erschließen, dass Protokolle und Richtlinien entwickelt werden, die ein langfristiges Wachstum des Web sichern.

WS-I siehe **Web Services Interoperability Organisation**

WSDL siehe **Web Service Description Language**

X.509-Zertifikat X.509 ist heute der gängige Standard für digitale Zertifikate.

XACML siehe **eXtensible Access Control Markup Language**

XKMS siehe **XML Key Management Specification**

XML siehe **eXtensible Markup Language**

XML Key Management Specification (XKMS) XKMS ist eine Sprache zur Verteilung und Registrierung von öffentlichen Schlüsseln. XKMS basiert auf **XML** und ist für den Einsatz zusammen mit XML-Signature gedacht.

Literaturverzeichnis

[JSR173 2003] *Streaming API For XML JSR-173 Specification v1.0*, 2003

[Apache 2007a] APACHE: *Apache Axis 2 Homepage*. 2007. – URL http://ws.apache.org/axis2/

[Apache 2007b] APACHE: *Apache Neethi Homepage*. 2007. – URL http://ws.apache.org/commons/neethi/

[Apache 2007c] APACHE: *Apache WSS4J Homepage*. 2007. – URL http://ws.apache.org/wss4j/

[Apache 2007d] APACHE: *Apache XML Security Homepage*. 2007. – URL http://xml.apache.org/security/index.html

[Bajaj u. a. 2006] BAJAJ, Siddharth ; BOX, Don ; CHAPPELL, Dave ; CURBERA, Francisco ; DANIELS, Glen ; HALLAM-BAKER, Phillip ; HONDO, Maryann ; KALER, Chris ; LONGWORTHY, Dave ; MALHOTRA, Ashok ; NADALIN, Anthony ; NAGARATNAM, Nataraj ; NOTTINGHAM, Mark ; PRAFULLCHANDRA, Hemma ; RIEGEN, Claus von ; SCHLIMMER, Jeffrey ; SHARP, Chris ; SHEWCHUK, John: Web Services Policy Framework 1.2 (WS-Policy). (2006)

[Ballinger u. a. 2006] BALLINGER, Keith ; EHNEBUSKE, David ; FERRIS, Christopher ; GUDGIN, Martin ; LIU, Canyang K. ; NOTTINGHAM, Mark ; YENDLURI, Prasad: Basic Profile. (2006). – URL http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[Bieberstein u. a. 2006] BIEBERSTEIN, Norbert ; BOSE, Sanjay ; FIAMMANTE, Marc ; JONES, Keith ; SHAH, Rawn ; FOO, Linda (Hrsg.): *Service Oriented Architecture Compass*. IBM Press, 2006

[Bien 2003] BIEN, Adam: *J2EE Patterns*. Addison Wesley, 2003

[Booth u. a. 2004] BOOTH, David ; HAAS, Hugo ; MCCABE, Francis ; NEWCOMER, Eric ; CHAMPION, Michael ; FERRIS, Chris ; ORCHARD, David: Web Services Architecture. (2004)

[BSI 2006] BSI: *IT-Grundschutz-Kataloge*. Bundesamt für Sicherheit in der Informationstechnologie, 2006. – URL http://www.bsi.de/gshb

[Burrows u. a. 1990] BURROWS, Michael ; ABADI, Martín ; NEEDHAM, Roger: *A Logik of Authentication*. 1990

[Eastlake und Reagle 2002] EASTLAKE, Donald ; REAGLE, Joseph: XML Encryption Syntax and Processing. (2002)

[Eastlake u. a. 2002] EASTLAKE, Donald ; REAGLE, Joseph ; SOLO, David: XML-Signature Syntax and Processing. (2002)

[Eckert 2006] ECKERT, Claudia ; ROTH, Margit (Hrsg.): *IT-Sicherheit*. Oldenbourg, 2006

[Frotscher u. a. 2007] FROTSCHER, Thilo ; TEUFEL, Marc ; WANG, Dapeng: *Java Web Services mit Apache Axis 2*. entwickler.press, 2007

[Gamma u. a. 1995] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995

[Gordon u. a. 2006] GORDON, Lawrence A. ; LOEB, Martin P. ; LUCYSHYN, William ; RICHARDSON, Robert: CSI/FBI Computer crime and security survey. (2006)

[Hohpe und Woolf 2004] HOHPE, Gregor ; WOOLF, Bobby: *Enterprise Integration Patterns*. Addison Wesley, 2004

[Hommel und Reiser 2005] HOMMEL, Wolfang ; REISER, Helmut: Federated Identity Management: Shortcomings of existing Standards. (2005)

[Hommel 2005] HOMMEL, Wolfgang: Using XACML for Privacy Control in SAML-Based Identity Federations. (2005)

[IBM 2007] IBM: *WebSphere Application Server*. 2007. – URL http://www-306.ibm.com/software/webservers/appserv/was/index.html?S_TACT=105AD02W&S_CMP=campaign

[International Telecommunication Union 2005] INTERNATIONAL TELECOMMUNICATION UNION: *ITU-T Recommendation X.509*. 2005

[Kahlbrandt 2001] KAHLBRANDT, Bernd: *Software-Engineering mit der Unified Modeling Language*. Springer, 2001

[Lawrence und Kaler 2006a] LAWRENCE, Kelvin ; KALER, Chris: Web Services Security: Kerberos Token Profile 1.1. (2006)

[Lawrence und Kaler 2006b] LAWRENCE, Kelvin ; KALER, Chris: Web Services Security: SAML Token Profile 1.1. (2006)

[Lawrence und Kaler 2006c] LAWRENCE, Kelvin ; KALER, Chris: Web Services Security: SOAP Message Security 1.1. (2006)

[Lawrence und Kaler 2006d] LAWRENCE, Kelvin ; KALER, Chris: Web Services Security: Username Token Profile 1.1. (2006)

[Lawrence und Kaler 2006e] LAWRENCE, Kelvin ; KALER, Chris: Web Services Security: X.509 Certificate Token Profile 1.1. (2006)

[Lawrence und Kaler 2006f] LAWRENCE, Kelvin ; KALER, Chris: WS-SecureConversation 1.3. (2006)

[Lawrence und Kaler 2006g] LAWRENCE, Kelvin ; KALER, Chris: WS-Trust 1.3. (2006)

[Lawrence und Kaler 2007] LAWRENCE, Kelvin ; KALER, Chris: WS-SecurityPolicy 1.2 (Editors Draft 02). (2007)

[Lockhart u. a. 2006] LOCKHART, Hal ; ANDERSEN, Steve ; BOHREN, Jeff ; SVERDLOV, Yakov ; HONDO, Maryann ; MARUYAMA, Hiroshi ; NADALIN, Anthony ; NAGARATNAM, Nataraj ; BOUBEZ, Toufic ; MORRISON, K S. ; KALER, Chris ; NANDA, Arun ; SCHMIDT, Don ; WALTERS, Doug ; WILSON, Hervey ; BURCH, Lloyd ; EARL, Doug ; BAJA, Siddharth ; PRAFULLCHANDRA, Hemma: Web Services Federation Language 1.1 (WS-Federation). (2006)

[Microsoft 2007] MICROSOFT: *Web Service Enhancements (WSE)*. 2007. – URL http://msdn2.microsoft.com/en-us/webservices/aa740663.aspx

[Naedele 2003] NAEDELE, Martin: Standards for XML and Web Services Security. (2003)

[Steel u. a. 2006] STEEL, Christopher ; NAGAPPAN, Ramesh ; LAI, Ray: *Core Security Patterns*. Prentice Hall, 2006

[Viega und Epstein 2006] VIEGA, John ; EPSTEIN, Jeremy: Why Applying Standards to Web Services Is Not Enaugh. (2006), Nr. 1540-7993/06

[Weerawarana u. a. 2005] WEERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture*. Prentice Hall, 2005

[Wiehler 2004] WIEHLER, Gerhard: *Mobility, Security and Web Services*. Publicis Corporate Publishing, Erlangen, 2004

[Woods 2003] WOODS, Dan: *Enterprise Service Architecture*. O'Reilly, 2003

[Zedlitz 2006] ZEDLITZ, Jesper: *Webservice-Firewall*. VDM Verlag Dr. Müller, 2006

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne des §22(4) der Prüfungsordnung des Masterstudiengangs Informatik an der Hochschule für Angewandte Wissenschaften Hamburg vom 22. November 2001, geändert am 7. Dezember 2004, ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. April 2007	
Ort, Datum	Unterschrift