TECHNISCHE UNIVERSITÄT DARMSTADT

Fraunhofer IGD Institut Graphische Datenverarbeitung

Diplomarbeit

Sicherheitsmechanismen in Service-orientierten Architekturen

von

Dimitri Idessis
Matrikelnummer 1087019

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Graphisch-Interaktive Systeme
Fraunhoferstraße 5
64283 Darmstadt

Betreuer: Dipl.-Inform. Jan Peters

Prüfer: Prof. Dr.-Ing. J.L. Encarnação

Aufgabenstellung für die Diplomarbeit des
Herrn cand.-Inform. Dimitri Idessis
Matrikel-Nr. 1087019

Thema: “Sicherheitsmechanismen in Service-orientierten Architekturen”

In der Abteilung für Sicherheitstechnologie am Fraunhofer-Institut für Graphische Datenverarbeitung IGD wird an einer Java-basierten Sicherheitsplattform für Softwarekomponenten geforscht, die in ubiquitären Umgebungen Verwendung finden soll. Die im Rahmen des Projekts SicAri entwickelte und eingesetzte Plattform besteht aus drei Komponenten. Die Kommando-Shell wird vom Plattform-Administrator genutzt, um die Plattform zu starten sowie zur Laufzeit flexibel zu konfigurieren, zu erweitern und zu beobachten. Das sogenannten Environment dient als Dienstumgebung für das lokale Service-Management. Ein Sicherheitskontext setzt bei der Interaktion von Nutzern, Anwendungen und Diensten Sicherheitspolitiken durch. Die Funktionalität einer speziellen Plattforminstanz zeichnet sich durch die lokal gestarteten Dienste und Anwendungen aus.

Damit verschiedene Plattforminstanzen in einer verteilten Infrastruktur miteinander kommunizieren können, lassen sich lokal geladene Java-Komponenten während der Laufzeit automatisiert und transparent für den Programmierer in Web Services transformieren, die über einen externen Verzeichnisserver registriert werden und dann innerhalb der Infrastruktur zur Verfügung stehen. Das Framework zur Integration von Web Services basiert auf Apache AXIS und UDDI4J.

Ziel der Diplomarbeit ist der Entwurf und die Integration von Sicherheitsmechanismen, die bei der Web Service basierten Interaktion die Authentisierung von registrierten Nutzern und dadurch (abhängig von der definierten Sicherheitspolitik) die Autorisierung des Dienstzugriffs ermöglichen. Zudem soll es einem gegenüber der lokalen Plattforminstanz authentisierten Benutzer möglich sein, einen entfernten Dienst ohne wiederholte, aktive Authentisierung direkt anzusprechen (Single-Sign-On). Zur Lösung der Aufgabe sollen existierende Techniken aus dem Bereich Web Service Security (WS-Security) als auch generelle Sicherheitskonzepte aus dem Bereich der Service-orientierten Architekturen in Betracht gezogen und ggf. kombiniert und erweitert werden.

Die Diplomarbeit soll folgende Aspekte beinhalten:

- Literaturrecherche und Related Work
- Vergleich existierender Middleware-Lösungen
- Bedrohungs- und Anforderungsanalyse
- Entwurf und Integration in die Plattform

Die prototypisch implementierten Sicherheitsmechanismen sollen abschließend evaluiert werden.

Darmstadt, den 30.10.2005

Betreuer: Dipl.-Inform. Jan Peters **Prüfer:** Prof. Dr.-Ing. J. Encarnação

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Juni 2006 Dimitri Idessis

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Beschreibung der Grundlagen

1.1 Einführung

Dieses Kapitel soll die grundlegenden Mechanismen und Techniken einführen, die in den folgenden Kapiteln verwendet werden. Zuerst werden Aufgabenstellung, Typographie und ein Überblick über die Diplomarbeit beschrieben. Daraufhin folgt in Abschnitt 1.2 eine Beschreibung einer Service-orientierten Architektur. Nach Abschnitt 1.3, in dem die Eigenschaften eines Webservices erklärt werden, folgt Abschnitt 1.4, welcher in die Sicherheitsanforderungen eines verteilten IT-Systems und in die Realisationsansätze einführt. Die weiteren drei Abschnitte beinhalten die wichtigsten Konzepte und Protokolle zur Authentifizierung. Abschnitt 1.5 stellt Verfahren vor, die auf dem Vorweisen eines spezifischen Wissens basieren. Abschnitt 1.6 beschäftigt sich mit Authentifikationsprotokollen in Client-Server-Architekturen. In verteilten Systemen besteht der Wunsch nach Single-Sign-On (SSO), d.h. dass einem Benutzer es ermöglicht werden soll, sich ein Mal im verteilten System anzumelden und danach ohne weitere Authentifikation alle registrierten Dienste zu nutzen. In der Praxis eingesetzte Protokolle werden in Abschnitt 1.7 beschrieben. In Abschnitt 1.8 werden die Mechanismen und Verfahren zur Rechteverwaltung und Zugriffskontrolle vorgestellt. In verteilten Systemen kommt es oft vor, dass Dienste im Namen des Nutzers agieren und für diesen bestimmte Aufgaben erledigen. Um aber im Namen eines anderen agieren zu können, werden dessen Rechte benötigt. Auf dieses Problem wird in Abschnitt 1.9 eingegangen.

1.1.1 Aufgabenstellung und Motivation

In dieser Diplomarbeit geht es um Integration von Sicherheitsmechanismen in die Middleware-Plattform *SicAri* [1]. Die SicAri-Plattform basiert auf Javatechnologie und soll ihre Hauptanwendung in der sicheren, ubiquitären Internetnutzung finden und den Nutzern verschiedene Webservices anbieten. Bei der Webservice-basierten Interaktion mit der Plattform sollen die registrierten Nutzer erst authentifiziert werden, d.h. dass die Nutzer sich gegenüber der Plattform identifizieren müssen, in dem sie ein vorab mit der Plattform vereinbartes Geheimnis, wie z.B. Name und Passwort nachweisen. Dann soll je nach Autorisation des authentifizierten Nutzers, d.h. dem Nutzer erteilten Rechte, der Dienstzugriff erfolgen. Zudem soll es den Nutzern nach einer erfolgreichen Authentifizierung gegenüber einer lokalen Plattforminstanz möglich sein, alle ihnen

[1] www.sicari.de

erlaubten Dienste nutzen zu können, ohne sich dabei erneut authentifizieren zu müssen und das unabhängig davon, ob diese Dienste lokal oder auf entfernten Rechnern vorhanden sind (SSO). Zur Lösung der Aufgabe werden existierende Techniken aus dem Bereich *Web Service Security (WS-Security)* als auch generelle Sicherheitskonzepte aus dem Bereich der Service-orientierten Architekturen in Betracht gezogen und ausgewertet. Anschließend wird eine an die Plattform angepasste Lösung entworfen und in die Plattform integriert.

1.1.2 Typographie

Klassennamen, Datentypen sowie Quelltextauszüge sind in `Schreibmaschinenschrift` gesetzt. Um die für einen Absatz relevanten Schlüsselwörter oder Satzteile, wird ein serifenloser Schriftsatz verwendet oder sie werden **fett** dargestellt. Neu eingeführte und englische Begriffe sind im Text an ihrer *kursiven* Darstellung zu erkennen. Abkürzungen werden bei ihrem erstmaligem Auftreten *kursiv* dargestellt, explizit ausgeschrieben und dann anschließend ohne besondere Hervorhebung verwendet.

1.1.3 Überblick über den Aufbau der Diplomarbeit

Diese Diplomarbeit besteht aus vier Kapiteln. Im ersten Kapitel werden die grundlegenden Mechanismen und Techniken vorgestellt. Im zweiten Kapitel wird der aktuelle Stand der SicAri-Plattform-Implementierung beschrieben. Im dritten Kapitel geht es um die Erweiterung der SicAri-Plattform. Im letzten Kapitel wird ein Ausblick auf weitere Aufgaben und mögliche Erweiterungen gemacht.

1.2 Service-orientierte Architekturen

In der IT-Welt wurden und werden immer wieder neue Entwicklungsparadigmen entwickelt, um den Softwareentwicklungsprozess effektiver zu gestalten und gleichzeitig die Softwarequalität zu erhöhen. So hat sich z.B. Objektorientierung in den meisten Gebieten der Softwareentwicklung gegenüber der prozeduraler Programmierung durchgesetzt. Im Bereich der Applikationsintegration wurde bei der Suche nach neuen Lösungsansätzen die so genannte *Service-orientierte Architektur (SOA)* ins Leben gerufen [51, 11, 42].

Der allgemeine Begriff Architektur eines Softwaresystems beschreibt grob die Struktur des Systems, nämlich die Komponente, aus denen das System besteht und die Interaktion dieser Komponente miteinander. Eine Service-orientierte Architektur ist eine spezielle Art einer Softwarearchitektur, die den Aspekt der Service-Orientierung besonders hervorhebt. Eine Service-orientierte Architektur ist ein Managementkonzept und setzt erst in zweiter Linie ein Konzept der Systemarchitektur voraus. Das Managementkonzept strebt eine an den gewünschten Geschäftsprozessen ausgerichtete IT-Infrastruktur an, die schnell auf veränderte Anforderungen im Geschäftsumfeld reagieren kann. Bei der Anpassung des Systems an neue Anforderungen spielen Modifikation des Systems und Integration neuer Applikationen in das System eine wesentliche Rolle.

Das Konzept der Systemarchitektur sieht die Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services vor. Ein Service sei in diesem Kontext als eine Funktionalität

definiert, die über eine standardisierte Schnittstelle in Anspruch genommen werden kann. Ein Synonym also für die lange bekannte Software-Komponente. Um eine einfache Applikationsintegration zu ermöglichen, wird eine Serviceebene über der eigentlichen Softwareinfrastruktur eingeführt. In dieser Serviceebene werden bestimmte Funktionalitäten der vorhandenen Software als Services zur Verfügung gestellt, sodass die Funktionalität über Netzwerkkommunikation aufgerufen werden kann.

Konkret von SOA gesprochen wird erst seit der Einführung von Jini durch Sun Microsystems Ende der 90er Jahre, da Jini nicht nur den Servicebegriff betont, sondern auch ein Konzept für Auffinden eines Services (*service discovery*) und Referenzübergabe vom Verzeichnisdienst an den Nutzer auf passenden Service-Provider (*service leasing*) mitbringt. Bei Jini handelt es sich um eine Technologie, die ein dynamisches Auffinden und Benutzen von Netzwerkdiensten erlaubt. Populär wurde der Begriff Service-orientierte Architektur erst später, nachdem Webservices, die in Abschnitt 1.3 beschrieben werden, sich schlagartig in der IT-Welt verbreitet haben. Der Grund für die schnelle Verbreitung liegt darin, dass Webservices eine Realisierung einer Service-orientierten Architektur erlauben.

Was ist nun ein Service? Ein Service ist vergleichbar mit einem Objekt aus der objektorientierten Welt. Wie ein Objekt besitzt jeder Service eine eindeutige Identität (*identity (ID)*). Über diese ID wird auch die Adresse des Services (*service endpoint*), über die dieser Service erreichbar ist, repräsentiert. Ein Service verfügt wie ein Objekt über eine klar definierte Schnittstelle, über die er aufgerufen werden kann. Als die kleinste Einheit der Service-orientierten Architektur ist ein Service im Gegensatz zu einem Objekt komplexer. Die interne Struktur des Services ist gut gekapselt und nach außen nicht sichtbar. Was ein Service nach außen bietet, ist nur die Serviceschnittstelle. Ein wichtiger Aspekt der Service-orientierten Architektur ist daher, dass die Servicebeschreibung und die Serviceentwicklung strikt getrennt werden.

Servicebeschreibung: Die Servicebeschreibung besteht im Wesentlichen aus der Beschreibung der Serviceschnittstelle, die in einer Schnittstellenbeschreibungssprache erfolgt. Diese Schnittstellenbeschreibung definiert, welche Operationen ein Service anbietet und wie sie aufgerufen werden können. Ein Service-Nutzer oder Service-Client betrachtet einen Service als einen Endpunkt, der ein bestimmtes Anfrageformat erwartet. Es ist irrelevant für den Service-Nutzer, wie der Service die Anfrage verarbeitet bzw. auf welchen Plattformen und Architekturen der Service aufbaut. Für ihn ist entscheidend, dass er eine Antwort in einem vereinbarten Format für seine Anfrage erwarten kann. Die Schnittstellenbeschreibung kann durch weitere Beschreibungen für Kategorisierung oder Metadaten ergänzt werden.

Serviceimplementierung: Die Serviceimplementierung bietet eine schnittstellenkonforme Umsetzung der Servicebeschreibung. Hinter der Serviceimplementierung kann ein modernes, objektorientiertes (z.B. in Java entwickeltes) Softwaresystem oder ein 30 Jahre altes Programm stehen.

Eine weitere Eigenschaft, die Service-orientierte Architekturen kennzeichnet, ist die Möglichkeit, Services zu publizieren und dynamisch zu lokalisieren. Diese Eigenschaft wird ermöglicht, indem alle wiederverwendbaren Services in einem zentralen Verzeichnis (*Registry*) registriert werden. Dabei werden die Beschreibung und andere Metainformationen des Services in einer Datenbank innerhalb des Verzeichnisses abgelegt und von diesem verwaltet. Über definierte Schnittstellen können die Services in diesem Verzeichnis publiziert und lokalisiert werden. Über das Serviceverzeichnis können vorhandene Services leicht gefunden werden.

Zusammengefasst soll die Service-orientierte Architektur bei der Applikationsintegration folgende Fragen beantworten:

Was: Welche Funktionalität beinhaltet ein Service und wie ist die Schnittstelle dieses Services definiert? Eine Antwort auf diese Frage liefert die Servicebeschreibung, die genaue Auskunft darüber gibt, wie die Schnittstelle eines Services aussieht und wie der Service aufgerufen werden kann. Eine von der plattform-, programmiersprachen- und protokollunabhängige Schnittstellenbeschreibungssprache ist an dieser Stelle erwünscht, um die Interoperabilität zu erhöhen.

Wie: Wie kann ein Service aufgerufen werden und welches Anfrageformat wird vom Service erwartet? Hier muss ein Kommunikationsprotokoll ausgewählt werden, das ebenfalls aus Interoperabilitätsgründen möglichst unabhängig von Programmiersprachen und Plattformen sein sollte.

Wo: Wo kann ein wiederverwendbarer Service gefunden und adressiert werden? Hier setzt das Konzept des Serviceverzeichnisses an, das eine Suche nach publizierten Services erlaubt.

1.2.1 Interaktion in der Service-orientierten Architektur

Die Service-orientierte Architektur basiert auf der Interaktion zwischen drei Rollen: Service-Provider, Service-Nutzer und Service-Broker.

Service-Provider: Ein Service-Provider implementiert einen Service und stellt diesen Service im Netzwerk zur Verfügung. Dieser Service besitzt eine klar definierte Schnittstelle und ist selbst im Netzwerk adressierbar. Optional kann ein Service-Provider die Servicebeschreibung samt Metadaten bei einem oder mehreren Service-Brokern anmelden, damit dieser Service dynamisch von den potenziellen Service-Nutzern gefunden werden kann. Der Begriff „Provider“ ist teilweise doppeldeutig, weil er sich sowohl auf die Organisation beziehen kann, die den Service bereitstellt, als auch auf das Programm, das den Service implementiert. Die genaue Bedeutung ist immer dem Kontext zu entnehmen.

Service-Nutzer: Ein Service-Nutzer ist in der Lage einen Service in Anspruch zu nehmen. Dabei kann es sich um eine Person handeln, die z.B. über einen Browser einen Webservice aufruft, oder um eine Softwareanwendung. Falls die Schnittstelle und die Adresse des Services nicht schon bekannt sind, muss er zunächst durch eine entsprechende Anfrage beim Service-Broker die notwendigen Informationen in Erfahrung bringen. Danach kann er den Service aufrufen, indem er eine Anfrage in einem geeigneten Format, wie in der Schnittstellenbeschreibungssprache beschrieben, an den Service schickt.

Service-Broker: Ein Service-Broker ist der Vermittler zwischen Provider und dem Nutzer. Er hat die Aufgabe, einerseits die vom Service-Provider publizierten Services entgegenzunehmen und in einem Verzeichnis abzulegen, und andererseits die gespeicherten Services über intelligente Suchmöglichkeiten den Service-Nutzern zur Verfügung zu stellen. Zur Erfüllung dieser Aufgabe wird meistens ein zentrales Verzeichnis für Service-Registrierung und Service-Discovery (Aufsuchen eines Services) benutzt. Der Service-Broker ist ein optionales Element in der Service-orientierten Architektur und kommt nur dann ins Spiel, wenn sich Provider und Nutzer nicht direkt kennen.

In dem typischen Szenario einer Service-orientierten Architektur stellt ein Service-Provider die Implementierung eines Services bereit und veröffentlicht die Beschreibung bzw. einen Verweis auf die Beschreibung für diesen Service bei einem Service-Broker bzw. in einem Verzeichnisdienst. Ein Service-Nutzer benutzt das Verzeichnisdienst, um einen Service zu finden, der ihn interessiert. Nachdem der Service-Nutzer die Servicebeschreibung vom Broker bekommen hat,

kann er sich mit dem Service-Provider verbinden und den gewünschten Service aufrufen. Dieses Paradigma wird als „find, bind and execute“ bezeichnet und ist in Abbildung 1.1 dargestellt.

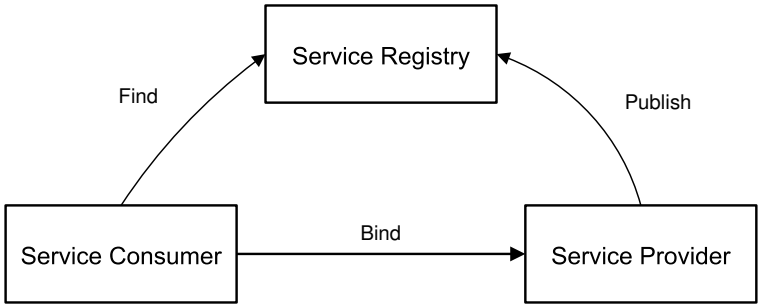

Abbildung 1.1: Die drei Rollen der Service-orientierten Architektur

1.2.2 Merkmale einer Service-orientierten Architektur

Folgende Merkmale zeichnen eine Service-orientierte Architektur aus.

1. Ein Service kann **dynamisch lokalisiert und gebunden** werden.

   Wenn ein Nutzer einen Service benötigt, kann er diesen Service anhand bestimmter Kriterien finden. Dabei stellt der Nutzer eine Abfrage an das Serviceverzeichnis und kann aus der Ergebnismenge den passenden Service aussuchen. Anhand der Servicebeschreibung, die der Nutzer ebenfalls von dem Serviceverzeichnis erhält, ist er in der Lage, sich mit diesem neu lokalisierten Service dynamisch zu verbinden und ihn aufzurufen. Die Verbindung wird deshalb als dynamisch bezeichnet, weil der Nutzer zur Zeit der Kompilierung keinerlei Information über den Service besitzen muss und die Serviceanfrage dynamisch aufbauen kann. Diese Unabhängigkeit zur Zeit der Kompilierung hat zur Folge, dass sich die Modifizierung des Services erleichtert. Zur Veranschaulichung dieses Vorgangs kann folgendes Beispiel betrachtet werden: Ein Nutzer möchte den günstigsten Call-by-Call-Service in Anspruch nehmen. Da er den momentan günstigsten Anbieter nicht kennt, kann er eine Abfrage nach allen Call-by-Call-Services an ein Serviceverzeichnis stellen. Von der Treffermenge sucht er den günstigsten Anbieter aus und bekommt dabei auch weitere Informationen, wie zum Beispiel die Adresse (Zugangsnummer) und die Schnittstelle dieses Services. Nun ist er in der Lage, diesen Service zu nutzen. Dadurch, dass der Nutzer bei dieser Vorgehensweise den Provider immer dynamisch aussucht und bindet, kann er sicher sein, dass er immer den günstigsten Service benutzt, auch wenn die Anbieter ihre Tarife ständig ändern.

2. Jeder Service wird als **ein eigenständiges Modul** gekapselt und bereitgestellt.

   Jeder Service unterstützt eine klar definierte Schnittstelle, deren Beschreibung veröffentlicht wird. Die interne Struktur sowie die Serviceimplementierung sind für den Nutzer nicht sichtbar. Durch die Unabhängigkeit der Services wird eine leichte Wiederverwendung ermöglicht.

3. Die Operationen im Serviceinterface sind meistens **grob granular**.

   Während verteilte Objekte aus den Vorstellungen der *distributed objects* oder *objectweb* [50] zu fein granular sind und zu viele Abhängigkeiten besitzen. Dies steht vor allem im Widerspruch zum Performance-Overhead, der bei jeder Netzwerkkommunikation entsteht. So wird meistens der Weg zu den Webservices gewählt, weil hier die feinen Objekte immer mehr zusammengefasst und abstrahiert werden, sodass aus den Objekten größere Komponenten und aus den Komponenten wiederum Services entstehen. Der Granularitätslevel eines Services scheint ein geeigneter Kompromiss zwischen Wiederverwendung und Performance zu sein.

4. Ein Basiskonzept von Service-orientierte Architekturen ist die **lose Kopplung** zwischen *Provider* und *Nutzer*.

   Der Begriff der Kopplung beschreibt in der Software-Architektur, wie eng die Komponenten einer Software miteinander verbunden sind.

   Bei *enger Kopplung* können die Komponenten auf vielen Annahmen aufbauen, wie z.B der ständigen Verfügbarkeit anderer Komponenten und darauf, dass sich Vereinbarungen über die Aufruf-Syntax von bestimmten Funktionen nicht ändert, oder darauf, dass Daten in einem ganz speziellen Format geliefert werden. Diese enge Kopplung hat den Vorteil, dass Komponente A sehr speziell auf Komponente B zugeschnitten werden kann und so optimale Leistungsfähigkeit bietet. Der Nachteil ist, dass jegliche Änderung von Komponente B (etwa bei Auffinden eines Fehlers) zu einer Änderung in Komponente A führen muss. Dies ist bei großen, verteilten Anwendungen sehr mühevoll bis nicht realisierbar. Allein die Annahme der ständigen Verfügbarkeit schließt enge Kopplung bei Anwendungen aus, die quer über die Welt verteilt und über das Internet verbunden sind.

   Deswegen setzt sich heute immer mehr der Ansatz der *losen Kopplung* durch. Hier werden möglichst wenige Annahmen gemacht, und die einzelnen Komponenten agieren so autonom wie möglich. Da der Nutzer einen Service nicht unbedingt im Voraus kennen muss, um sich mit ihm dynamisch zu verbinden, wird die Abhängigkeit zwischen *Provider* und Nutzer minimiert.

5. Ein Service ist **netzwerkadressierbar** und bietet **Ortstransparenz**.

   Services werden nicht an Nutzer verteilt und bei Nutzern lokal eingesetzt, sondern können direkt im Netzwerk aufgerufen werden. Die Servicebeschreibung enthält genügend Information über den Ort (z.B. über URL), den Transportmechanismus (z.B. HTTP oder SMTP) und das Payloadformat (z.B. SOAP oder IIOP), sodass der Nutzer anhand dieser Informationen den Service adressieren und aufrufen kann. Da der Service von den Nutzern dynamisch gebunden wird, kann er auch im Netzwerk ohne weiteres verschoben werden. Dadurch ist es z.B. möglich, durch Einrichten eines Lastverteilungsmechanismus die Skalierbarkeit und Verfügbarkeit eines Services zu erhöhen. Diese Änderung bleibt für die Nutzer transparent.

6. **Interoperabilität** wird in Service-orientierten Architekturen besonders hervorgehoben.

   Interoperabilität bezeichnet die Fähigkeit eines Services, von möglichst vielen unterschiedlichen Nutzern aufgerufen werden zu können. Wenn ein Service nur von Java-Clients aufgerufen werden soll, reicht es aus, diesen Service über RMI zugreifbar zu machen. Aber wenn auch Clients in anderen Programmiersprachen diesen Service nutzen

möchten, ist RMI bzw. *Java Remote Method Protocol (JRMP)* allein als Kommunikationsprotokoll nicht mehr ausreichend. Eine Standardtechnik um die Interoperabilität zu erhöhen, besteht darin, zusätzliche Adapter zur Verfügung zu stellen, wie es in Abbildung 1.2 dargestellt ist. Mit Hilfe dieser Adapter kann ein Nutzer einen Service nicht nur über ein anderes Protokoll (RMI, RPC, SOAP) aufrufen, sondern auch in einem anderen Format (XML oder serialisierte Java-Objekte). Bevorzugt werden natürlich Protokolle und Formate, die unabhängig von Programmiersprachen und Plattformen sind.

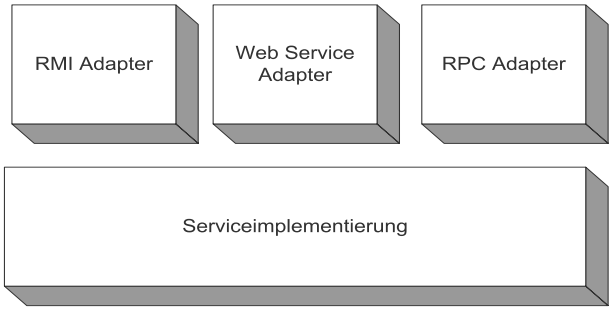

Abbildung 1.2: Serviceadapter

7. Mit **Servicekomposition** *(service assembly)* können neue Services aus vorhandenen Services aufgebaut werden.

   Das Serviceverzeichnis enthält einen Katalog von wiederverwendbaren Services, die Bausteine für neue, gröbere Services liefern. Durch Kombination solcher Bausteine können leicht und kostengünstig neue Services gewonnen werden.

8. Service-orientierte Architekturen mit einem gewissen Mehrwert schützen die Investition und **verkürzen den Entwicklungszyklus** neuer Anwendungen.

   Weil Service-orientierte Architekturen nicht das Ziel haben, ein System komplett neu zu erstellen, sondern eine zusätzliche Service-Ebene für Integration einführen, bleibt die vorhandene Software bestehen und kann weiter genutzt werden. Die Service-Ebene erhöht aber die Interoperabilität des Systems, sodass die vorhandene Software von mehr Clients genutzt werden kann. Die dadurch erzielte Zunahme des Wiederverwendungsgrads hat wiederum zur Folge, dass sich neue Applikationen schneller entwickeln lassen, was im Umkehrschluss die Entwicklungskosten senkt.

An diesen Merkmalen kann man erkennen, warum Service-orientierte Architekturen eine große Zukunft vor sich haben.

1.3 Webservice

Obwohl Service-orientierte Architekturen mit verschiedenen Technologien implementiert werden können, liefern Webservices einen idealen Satz von Standards, um Service-orientierte Architekturen umzusetzen. Dabei stützen sich Webservices auf drei zentrale Standards: *Simple Object Access Protocol (SOAP)* [29], *Web Service Description Language (WSDL)* [8] und U*niversal Description, Discovery and Description (UDDI)* [9].

Alle drei Standards basieren auf *Extensible Markup Language (XML)* und besitzen damit ein plattform-, programmiersprachen- und protokollunabhängiges Format, dass die Lesbarkeit verbessert und die Lesbarkeit von Webservices erhöht. Als Trägerprotokoll für den Nachrichtentransport kommt meistens HTTP zum Einsatz, das wegen seiner Verbreitung eine gute Voraussetzung für den Erfolg der Webservice-Technologie darstellt. Darüber hinaus wird durch den Einsatz von HTTP auch das Firewall-Problem vermieden [51, 11].

In einem Webservice-Szenario kann ein Service-Provider seinen Service in beliebiger Programmiersprache auf beliebiger Plattform erstellen. Zusätzlich kann der Service-Provider eine Beschreibung für seinen Service erstellen (oder von entsprechenden Tools erstellen lassen), und zwar in WSDL. Darin wird unter anderem festgelegt, welche Nachrichten als Anfragen (*requests*) bzw. Antworten (*responses*) erwartet werden und wo im Netzwerk der Service zur Verfügung steht. Dieser Service bzw. die zugehörige Servicebeschreibung kann später bei einer UDDI-Registry registriert werden. Wenn ein Nutzer einen derartigen Service in Anspruch nehmen möchte, muss er zuerst in Erfahrung bringen, wo der Service verfügbar ist. Dabei kann er sich an die UDDI-Registry wenden, um den passenden Service herauszufinden. Anhand der Servicebeschreibung, oder genauer gesagt dem WSDL-Dokument dieses Services, kann der Client feststellen, was dieser Service genau tut. Als Kommunikationsprotokoll für Webservices wird überwiegend SOAP eingesetzt, das im nächsten Abschnitt beschrieben wird.

1.3.1 XML

Extensible Markup Language (XML) [2] ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur, der vom *World Wide Web Consortium (W3C)* definiert wird. XML definiert dabei die Regeln für den Aufbau solcher Dokumente. Für einen konkrete XML-Anwendung müssen die Details der jeweiligen Dokumente spezifiziert werden. Dies betrifft insbesondere die Festlegung der Strukturelemente und ihre Anordnung innerhalb des Dokumentenbaums. Damit eine XML-Anwendung ein XML-Dokument interpretieren kann, muss dieses Dokument wohlgeformt und gültig sein.

Wohlgeformtheit: ein XML-Dokument ist wohlgeformt (*well formed*), falls es sämtliche Regeln für XML einhält. Beispielsweise muss für jedes öffnende Element (z. B. “<item>“) auf gleicher hierarchischer Ebene ein schließendes Element vorhanden sein (z. B. “</item>“).

Gültigkeit: soll XML für den Datenaustausch verwendet werden, ist es von Vorteil, wenn das Format mittels einer Grammatik (z. B. einer *Document Type Definition (DTD)* oder eines XML-Schemas) definiert ist. Ein XML-Dokument, welches wohlgeformt ist und ein durch eine Grammatik beschriebenes Format einhält, ist als gültig (*valid*) definiert.

Die Namen der Strukturelemente (XML-Elemente) für eine XML-Anwendung lassen sich frei wählen. Ein XML-Element kann ganz unterschiedliche Daten enthalten und beschreiben, meistens Text, aber auch Grafiken oder abstraktes Wissen. Ein Grundgedanke hinter XML ist es, Daten und ihre Repräsentation zu trennen, um Daten beispielsweise einmal als Tabelle und einmal als Grafik auszugeben, aber für beide Arten der Auswertung die gleiche Datenbasis im XML-Format zu nutzen. XML verwendet *tags* (durch ’<’ und ’>’ geklammerte Wörter) und Attribute (der Form name=“value“). Die Bedeutung der *tag* und der Attribute sind nicht festgelegt und werden nur zur Abgrenzung von Daten benutzt. Die Interpretation der Daten wird allein der Anwendung überlassen, die sie verarbeitet.

[2] http://www.w3.org/XML/

XML erlaubt es einem Anwender, ein neues Dokumentenformat zu definieren, indem man andere Formate kombiniert oder wiederbenutzt. Wenn jedoch zwei Formate völlig unabhängig voneinander entwickelt worden sind, können sie Elemente und Attribute enthalten, die in beiden Formaten mit dem gleichen Namen vorkommen. Wenn man diese dann kombinieren will, muss man entsprechend vorsichtig sein (z.B. meint “<p>“ jetzt “Absatz“ aus dem einen Format, oder aber “Person“ aus dem anderen Format?). Um bei der Kombination von Formaten Namenskollisionen zu vermeiden, stellt XML den Namensraummechanismus zur Verfügung. Für jeden benutzten Namensraum wird im XML-Dokument jeweils ein Präfix deklariert. Dieses wird vor jedes Element aus diesem Namensraum getrennt durch einen "Doppelpunkt"geschrieben wird. (z.B. `<wsse:Signature ... />`). Zusammengefasst könnte es wie in Abbildung 1.3 aussehen.

```
<Envelope xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext>"
  <wsse:Signature>
     ...
  </wsse:Signature>
</Envelope>
```

Abbildung 1.3: Deklaration eines Namensraumes

Einem Namensraum wird ein XML-Schema [3] zugewiesen. Das XML-Schema ist eine Empfehlung des W3C zum Definieren von XML-Elementen und XML-Dokumentstrukturen. Durch das XML-Schema wird das Verbinden von zwei Schemata vereinfacht. So kann man ein drittes XML-Schema herstellen, welches die Struktur der zusammengefügten Dokumente abbildet.

1.3.2 SOAP

SOAP (der Name stand ursprünglich für Simple Object Access Protocol) ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und entfernte Aufrufe *Remote Procedure Calls (RPCs)*, die in Abschnitt 1.6.2 beschrieben sind, durchgeführt werden können. Die SOAP-Spezifikation beschreibt den Aufbau und das Format der Nachrichten, die z.B. bei einem Webservice-Aufruf zwischen Service-Provider und Service-Nutzer ausgetauscht werden. SOAP stützt sich auf andere Standards, XML zur Repräsentation der Daten und Internet-Protokolle der Transport- und Anwendungsschicht zur Übertragung der Nachrichten. Die gängigste Kombination ist SOAP über HTTP und TCP, aber auch andere Transportprotokolle können je nach Anwendung benutzt werden. Die Abkürzung SOAP wird jedoch offiziell seit Version 1.2 nicht mehr als Abkürzung gebraucht, da es erstens nicht mehr so einfach (Simple) ist und zweitens nicht nur dem Zugriff auf Objekte (Object Access) dient. Wie SOAP in den TCP/IP-Protokollstack integriert ist, zeigt Abbildung 1.4.

1.3.3 WSDL

Während Kommunikationsprotokolle und Nachrichtenformate in der Internet-Gemeinschaft standardisiert werden, ist es zunehmend auch möglich und wichtig, die Internet-Kommunikation in einer strukturierten Weise zu beschreiben. Die *Web Service Description Language (WSDL)* geht

[3] http://www.w3.org/XML/Schema

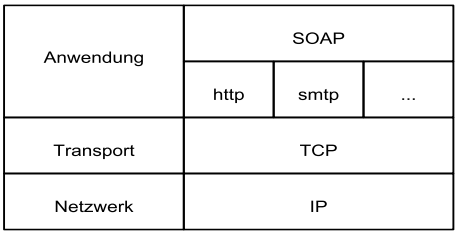

Abbildung 1.4: SOAP im TCP/IP-Protokollstack

dieser Notwendigkeit nach und definiert in [8] eine XML-Spezifikation zur Beschreibung von Webservices. WSDL ist eine Metasprache, mit deren Hilfe die Funktionalität eines Webservices beschrieben werden kann. WSDL beschreibt so zu sagen die Geschäftslogik, die durch die Implementierung in Form von Webservices ein hohes Maß an Flexibilität mit sich bringt, denn diese sind interoperabel und plattformunabhängig. Ein in Java geschriebener Webservice könnte beispielsweise unter Axis (auf Axis wird in Abschnitt 2.5.2 eingegangen) auf einem Linux-Betriebssystem laufen und von einem C#-Windows-Client verwendet werden. Damit nun ein beliebiger Client dynamisch auf den Webservice zugreifen kann, ist es erforderlich, dass der Webservice seine Schnittstellen in einem standardisierten Format zur Verfügung stellt. WSDL definiert auf einer hohen Abstraktionsebene, welche Methoden der Webservice anbietet, welche Parameter diese erwarten und über welche Transportmechanismen und Nachrichtenformate die Kommunikation stattfindet.

Eine in WSDL erstellte Schnittstellenbeschreibung eines Webservices, die vom Entwickler eingesehen oder von einer Clientanwendung angefordert wird, nennt man WSDL-Dokument. Da dieses Dokument ein XML-Dokument ist und kein binäres, ist es für den Menschen lesbar und man sollte mit ein paar Kenntnissen über Aufbau und Struktur solcher Dokumente auf den ersten Blick die Funktionalität des beschriebenen Services erkennen. Eine WSDL-Beschreibung kann außerdem von Clients angefordert werden, um automatisch einen Code zu erzeugen (sog. Proxy-Klassen oder Proxy-Objekte), über den der beschriebene Webservice schließlich aufgerufen wird.

1.3.4 UDDI

Durch die Verbreitung der Webservice-Technologien SOAP und WSDL wurde schnell erkannt, dass ein Vermittlungsmechanismus fehlt, der die Verbindungen zwischen den Beteiligten herstellen bzw. vermitteln kann. Dadurch soll ermöglicht werden, dass Unternehmen und vor allem die von den Unternehmen angebotenen Services schnell und einfach identifiziert werden können. So wurde UDDI von IBM, Microsoft und Ariba ins Leben gerufen und ist heutzutage der De-facto-Standard für XML-Verzeichnisse von Webservices. UDDI steht für „Universal Description, Discovery and Integration“ und stellt neben SOAP und WSDL die dritte Technologiesäule von Webservices dar.

Wenn man das Architekturbild einer typischen, verteilten Anwendung (sei es auf CORBA-, RMI- oder EJB-Basis) mit dem der Webservice-Architektur vergleicht, wird sofort ersichtlich, dass UDDI bzw. der Service-Broker die Rolle des Namensdienstes (CoS-Naming, RMI-Registry oder JNDI) spielt. Ein Namensdienst ist besonders in der Kontaktphase wichtig, weil er die Initialverbindung zwischen Nutzer und Service-Provider herstellen kann. UDDI erfüllt aber nicht

nur die Aufgabe eines klassischen Namensservers, sondern unterstützt wesentlich mehr Funktionalität, indem er Schnittstellen für komplexe Abfragen bereitstellt.

White Pages: Ein UDDI-Verzeichnis enthält Namen, Beschreibungen und sämtliche Kontaktinformationen der Unternehmen, die Webservices publiziert haben.

Yellow Pages: Alle Unternehmen sowie ihre veröffentlichten Services werden innerhalb der UDDI-Registry klassifiziert und kategorisiert, sodass das ganze Verzeichnis wie ein Branchenbuch oder Gelbe Seiten benutzt werden kann.

Green Pages: Ein UDDI-Verzeichnis enthält zusätzlich noch die technischen Dokumentationen (u.a. WSDL), die die veröffentlichten Services beschreiben.

Somit wird dem Benutzer des UDDI-Verzeichnisses die Möglichkeit geboten, auf unterschiedliche Art und Weise intelligente Abfragen zu formulieren, die schnell und effizient zu der gewünschten Information führen.

Die wesentlichen Bestandteile der UDDI-Spezifikation sind die Beschreibung der Datenstrukturen in UDDI und die *Application Programming Interface (API)* zum Abfragen und Publizieren in dem Verzeichnis. UDDI selbst ist ebenfalls eine Web-Service-Anwendung und benutzt SOAP als Kommunikationsprotokoll. Alle APIs und Datenstrukturen werden in XML bzw. im XML-Schema-Format spezifiziert.

1.4 IT-Sicherheit in verteilten Systemen

Definition nach [49]:

„Ein verteiltes System ist eine Menge von einander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen.“

Eine der wichtigsten Stärken von Webservices ist deren Fähigkeit, heterogene Computer, Netzwerke, Anwendungen und Systeme auf vergleichsweise einfache Weise miteinander zu verbinden und zur Zusammenarbeit zu bewegen und nebenbei dem Nutzer, den Anschein eines einzelnen Systems zu erwecken. Dies birgt natürlich große Potenziale, weshalb fast alle Experten dieser Technologie eine große Zukunft voraussagen. Zuerst waren Webservices allerdings relativ selten zum Einsatz gekommen. Einer der Hauptgründe für diese Zurückhaltung war das Fehlen eines Sicherheitsstandards für die SOAP-Kommunikation. Bis eine Sicherheitslösung gesucht wurde, wurden aber Webservices in der Zwischenzeit bereits häufig in Intranets eingesetzt, in denen Sicherheitsfragen wenigstens vorübergehend ausgeklammert werden konnten. Mittlerweile existiert eine Spezifikation namens WS-Security, die ein Verfahren zur sicheren SOAP-Kommunikation beschreibt [51, 11].

In diesem Abschnitt sollen zu Beginn die Grundbegriffe der IT-Sicherheit eingeführt werden. Dann wird geklärt, welche Sicherheitsanforderungen an Webservice-Anwendungen generell zu stellen sind. Anschließend erläutert ein kurzer Exkurs in die Kryptographie die wichtigsten Begriffe und Verfahren moderner Sicherheitstechnologie. Darauf aufbauend wird dargestellt, welche Möglichkeiten es gibt, Webservice-Anwendungen sicher zu gestalten und die eingangs vorgestellten Sicherheitsanforderungen zu erreichen. Zu diesen Möglichkeiten zählen insbesondere verschiedene XML-Sicherheitsstandards.

1.4.1 Definition der Grundbegriffe

In diesem Abschnitt soll erklärt werden, wozu Sicherheit in einem IT-System notwendig ist. Erst wird definiert was, vor wem und wie geschützt werden soll, um dann wird die Definition des Begriffs Sicherheit hergeleitet.

Was? In IT-Systeme geht es um Speicherung und Verwaltung von Informationen. Informationen, die in Form von Daten bzw. Datenobjekten repräsentiert werden, sind abstrakt und müssen erst interpretiert werden. Es existieren passive Objekte (z.B. Dateien, Datenbankeinträge) mit der Fähigkeit, Informationen zu speichern, und aktive Objekte (z.B. Prozesse) mit der Fähigkeit, Informationen sowohl zu speichern als auch zu verarbeiten. Informationen und die entsprechenden Objekte, die sie repräsentieren, sind schützenswerte Güter eines Systems [12].

Vor wem? Datenobjekte sind über wohl definierte Schnittstellen in Form von Operationen bzw. Methoden von anderen Objekten des Systems oder von der Umwelt des Systems benutzbar. Zur Umwelt gehören insbesondere seine Benutzer. Die Benutzer eines Systems und alle Objekte, die im Auftrag von Benutzern im System aktiv sein können, wie z.B. Prozesse, Server und Prozeduren, werden als die Subjekte des Systems bezeichnet.

Wie? Eine Interaktion zwischen einem Subjekt und einem Objekt in einem System, durch die ein Informationsfluss zwischen Subjekt und Objekt auftritt, wird als Zugriff auf die Information bezeichnet. Jeder Zugriff auf ein Datenobjekt ist gleichzeitig auch ein Zugriff auf die dadurch repräsentierte Information. Für den Zugriff auf die zu schützende Information bzw. auf die sie repräsentierenden Daten eines IT-Systems sind Zugriffsrechte festzulegen und an die Subjekte zu vergeben. Besitzt ein Subjekt die Berechtigung zum Zugriff auf eine Information bzw. auf ein Datenobjekt, so ist das Subjekt zu diesem Zugriff autorisiert.

Bei der Definition des Begriffs **Sicherheit** unterscheidet man zwischen Funktionssicherheit (*safety*), Informationssicherheit (*security*) und Datensicherheit (*protection*).

Unter **Funktionssicherheit** eines Systems versteht man die Eigenschaft, dass die realisierte Ist-Funktionalität der Komponenten mit der spezifizierten Soll-Funktionalität übereinstimmt. Ein funktionssicheres System nimmt keine funktional unzulässigen Zustände an. Anders formuliert ist unter der Funktionssicherheit eines Systems zu verstehen, dass es unter allen (normalen) Betriebsbedingungen funktioniert.

Die **Informationssicherheit** ist die Eigenschaft eines funktionssicheren Systems, nur solche Systemzustände anzunehmen, die zu keiner unautorisierten Informationsveränderung oder -gewinnung führen.

Die **Datensicherheit** ist die Eigenschaft eines funktionssicheren Systems, nur solche Systemzustände anzunehmen, die zu keinem unautorisierten Zugriff auf Systemressourcen und insbesondere auf Daten führen. Damit umfasst die so beschriebene Sicherheit der Daten insbesondere auch Maßnahmen zur Datensicherung (*backup*), also den Schutz vor Datenverlusten durch Erstellung von Sicherungskopien.

Mit den getroffenen Festlegungen kann man zwischen Systemen, die auf den Schutz der verarbeiteten Informationen (Informationssicherheit) und solchen, die auf den Schutz der Daten als deren Repräsentanten (Datensicherheit) konzentrieren, unterscheiden.

Aus diesen Definitionen der Sicherheit ist ferner die wesentliche Erkenntnis abzuleiten, dass die Sicherstellung der Informations- und Datensicherheit eines IT-Systems ein Prozess ist, der

einer ständigen Überprüfung und Anpassung unterliegt, da sich Systemeigenschaften über die Zeit dynamisch ändern können. Desweiteren verdeutlichen sie, dass die Funktionssicherheit die Grundlage für die Informations- bzw. Datensicherheit eines Systems ist. Die Funktionssicherheit ist eng verwandt mit den Begriffen der Zuverlässigkeit bzw. der Verlässlichkeit.

In dieser Diplomarbeit wird auf die Sicherheit technischer Systeme im Sinne der Informationssicherheit und Datensicherheit eingegangen. Behandelt werden Konzepte zur Authentifikation und Zugriffskontrolle, die vor unberechtigten Zugriffen auf die Güter des IT-Systems schützen sollen und im Wesentlichen von außen erfolgen. Anzumerken ist, dass die Grenzen zwischen Safety- und Security-Fragestellungen fließend sind und es durchaus Überlappungen gibt.

1.4.2 Schutzziele

Wie im vorigen Abschnitt erklärt, sind Informationen bzw. Daten zu schützende Güter. Der Zugriff auf diese, ist zu beschränken und zu kontrollieren, so dass nur dazu autorisierten Subjekten ein Zugriff gewährt wird. Die Schutzziele, die diese Anforderungen präzisieren, sind Datenintegrität und Informationsvertraulichkeit. Zugreifende Subjekte müssen eindeutig identifiziert und ihre Identität muss verifiziert sein. Die entsprechende Eigenschaft nennt man die Authentizität von Subjekten. Ist ein Subjekt authentifiziert und berechtigt, also autorisiert, einen Zugriff auf ein Objekt bzw. eine Information durchzuführen, dann sollte das System gewährleisten, dass dieser Zugriff auch möglich ist und dass die Daten verfügbar sind. Hat ein Subjekt einen Zugriff bzw. eine Aktion durchgeführt, so ist es in vielen Anwendungen wünschenswert, dass dieser Zugriff bzw. die Aktion eindeutig dem entsprechenden Subjekt zugeordnet werden kann und dieses Subjekt die durchgeführte Aktion nicht im Nachhinein abstreiten kann. Man spricht hier von der Verbindlichkeit des Systems. Die angesprochenen Schutzziele werden nun im Folgenden präzisiert [11, 12, 5].

1.4.2.1 Vertraulichkeit

Definition 1.1 *Vertraulichkeit:*

Ein System gewährleistet Vertraulichkeit (confidentiality), wenn es keine unautorisierte Informationsgewinnung ermöglicht [12].

Eine der wichtigsten Aufgaben im geschäftlichen Leben besteht darin, zu verhindern, dass Unbefugte Zugang zu Daten erlangen, die nicht für sie bestimmt sind. Dabei geht es nicht nur um den Schutz von Passwörtern, Kreditkartennummern oder Geschäftsgeheimnissen, etwa auch darum, private, personenbezogene Informationen für Unbefugte unzugänglich zu machen, mit denen zumindest indirekt finanzieller oder ideeller Schaden angerichtet werden kann. Die Vertraulichkeit von Daten und Kommunikation zählt daher zu den wichtigsten Sicherheitsanforderungen. Sie wird im Wesentlichen durch Verschlüsselung erreicht. Im Zusammenhang mit Webservices bedeutet dies, dass ein Mechanismus vorhanden sein muss, SOAP-Nachrichten zu verschlüsseln. In der Praxis kommen für fortgeschrittene Webservice-Anwendungen häufig besondere Anforderungen hinzu, etwa dass es möglich sein muss, eine Nachricht nur teilweise zu verschlüsseln oder verschiedene Teile der Nachricht jeweils unterschiedlichen Kommunikationspartnern zugänglich zu machen.

1.4.2.2 Integrität

Definition 1.2 *Datenintegrität:*

Ein System gewährleistet Datenintegrität (integrity), wenn es Subjekten nicht möglich ist, die zu schützenden Daten unautorisiert und unbemerkt zu manipulieren [12].

Das Internet ist bekanntermaßen ein öffentliches, unsicheres Netzwerk. Es erfordert kein außerordentliches Expertenwissen, um die Kommunikation zweier Systeme oder Personen abzuhören, einzelne Nachrichten abzufangen, zu manipulieren und die veränderte Nachricht dann an den ursprünglichen Empfänger weiterzuleiten. Daher es ist sehr wichtig, die Integrität empfangener Daten sicherstellen zu können, also zu prüfen, ob die empfangenen Daten auch wirklich die gleichen sind, die der Absender verschickt hat bzw. ob sie auf dem Transportweg manipuliert oder verändert wurden.

1.4.2.3 Authentizität

Definition 1.3 *Authentizität:*

Unter der Authentizität eines Objekts bzw. Subjekts (authenticity) versteht man die Echtheit und Glaubwürdigkeit des Objekts bzw. Subjekts, die anhand einer eindeutigen Identität und charakteristischen Eigenschaften überprüfbar ist [12].

Authentizität beschäftigt sich also mit der Frage der Identität eines Kommunikationspartners. Es muss zweifelsfrei sichergestellt sein, dass ein Kommunikationspartner tatsächlich derjenige ist, für den er sich ausgibt. Im Internet wird üblicherweise versucht, dies durch die Eingabe eines Benutzernamens und Passwortes sicherzustellen. Im Falle von Webservices ist das allerdings nicht immer möglich, da der Kommunikationspartner ein Prozess sein könnte. Nach erfolgreicher Authentifizierung erfolgt in aller Regel eine Autorisierung, also die Entscheidung, zu welchen Daten und Funktionalitäten dem Kommunikationspartner Zugang gewährt wird.

1.4.2.4 Verbindlichkeit oder Unabstreitbarkeit

Definition 1.4 *Definition (Verbindlichkeit):*

Ein System gewährleistet Verbindlichkeit (non repudiation) einer Menge von Aktionen, wenn es einem Subjekt im Nachhinein nicht möglich ist, die Durchführung einer solchen Aktion abzustreiten [12].

Dieses Sicherheitsziel hat besonders im geschäftlichen Leben eine große Bedeutung, nämlich sicherzustellen, dass der Absender einer Nachricht zweifelsfrei bestimmt werden kann und dass der Absender der Nachricht verbindlich bleibt. Es darf nicht möglich sein, dass ein Kunde einen Auftrag an einen Dienstleister schickt und im Anschluss abstreiten kann, den Auftrag jemals erteilt zu haben. Der Dienstleister muss die Möglichkeit haben, nachzuweisen, dass eine ihm vorliegende Nachricht zweifelsfrei von einem bestimmten Absender verschickt wurde. Ebenso muss der Kunde davor geschützt sein, dass ein Dienstleister leugnet, eine vorliegende Versand- oder Buchungsbestätigung jemals verschickt zu haben.

1.4.3 Kryptographische Mechanismen

Als Kryptographie bezeichnet man die Verschlüsselung von (geheimen) Informationen. Kryptographische Verfahren bilden die Grundlage für alle Sicherheitsalgorithmen. Hinter all diesen Verfahren steckt komplexe Mathematik, deren umfassende Erläuterung sich z.B. in [5] findet. Um aber entscheiden zu können, welche Technologien eingesetzt werden sollen, ist es wichtig, sich mit den zugrunde liegenden Begriffen zu beschäftigen und diese zumindest hinsichtlich ihrer Vor- und Nachteile einschätzen zu können. Daher werden in den folgenden Abschnitten zunächst die wichtigsten Grundlagen der Sicherheitstechnologien erläutert [5, 11, 12].

Kryptographische Verfahren, die nachweislich sicher sind, wurden bisher nicht gefunden. Im Gegenteil: Bekannte Verfahren werden im Laufe der Zeit immer unsicherer. Weil kryptographische Verfahren auf mathematischen Problemen basieren, ist ihre Sicherheit von der Schwierigkeit des mathematischen Problems abhängig. Deswegen können heute noch als sicher geltende Verfahren, schon morgen unsicher sein, weil jemand über Nacht eine hervorragende mathematische Idee hatte. In der Praxis werden alle Verfahren als sicher angesehen, für die bisher nicht bekannt ist, wie sie gebrochen werden könnten, was aber nicht bedeutet, dass es unmöglich wäre.

Des Weiteren werden teilweise auch Verfahren als sicher bezeichnet, für die bereits ein Weg bekannt ist, wie man sie brechen kann. Wenn aber die Durchführung eines solchen Angriffs nur theoretischer Natur ist, zum Beispiel weil der technologische und finanzielle Aufwand astronomisch hoch ist, oder weil es schlicht viel zu lange dauert, bis die Verschlüsselung gebrochen wäre. Wenn also der Aufwand nicht das Ergebnis rechtfertigt, dann werden auch solche Verfahren oftmals als sicher angesehen. Um beim Beispiel Online-Banking zu bleiben: Benötigt man Hardware im Wert von mehreren Millionen Euro, um eine Online-Banking-Transaktion innerhalb von 10 Stunden zu entschlüsseln und zu manipulieren, macht dies offensichtlich keinen Sinn, da die Transaktion nach wenigen Minuten entweder abgeschlossen oder aufgrund einer Zeitüberschreitung abgebrochen wurde, und abgesehen davon ein Höchstbetrag von wenigen Tausend Euro pro Transaktion existiert.

1.4.3.1 Datenverschlüsselung

Die erste der genannten Sicherheitsanforderungen war die Vertraulichkeit von Daten und Kommunikation. Vertraulichkeit lässt sich durch Chiffrierung erreichen. Unter Chiffrierung versteht man eine mathematische Funktion oder einen Algorithmus, der verwendet wird, um Daten zu verschlüsseln und zu entschlüsseln. Damit kein Unbefugter die verschlüsselten Daten entschlüsseln kann, muss entweder die mathematische Funktion oder ihre Parameter geheim gehalten werden und nur einer begrenzten Menge von Personen bekannt sein. Meist werden die Parameter geheim gehalten. Nur wer diese Parameter kennt, kann Zugang zu den Originaldaten erlangen. Aus diesem Grund bezeichnet man diese Information auch als Schlüssel. Wird ein solcher Schlüssel verwendet, muss die Chiffrierung nicht mehr geheim sein, sondern nur noch der Schlüssel. Im Gegenteil: Im Allgemeinen werden häufig nur solche Chiffrierungen als sicher angesehen, deren Algorithmus offen gelegt wurde, sodass die Fachwelt Gelegenheit hatte, diesen auf seine Anfälligkeit gegenüber Angriffen zu untersuchen. Ein Verfahren gilt als sicher, solange kein Experte eine Schwachstelle finden konnte. Für die Verschlüsselung und Entschlüsselung von Daten muss nicht notwendigerweise derselbe Schlüssel verwendet werden. Bei den Algorithmen zur Verschlüsselung unterscheidet man zwischen symmetrischen und asymmetrischen Verfahren.

Symmetrische Verfahren

Bei den sogenannten symmetrischen Verfahren wird der gleiche Schlüssel verwendet, um Daten zu verschlüsseln und zu einem späteren Zeitpunkt wieder zu entschlüsseln. Handelt es sich bei den Daten um eine Nachricht, die von einem Sender an einem Empfänger verschickt wird, müssen demnach beide im Besitz dieses Schlüssels sein. Das bedeutet, dass die Kommunikationspartner den Schlüssel entweder im Voraus vereinbart haben oder auf sicherem Wege austauschen müssen. Hierbei ist darauf zu achten, dass der Schlüssel geheim bleibt und nicht in falsche Hände gerät, da jeder, der den Schlüssel kennt, die chiffrierten Nachrichten entschlüsseln kann. Man spricht in diesem Zusammenhang daher auch von geheimen, gemeinsamen Schlüsseln (*secret keys* oder *shared keys*).

Einer der Vorteile symmetrischer Verschlüsselungsverfahren liegt in ihrer Leistungsfähigkeit. Sie sind in der Regel deutlich schneller als asymmetrische Verfahren.

Ein großer Nachteil ergibt sich dagegen aus der großen Anzahl benötigter Schlüssel in Szenarien mit vielen Kommunikationspartnern. Da jeweils zwei Kommunikationspartner einen gemeinsamen geheimen Schüssel verwenden müssen, werden beispielsweise bei 15 Beteiligten schon 105 Schlüssel benötigt, für eine Stadt mit 60.000 Einwohnern gar 1.799.970.000 Schlüssel. Zu den bekanntesten symmetrischen Verfahren zählen DES, 3DES und AES.

Asymmetrische Verfahren

Bei asymmetrischen Verfahren besitzt jeder Kommunikationspartner genau zwei Schlüssel: einen öffentlichen (*public key*) und einen privaten (*private key*). Während der private Schlüssel geheim gehalten wird, kann der öffentliche Schlüssel beliebig verteilt werden. Er kann zum Beispiel in einem öffentlichen Schlüsselverzeichnis (*key server*) abgelegt, auf der Website oder in einer Zeitung veröffentlicht bzw. als Teil der E-Mail-Signatur an versendete E-Mails angehängt werden.

Möchte man also eine chiffrierte Nachricht versenden, verschlüsselt man diese mit dem öffentlichen Schlüssel des Empfängers. So ist sichergestellt, dass nur dieser die Nachricht entschlüsseln kann, nämlich mit dem von ihm geheim gehaltenen privaten Schlüssel. Die Sicherheit asymmetrischer Verfahren beruht auf der Tatsache, dass es nicht möglich ist, den privaten Schlüssel aus dem öffentlichen herzuleiten. Eines der bekanntesten asymmetrischen Verfahren ist RSA, das nach seinen Erfindern Ronald L. Rivest, Adi Shamir und Leonard Adleman benannt ist.

Der Vorteil von asymmetrischen Verfahren ist zum einen, dass zwischen zwei Kommunikationspartnern keine geheimen Schlüssel über potenziell unsichere Wege ausgetauscht werden müssen. Zum anderen sind pro Beteiligtem nur genau zwei Schlüssel notwendig. Bei 15 Beteiligten bedeutet das also 30 Schlüssel, in einer Stadt mit 60.000 Einwohnern sind demnach 120.000 Schlüssel notwendig, also in jedem Fall deutlich weniger als bei symmetrischen Verfahren. Allerdings kann die Verwaltung öffentlicher Schlüssel unter Umständen einen nicht unerheblichen Aufwand bedeuten.

Hybride Verfahren

In der Praxis kommen häufig sogenannte hybride Verfahren zum Einsatz, welche die Vorteile symmetrischer (Schnelligkeit) und asymmetrischer (Sicherheit) Verfahren kombinieren. Die Verschlüsselung der Daten erfolgt dabei mittels symmetrischer Verfahren, wobei für jede Sitzung ein eigener Schlüssel generiert wird, der dann durch asymmetrische Verschlüsselung dem Kommunikationspartner übermittelt wird. Da ein Schlüssel selbst nur eine sehr geringe Datenmenge darstellt, fällt hier die schlechtere Performance der asymmetrischen Verfahren nicht

wesentlich ins Gewicht. Das bekannteste Verfahren dieser Art ist sicherlich *Secure Socket Layer (SSL)*, das unter anderem in Browsern zum Einsatz kommt, wenn sichere Verbindungen benötigt werden, zum Beispiel beim Online-Banking. Auf SSL wird in Abschnitt 1.6.3 näher eingegangen.

1.4.3.2 Digitale Signaturen

Asymmetrische Verfahren können neben der Verschlüsselung von Daten auch zur Erzeugung digitaler Signaturen eingesetzt werden. Diese bilden das elektronische Gegenstück zur handschriftlichen Unterschrift und können dazu verwendet werden, die restlichen drei beschriebenen Sicherheitsanforderungen, Sicherstellung der Identität des Kommunikationspartners, Integrität der Daten und Unabstreitbarkeit, zu erfüllen.

Sollen Daten, also zum Beispiel eine Nachricht, signiert werden, wird typischerweise zunächst ein Hashwert (*Message Digest (MD)*) erzeugt. Dies ist eine Art Prüfsumme, die mit Hilfe einer mathematischen Einweg- oder Hash-Funktion errechnet wird. Dabei wird eine Datenmenge beliebiger Größe eindeutig auf einen sehr kleinen Wert fester Länge abgebildet. Hash-Funktionen haben zum einen die wichtige Eigenschaft, dass es nicht oder nur sehr schwer möglich ist, von einem errechneten Message Digest auf die Ursprungsnachricht zurückzuschließen, zum anderen ergeben selbst geringste Änderungen in den Ursprungsdaten deutliche Abweichungen im Message Digest. Bekannte Algorithmen sind *Message Digest Algorithm (MD5)* von RSA Datasecurity Inc. und *Secure Hash Algorithm (SHA)*.

Gegebenenfalls können dem Message Digest noch Informationen über den Unterzeichner, ein Zeitstempel oder weitere nützliche Daten hinzugefügt werden. Anschließend wird die resultierende Zeichenkette mit Hilfe des privaten Schlüssels des Unterzeichners verschlüsselt. Die resultierende Datenmenge bezeichnet man als Signatur. Diese Signatur wird schließlich an die weiterhin im Klartext befindlichen Daten bzw. Nachricht angehängt.

Um die Signatur einer Nachricht zu verifizieren, wird zunächst der öffentliche Schlüssel des angeblichen Absenders benötigt. Gelingt es mit Hilfe dieses öffentlichen Schlüssels, die Signatur zu entschlüsseln, so ist sichergestellt, dass die Nachricht tatsächlich von diesem Absender stammt, denn nur er ist im Besitz des zugehörigen privaten Schlüssels, um die Signatur zu erzeugen. Die Identität des Kommunikationspartners ist also sichergestellt. Ebenso kann der Absender nicht abstreiten, die Signatur angefertigt zu haben, womit die Verbindlichkeit sichergestellt wäre.

Um auch noch die Datenintegrität sicherzustellen, muss zusätzlich der in der Signatur enthaltene Message Digest verifiziert werden. Dazu errechnet der Empfänger der Nachricht ebenfalls den Message Digest und vergleicht ihn mit dem in der Signatur enthaltenen. Sind beide identisch, so ist die Nachricht auf dem Transportweg nicht verändert worden. Dem Empfänger der Nachricht ist es dadurch möglich, den Message Digest zu berechnen, weil er die empfangenen Daten im Klartext hat und die Einweg- oder Hash-Funktion, die für Erzeugung des Message Digest verwendet wurde, öffentlich oder vereinbart ist.

Beim Versenden von signierten, aber nicht verschlüsselten Nachrichten, muss man beachten, dass es möglich ist, die Signatur zu entfernen und stattdessen eine andere Signatur zu platzieren.

1.4.3.3 Digitale Zertifikate

Wie in den vorangegangenen Abschnitten deutlich wurde, hängt der Einsatz asymmetrischer Verfahren wesentlich davon ab, dass die einzelnen Kommunikationspartner Zugriff auf die öffentlichen Schlüssel der anderen erhalten. Diese können etwa per E-Mail verschickt, auf Webseiten veröffentlicht oder in öffentlichen Schlüsselverzeichnissen abgelegt werden. Wenn sich alle Beteiligten untereinander kennen und sich gegenseitig vertrauen, besteht die Gelegenheit, die Schlüssel im Voraus auszutauschen. Im geschäftlichen Alltag kommt es dagegen häufig vor, dass Nachrichten von bislang Unbekannten empfangen werden (z.B. von Neukunden), oder dass Nachrichten an Unternehmen gesendet werden sollen, mit denen bisher noch kein Kontakt bestand. In diesen Fällen stellt sich nicht nur die Frage, woher der öffentliche Schlüssel des anderen (möglichst automatisiert) bezogen werden kann, sondern vor allem auch, ob der Besitzer eines Schlüssels auch wirklich derjenige ist, für den er sich ausgibt.

Schlüsselpaare können auf einfache Weise von jedermann erzeugt werden. Eine Vielzahl von Programmen und Werkzeugen steht zu diesem Zweck zur Verfügung. Somit stellt ein öffentlicher Schlüssel allein keinerlei Hinweise auf die Identität des Besitzers dar. Es ist wie im „richtigen“ Leben: eine Meldebescheinigung zum Beispiel kann sich jeder Computerbesitzer selbst ausstellen, vorausgesetzt er weiß, wie diese im Allgemeinen aussehen. Eine wirkliche Aussagekraft erlangt dieses Dokument erst dann, wenn die zuständige Behörde mit Hilfe eines Stempels dafür garantiert, dass die enthaltenen Informationen auf ihren Wahrheitsgehalt geprüft wurden. Benötigt wird also eine dritte Partei, eine allgemein als vertrauenswürdig angesehene Instanz wie z.B. das Ordnungsamt, welche die Echtheit des Dokumentes oder die Zugehörigkeit der enthaltenen Daten zu einer konkreten Person garantiert.

Ebenso verhält es sich auch mit den öffentlichen Schlüsseln. Auch hier wird eine Instanz benötigt, welche bestätigt, dass ein Schlüssel zu einer bestimmten Person gehört. An Stelle der Bescheinigung, treten in der elektronischen Welt die sogenannten digitalen Zertifikate, die an Stelle des Stempels, von der vertrauten dritten Instanz signiert werden und Informationen über die bestimmte Person und deren öffentlichen Schlüssel enthalten. Diese digitale Dokumente werden benutzt, um eine Person, einen Server, ein Unternehmen oder Ähnliches zu identifizieren. Die vertrauenswürdige Instanz wird *Certificate Authority (CA)* genannt, bekannte CAs sind z.B. die Firmen VeriSign und Entrust. Der verbreitetste Standard für digitale Zertifikate heißt X.509 [20].

1.4.4 Sicherheit der Webservices

Wie können nun Web-Service-Anwendungen sicher gemacht werden? Letztlich dreht sich diese Frage immer um den Versand der SOAP-Nachrichten, die zwischen Client und Webservice (oder zwischen mehreren Webservices untereinander) ausgetauscht werden, und wie dieser Versand geschützt werden kann. Diese Problematik kann - wie generell in anderen Fällen auch - auf verschiedenen Schichten des ISO/OSI-Modells [1] angegangen werden. Auf der Netzwerkschicht könnte zum Beispiel *Internet Protocol Security (IPsec)* eingesetzt werden. In diesem Fall würden nicht die versendeten Daten, sondern der Kommunikationskanal gesichert werden. Lösungen auf der Netzwerkschicht können in [12] nachgelesen werden. Daneben kann die gewünschte Sicherheit durch Eingriffe auf der Transportschicht oder auf der Anwendungsschicht erreicht werden. Die folgenden Abschnitte erläutern diese beiden Alternativen und beschreiben die jeweiligen Vor- und Nachteile [51].

1.4.4.1 Sicherheit auf der Transportschicht

SOAP-Nachrichten sind zunächst nichts anderes als XML-Dokumente. Diese Dokumente müssen jedoch irgendwie durch das Netz transportiert werden. Es wird also immer auch ein sogenanntes Transportprotokoll benötigt, welches die SOAP-Nachrichten vom Sender zum Empfänger befördert. In der Praxis übernimmt diese Aufgabe heute überwiegend HTTP. Dies ist jedoch keinesfalls vorgeschrieben. Die SOAP-Spezifikation lässt nämlich offen, welches Protokoll für den Transport verwendet werden soll.

Sicherheit auf der Transportschicht bedeutet nun, dass die Funktionalitäten des eingesetzten Transportprotokolls verwendet werden. Im Falle von HTTP, welches an dieser Stelle aufgrund seiner fast ausschließlichen Verwendung betrachtet werden soll, steht daher insbesondere SSL zur Verfügung. Zur Authentifizierung können daneben auch *HTTP Basic Authentication* oder *HTTP Digest Authentication* dienen: Auf sie wird später in Abschnitt 1.6.1 noch eingegangen.

Das SSL-Protokoll, auf das in Abschnitt 1.6.3 kurz eingegangen wird, wurde ursprünglich von Netscape entwickelt. Beim Einsatz von SSL wird eine sichere Verbindung zwischen Client und Server aufgebaut, über welche dann die Daten verschlüsselt versendet werden. Daneben bietet SSL Funktionalitäten zur Identifizierung bzw. Authentifizierung der Kommunikationspartner auf der Basis von digitalen X.509-Zertifikaten. Das Protokoll wird von vielen Websites verwendet, wenn Benutzer oder Kunden vertrauliche Daten wie etwa Kreditkartennummern an eine Web-Anwendung schicken sollen. URLs, die eine SSL-Verbindung erfordern, beginnen per Konvention mit `https:` (*HTTP secure*) statt mit `http:`. Entscheidend ist hier letztlich der Netzwerkport; so werden Anfragen mit https-URL vom Browser intern automatisch auf Port 443 geschickt, dem Standardport für HTTP über SSL.

Wenn also SOAP-Nachrichten über HTTP verschickt werden, könnte SSL verwendet werden, um diese Kommunikation abzusichern. Es ergibt sich hier allerdings das Problem, dass alle Sicherheitsmaßnahmen, die auf der Transportschicht wirken, nur sogenannte Punkt-zu-Punkt-Sicherheit bieten, nicht aber Ende-zu-Ende-Sicherheit. Wenn die Kommunikation über Zwischenstationen (*intermidiaries*) erfolgt, kann die durchgehende Vertraulichkeit nicht mehr gewährleistet werden, weil die Daten auf Zwischenstationen im Klartext vorliegen. Die Zwischenstationen sind ein wichtiger Bestandteil fortgeschrittener Webservice-Architekturen, auf die in Abschnitt Delegation 1.9 eingegangen wird. Man versteht darunter Webservices, die SOAP-Nachrichten zwar empfangen und gegebenenfalls auch verarbeiten, die gleichen Nachrichten aber anschließend an einen oder mehrere andere Webservices weiterleiten. Ein Beispiel für eine solche Zwischenstationen könnte der Webservice eines Händlers sein, der SOAP-Nachrichten von seinen Kunden erhält, die Informationen über die bestellten Waren sowie über das gewünschte Zahlungsmittel enthalten. Der Webservice des Händlers würde diese SOAP-Nachrichten zunächst beispielsweise an ein Kreditinstitut weiterleiten, welches die Informationen zur Zahlung prüft und gegebenenfalls das Konto oder die Kreditkarte des Kunden belastet. Meldet das Kreditinstitut an den Händler, dass die Zahlung erfolgreich abgeschlossen wurde, kann dieser mit dem Versand der bestellten Ware beginnen.

Punkt-zu-Punkt-Sicherheit, wie SSL sie bietet, bedeutet, dass die Kommunikation nur jeweils zwischen den einzelnen Kommunikationspartnern gesichert abläuft, also quasi zwischen den einzelnen Stationen. Die SOAP-Nachricht kann nur auf dem Transportweg vom Kunden zum Händler und gegebenenfalls auch auf dem Transportweg vom Händler zum Kreditinstitut vor unbefugtem Einsehen geschützt werden. Der Händler und auch das Kreditinstitut können die komplette Nachricht im Klartext einsehen. Nun ist aber unter Umständen der Kunde nicht damit

einverstanden, dass der Händler die Nummer seiner Kreditkarte erfährt. Das muss der Händler auch nicht, denn schließlich reicht ihm die Information des Kreditinstituts, dass die Zahlung erfolgreich abgewickelt wurde. Ebenso kann der Kunde ein berechtigtes Interesse daran haben, dass das Kreditinstitut nicht erfährt, was er zu kaufen gedenkt. Für das Kreditinstitut genügen die Informationen, wer an wen zahlen möchte, um welche Summe es sich handelt und mit welchem Zahlungsmittel dies geschehen soll. Zudem kann der Kunde nicht sicher sein, ob der Händler für den weiteren Transport der Nachricht zum Kreditinstitut überhaupt irgendeine Form der Kommunikationssicherung einsetzt. Wünschenswert ist demnach, dass ein Teil der in der SOAP-Nachricht enthaltenen Informationenen nur für den Händler einzusehen ist, während ein anderer Teil, nämlich die Zahlungsinformationen, der Zwischenstation verborgen bleibt und nur für den endgültigen Empfänger der Nachricht einzusehen ist. In solchen Fällen spricht man von Ende-zu-Ende-Sicherheit, und diese ist mit Sicherheitsmaßnahmen auf der Transportschicht nicht zu erreichen. Hierfür muss auf der Anwendungsschicht eingegriffen werden, mit der sich der nächste Abschnitt befasst.

1.4.4.2 Sicherheit auf der Anwendungsschicht

Sicherheitsmaßnahmen auf der Anwendungsschicht erfordern mehr Arbeit als solche auf der Transportschicht. Während das verwendete Transportprotokoll mehr oder weniger transparent umgeschaltet werden kann, bedeutet ein Eingreifen auf der Anwendungsschicht, dass die SOAP-Nachrichten inhaltlich modifiziert werden. Die gewünschte Sicherheit wird dadurch erreicht, dass innerhalb der Anwendungen ein bestimmtes Verschlüsselungsverfahren auf die zu versendenden Daten angewandt wird, bevor diese versendet werden. Somit ist garantiert, dass die eigentliche Nachricht gesichert bleibt, selbst wenn sie zwischenzeitlich über unsichere Transportwege verschickt oder von nicht vertrauenswürdigen Zwischenstationen verarbeitet wird.

Die einzelnen Kommunikationsendpunkte der Anwendung müssen das verwendete Sicherheitsverfahren in diesem Fall kennen und damit umgehen können. Das bedeutet, dass sowohl auf Seiten der Clients als auch auf Seiten der Webservices zusätzlicher Programmieraufwand entsteht. Für den Entwurf von Sicherheitslösungen gibt es bereits einige Standards wie *XML Encryption (XMLenc)* oder *XML Signature (XMLdsig)*, die in Abschnitten 1.4.5 und 1.4.5.1 beschrieben werden.

1.4.4.3 Webservices und Firewalls

Moderne Firewalls, die SOAP „verstehen“ und SOAP-Nachrichten auf der Basis ihres Inhaltes filtern und diese beispielsweise gegen XML-Schema-Definitionen validieren, können z.B. feststellen, ob eine SOAP-Nachricht gültige Werte für den empfangenden Webservice enthält und damit den Service vor Angriffen wie Pufferüberläufe schützen. Ebenso könnten sie sicherstellen, ob der Webservice, an den eine eintreffende SOAP-Nachricht gerichtet ist, für den Absender überhaupt verfügbar sein soll. Für solche Funktionalitäten sind präzise Informationen über das Format und den Inhalt der Nachrichten nötig, die ein Webservice erwartet. WSDL und XML-Schema ermöglichen die Definition dieser Informationen.

1.4.5 XML Security

Wenn es darum geht, Webservice-Anwendungen sicher zu gestalten, muss das Rad glücklicherweise nicht völlig neu erfunden werden. Die Entwickler von Webservice-Anwendungen können heute auf bereits definierte Standards zurückgreifen, die allgemein für XML-Anwendungen erarbeitet wurden. Dies sind vor allen Dingen XMLdsig und XMLenc.

1.4.5.1 XML Signature

XML Signature [33] ist eine Empfehlung aus dem Jahr 2002, die von W3C und IETF gemeinsam erarbeitet wurde. Die Spezifikation beschreibt, wie digitale Signaturen als XML-Struktur dargestellt werden können. Signaturen können grundsätzlich auf beliebigen digitalen Daten errechnet worden sein, so z.B. auch auf Bildern oder auf einem Word-Dokument. Signaturen für XML-Dokumente (wie etwa einer SOAP-Nachricht) sind nur einer von vielen Anwendungsfällen für XMLdsig.

Um die Erzeugung und den Aufbau von XMLdsig zu verstehen, ist es zunächst wichtig zu wissen, was unter der Kanonisierung von XML-Dokumenten zu verstehen ist. Bei der Signierung von XML-Elementen oder auch ganzen XML-Dokumenten gilt es, eine zusätzliche Schwierigkeit zu überwinden, die aus einer Grundeigenschaft von XML resultiert. XML ist nämlich weitgehend formatfrei ausgelegt. Das bedeutet, dass einem XML-Dokument zum Beispiel beliebig viele Leerzeichen oder Zeilenumbrüche hinzugefügt werden können, ohne dass sich die Bedeutung des Dokumentes verändert. Weiterhin ist es etwa möglich, die Reihenfolge von Elementattributen beliebig zu verändern. Es gibt also für logisch ein und dasselbe XML-Dokument mehrere Darstellungen. Hierdurch ergeben sich allerdings Probleme hinsichtlich der Berechnung des Hashwerte, da die entsprechenden Hash-Funktionen, wie in Abschnitt 1.4.3.2 erläutert, absichtlich so konstruiert sind, dass sie selbst im Falle geringster Änderungen in den Ursprungsdaten ein völlig anderes Ergebnis liefern. Die Verifikation der Signatur des Hashwertes würde also fehlschlagen, wenn es keine einheitliche Form der XML-Dokumente gäbe.

Um diesen Problemen entgegenzutreten, gibt es *Canonical XML*, eine Empfehlung des W3C aus dem Jahr 2001 [4]. Die Spezifikation beschreibt ein Verfahren, mit dem untersucht werden kann, ob zwei XML-Dokumente identisch sind. Hierzu werden die Dokumente in ein einheitliches Format gebracht, das als kanonische Darstellung bezeichnet wird. So werden überflüssige Leerzeichen entfernt sowie Attribute und Namensraumdeklarationen lexikalisch sortiert. Um also Hashwerte sinnvoll für XML-Dokumente einsetzen zu können, ist eine solche Kanonisierung der Ursprungsdokumente unbedingt erforderlich. Dabei können theoretisch auch andere Verfahren als Canonical XML zum Einsatz kommen, solange sie sowohl dem Absender als auch dem Empfänger bekannt sind.

Eine Signatur wird gemäß der Spezifikation von einem Element namens `Signature` repräsentiert. Darin verschachtelt befinden sich die weiteren Elemente `SignedInfo`, `SignatureValue` und optional auch noch ein Element namens `KeyInfo`.

In `SignedInfo` befinden sich die Informationen, die letztlich signiert werden. Hierzu zählen unter anderem Referenzen auf die Daten, die signiert werden sollen, sowie Hashwerte, die auf diesen Daten errechnet wurden. Tatsächlich werden also nicht die Daten selbst, sondern deren Hashwert signiert, was natürlich wesentlich weniger Rechenzeit beansprucht. `SignatureValue` enthält den Signaturwert, der über `SignedInfo` errechnet wurde. Abbildung 1.5 illustriert den Aufbau einer Signatur mit einem `CanonicalizationMethod`-Element.

```
<Signature Id="MyFirstSignature"
  xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
      Algorithm"http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="http://www.entwickler.com/data.xml">
      <Transforms>
        <Transform Algorithm=
          "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      </Transforms>
      <DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>j6lwx3rvEPQOvKtMup4NbeuS</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>MCCcFFrvLti... </SignatureValue>
</Signature>
```

Abbildung 1.5: Aufbau einer Signatur mit einem `CanonicalizationMethod`-Element

In Abbildung 1.5 enthält `SignedInfo` nur eine einzige Referenz. Sie verweist mit Hilfe des Attributes URI auf ein XML-Dokument auf dem Server *www.entwickler.com*. Dieses Dokument ist also signiert worden. Es ist ebenso möglich, dass `SignedInfo` mehrere Referenzen enthält. In diesem Fall würden dann mehrere Daten oder Dokumente gemeinsam signiert werden. Weiterhin ist auch möglich, dass sich die zu signierenden Daten im gleichen XML-Dokument befinden wie die Signatur selbst. Ist dies der Fall, müssen die zu signierenden Elemente mit einem Attribut namens `Id` versehen werden, also z.B. `<Body Id= “MsgsBody“>`. Die Referenzen verweisen dann auf diese Elemente mit Attributen der Form `URI=“#MsgBody“`.

Innerhalb jeder einzelnen Referenz befinden sich Informationen über den Hashwert, der über den referenzierten Daten errechnet wurde (`DigestValue`-Element), sowie über den Algorithmus, der hierfür zum Einsatz kam (`DigestMethod`-Element). Daneben können auch noch Transformationen angegeben werden, die auf den Daten durchgeführt wurden, bevor der Digest errechnet wurde. Dies können zum Beispiel XSLT-Transformationen sein, oder aber auch Kanonisierungen. In Abbildung 1.5 verweist im Element `Transform` das Attribut `Algorithm` mit Hilfe eines Namensraumes auf den Kanonisierungsalgorithmus nach Canonical XML.

Das Element `SignatureMethod` enthält Informationen über den verwendeten Signaturalgorithmus. Es handelt sich um eine Kombination aus einem Algorithmus (hier DSA) und Digest-Algorithmus (hier SHA-1). Bevor `SignedInfo` aber signiert werden kann, muss es unbedingt kanonisiert werden. Der hierfür eingesetzte Algorithmus ist im Element `CanonicalizationMethod` abgelegt. Die Validierung von `SignedInfo` auf Seiten des Empfängers besteht aus zwei Schritten: zum einen ist die Signatur über `SignedInfo` zu validieren, zum anderen den Hashwert jeder einzelnen Referenz innerhalb `SignedInfo`.

Besondere Beachtung verdient noch das optionale Element `KeyInfo`. Es kann dazu verwendet werden, dem Empfänger der signierten Daten anzuzeigen, welcher Schlüssel für die Überprüfung der Signatur benötigt wird. Wird `KeyInfo` weggelassen, so setzt dies voraus, dass der Empfänger über diese Information bereits verfügt. `KeyInfo` kann symbolische Namen für

Schlüssel enthalten, digitale Zertifikate oder auch die Schlüssel selbst. Die Spezifikation von XMLdsig definiert einige wenige Typen, die innerhalb von `KeyInfo` verwendet werden können, z.B. `DSAKeyValue`, `RSAKeyValue`, `PGPData` oder `X509Data`. Diese können jedoch von Anwendungsentwicklern erweitert oder durch eigene ersetzt werden.

Abbildung 1.6 demonstriert den Aufbau einer Signatur unter der Verwendung von `KeyInfo` anhand eines RSA-Schlüssels:

```
<Signature Id="MyFirstSignature"
  xmlns="http://www.w3.org/2000/09/xmldsig\#">
  <SignedInfo>...</SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>
          xA7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6W0
          ocLZAtNfyxSZDU16ksL6WjubafOgNEpcwR3RdF
          sT7bCgnXPBe5ELh5u4VEy19MzxkXRgrMvavzyB
          pVRgBUwU1V5foK5hhmbktQhyNdy/6Lp5foK5hh
          mbk5foK5hhmbktQhyNdy/61-pQRhDUDsTvK+g9
          cj47es9AQJ3U==
        </Modulus>
        <Exponent>AQAB</Exponent>
      </RSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
```

Abbildung 1.6: Aufbau einer Signatur mit einem `KeyInfo`-Element

1.4.5.2 XML Encryption

XML Encryption [24] wurde im Jahre 2002 vom W3C als Empfehlung veröffentlicht. Die Spezifikation beschreibt, wie verschlüsselte Daten als XML-Konstrukt dargestellt werden können. Bei den verschlüsselten Daten kann es sich dabei, ebenso wie bei XMLdsig, um beliebige Daten handeln. Das Resultat der Verschlüsselung ist ein Element namens `EncryptedData`, welches die verschlüsselten Daten entweder enthält (in Form eines verschachtelten Elements) oder diese durch einen URI referenziert.

Handelt es sich bei den zu verschlüsselnden Daten um ein XML-Dokument, also zum Beispiel um eine SOAP-Nachricht, besteht die Möglichkeit, entweder das gesamte Dokument zu verschlüsseln oder stattdessen nur einzelne Elemente. Fällt die Entscheidung auf einzelne Elemente, kann für jedes der Elemente weiter entschieden werden, ob das gesamte Element verschlüsselt werden soll (inklusive aller verschachtelten Elemente) oder nur dessen Inhalt. Man spricht hier von unterschiedlichen Granularitäten der Verschlüsselung. Zur Veranschaulichung soll das Beispiel aus der Spezifikation dienen, dargestellt in Abbildung 1.7.

Im Falle der Verschlüsselung des kompletten Dokumentes wird das Wurzelelement `PaymentInfo` durch das bereits angesprochene Element `EncryptedData` ersetzt. `EncryptedData`

```
<?xml version="1.0"?>
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Abbildung 1.7: Ein Beispiel für Zahlungsinformationen

besitzt in Abbildung 1.8 das optionale Attribut `MimeType`, welches einen Hinweis auf den Medientyp der Ursprungsdaten gibt. Das Element `CipherValue` enthält die verschlüsselten Daten.

```
<?xml version="1.0"?>
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
  MimeType="text/xml">
  <CipherData>
    <CipherValue>A23B45C56...</CipherValue>
  </CipherData>
</EncryptedData>
```

Abbildung 1.8: Verschlüsselung des kompletten Dokumentes

Soll stattdessen nur ein einzelnes Element verschlüsselt werden, also zum Beispiel das Element, welches die Kreditkartendaten enthält, tritt `EncryptedData` demnach an die Stelle des Elementes `CreditCard`. Der Rest des Dokumentes bleibt unangetastet, wie in Abbildung 1.9 zu sehen.

```
<?xml version="1.0"?>
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
   EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
    Type="http://www.w3.org/2001/04/xmlenc\#Element">
     <CipherData>
      <CipherValue>A23B45C56...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

Abbildung 1.9: Verschlüsselung der Kreditkarteninformationen

In diesem Fall besitzt `EncryptedData` das ebenfalls optionale Attribut `Type`. Es zeigt an, dass die Granularität `Element` gewählt wurde. Dadurch, dass das komplette Element `CreditCard` verschlüsselt ist, wird seine Identität komplett versteckt. Ein Angreifer, der die verschlüsselte Nachricht abfängt, weiß nicht, ob der Kunde mit Kreditkarte bezahlt hat oder mit einem anderen Zahlungsmittel.

Stattdessen ist es auch möglich, dass nur der Inhalt von `CreditCard` verschlüsselt werden soll, etwa wenn einzelne Beteiligte in der Verarbeitungskette des Dokumentes zwar durchaus wissen müssen, ob die Zahlung per Kreditkarte erfolgt ist, die genauen Daten der Karte jedoch geschützt werden sollen. In diesem Fall werden also nur die Inhalte von `CreditCard` durch `EncryptedData` ersetzt. Das Attribut `Type` zeigt nun an, dass die Granularität `Content` verschlüsselt wurde 1.10.

```
<?xml version="1.0"?>
  <PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <EncryptedData
      xmlns="http://www.w3.org/2001/04/xmlenc\#"
      Type="http://www.w3.org/2001/04/xmlenc\#Content">
      <CipherData>
        <CipherValue>A23B45C56...</CipherValue>
      </CipherData>
    </EncryptedData>
  </Creditcard>
</PaymentInfo>
```

Abbildung 1.10: Verschlüsselung der Kinderelemente

In den bisherigen Beispielen fehlte jegliche Angabe darüber, mit welchem Verfahren und welchem Schlüssel die Daten verschlüsselt wurden. Der Empfänger solcher Daten muss daher auf einem anderen Weg über den verwendeten Algorithmus informiert werden, da er ansonsten das Dokument nicht entschlüsseln kann. Alternativ können diese Informationen aber auch mit Hilfe des optionalen Elementes `EncryptionMethod` in das Dokument mit aufgenommen werden 1.11.

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
  Type="http://www.w3.org/2001/04/xmlenc\#Element"/>
  <EncryptionMethod Algorithm=
    http://www.w3.org/2001/04/xmlenc\#tripledes-cbc"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
    <ds:KeyName>Key A</ds:KeyName>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>A23B45C56...</CipherValue>
  </CipherData>
</EncryptedData>
```

Abbildung 1.11: Verschlüsselung mit Angabe von Verschlüsselungsmethode

Das Attribut `Algorithm` zeigt an, dass zur Verschlüsselung der Daten das symmetrische Verfahren Triple-DES eingesetzt wurde. Daneben befindet sich im Element `KeyInfo` der Hinweis, welcher Schlüssel verwendet wurde: es handelt sich um einen Schlüssel, der dem Empfänger unter dem symbolischen Namen `Key A` bekannt sein sollte. Das Element `KeyInfo` stammt eigentlich aus der Spezifikation für XMLdsig und wurde dort bereits näher vorgestellt.

Eine weitere Möglichkeit besteht darin, den verwendeten Schlüssel ebenfalls in das Dokument

mit aufzunehmen, also zusammen mit den verschlüsselten Daten. Dies macht natürlich nur dann Sinn, wenn der Schlüssel zuvor selbst chiffriert wurde, zum Beispiel mit Hilfe eines asymmetrischen Verfahrens. Das in Abbildung 1.12 dargestellte Beispiel ist dem vorherigen sehr ähnlich. Es unterscheidet sich lediglich darin, dass im Element `KeyInfo` dieses Mal nicht ein symbolischer Name auf den verwendeten Schlüssel hinweist, sondern ein Element namens `RetrievalMethod`. Dieses besitzt zwei Attribute, die zum einen anzeigen, wo sich der verwendete Schlüssel befindet (nämlich weiter unten im gleichen Dokument, beim Element mit der `Id=KeyA`), und zum anderen, dass es sich hierbei um einen chiffrierten Schlüssel handelt (`Type= ... EncryptedKey`). `Key A` wurde also verschlüsselt im gleichen Dokument abgelegt.

```
<EncryptedData Id="SecretData"
  xmlns="http://www.w3.org/2001/04/xmlenc\#">
  <EncryptionMethod Algorithm=
    "http://www.w3.org/2001/04/xmlenc\#tripledes-cbc"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
      <ds:RetrievalMethod URI="\#KeyA"
      Type="http://www.w3.org/2001/04/xmlenc\#EncryptedKey"/>
    </ds:KeyInfo>
    <CipherData>
      <CipherValue>DEADBEEF</CipherValue>
  </CipherData>
</EncryptedData>
<EncryptedKey Id="KeyA"
  xmlns="http://www.w3.org/2001/04/xmlenc\#">
  <EncryptionMethod Algorithm=
    "http://www.w3.org/2001/04/xmlenc\#rsa-1_5"/>
  <ds:KeyInfo
    xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
    <ds:KeyName>Key B</ds:KeyName>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>xyzabc...</CipherValue>
  </CipherData>
  <ReferenceList>
    <DataReference URI="\#SecretData"/>
  </ReferenceList>
</EncryptedKey>
```

Abbildung 1.12: Hybride Verschlüsselung

Das Element `EncryptedKey` enthält den verschlüsselten Schlüssel `Key A`. Wie an `EncryptionMethod` zu erkennen ist, wurde `Key A` mit Hilfe des asymmetrischen Verfahrens RSA verschlüsselt. `KeyInfo` enthält den symbolischen Namen des öffentlichen Schlüssels, mit dessen Hilfe `Key A` wieder entschlüsselt werden kann. In `CipherData` ist der verschlüsselte `Key A` abgelegt, `ReferenceList` enthält eine Liste aller Objekte, die mit `Key A` verschlüsselt wurden.

Was auf den ersten Blick sehr kompliziert erscheint, ist im Grunde ganz einfach. Zusammengefasst: Um Daten vor den Blicken Unbefugter zu schützen, werden sie mit einem symmetrischen Verfahren verschlüsselt. Der symmetrische Schlüssel wird zusammen mit den Daten in das Do-

kument gepackt. Damit nicht jedermann die Daten wieder entschlüsseln kann, wird der symmetrische Schlüssel selbst zuvor mit dem (asymmetrischen) öffentlichen Schlüssel des Empfängers chiffriert. Möchte dieser die Daten nun einsehen, benutzt er zunächst seinen privaten (asymmetrischen) Schlüssel, um den symmetrischen Schlüssel wiederherzustellen. Mit dessen Hilfe kann er nun die eigentlichen Daten entschlüsseln.

Es stellt sich in diesem Zusammenhang die Frage, weshalb die zu schützenden Daten nicht gleich mit Hilfe des asymmetrischen Verfahrens verschlüsselt wurden. Ein wichtiger Grund hierfür ist die deutlich schlechtere Leistungsfähigkeit asymmetrischer Verfahren, die zwar nicht in solchen kleinen Beispielen, jedoch bei größeren Datenmengen und möglicherweise mehreren gleichzeitigen Benutzern auf dem gleichen Server deutlich bemerkbar wird.

1.4.6 Weitere Sicherheitsstandards

Neben XMLenc und XMLdsig gibt es noch einige weitere XML-Sicherheitstechnologien, die auch im Zusammenhang mit Webservices eine wichtige Rolle spielen. Die wichtigsten sollen an dieser Stelle kurz vorgestellt werden.

Security Assertion Markup Language (SAML)

SAML ist ein XML-basiertes Framework für den Austausch sicherheitsrelevanter Informationen, das von OASIS vorangetrieben wird. Es unterscheidet sich dabei von anderen Ansätzen, die mit zentralen Ausgabestellen für Zertifikate arbeiten, indem diese Informationen in Form von Ausweisen (sogenannten *Assertions*) für Personen oder Computer ausgetauscht werden. In SAML kann jeder Netzwerkknoten den Ausweis erstellen, dass er die Identität eines bestimmten Subjektes kennt und/oder geprüft hat. Es liegt dann am Empfänger solcher SAML-Informationen zu entscheiden, ob er diesem Ausweis vertraut oder nicht. Dieses Prinzip des Austauschs sicherheitsrelevanter Erkenntnisse ist besonders dann von großem Interesse, wenn komplexe Webservice-Anwendungen entstehen, in denen mehrere Services unterschiedlicher Dienstleister zu geschäftlichen Workflows kombiniert werden. Dann müssen SOAP-Nachrichten nämlich durch verschiedene Systeme fließen, um die Gesamttransaktion zu erbringen. SAML alleine stellt aber keine Lösung zur Authentifizierung dar, sondern ist lediglich ein Protokoll für den Austausch von Authentifizierungsinformationen, die auf beliebige Weise gewonnen werden können. SAML-Assertions und das SAML-Protokoll werden in Abschnitt 1.7.3 näher beschrieben.

Extensible Access Control Markup Language (XACML)

Dieser OASIS-Standard ist eng mit SAML verwandt und soll Unternehmen ermöglichen, die Nutzung von Services auf authentifizierte und autorisierte Benutzer zu beschränken. Dies geschieht durch die Formulierung von Regeln, in denen der Autor festlegen kann, welche Zugriffsrechte eine Person hat und welche nicht. Neben der Spezifikation einer Sprache für diese Regeln definiert XACML eine weitere Sprache, mit deren Hilfe einfache Anfragen getätigt werden können, um herauszufinden, ob der Zugriff auf eine Ressource genehmigt werden darf [2].

XML Key Management Specification (XKMS)

XKMS steht unter der Obhut des W3C. Es spezifiziert Protokolle für die Verteilung und Registrierung öffentlicher Schlüssel sowie für die Abfrage bzw. Beschaffung der Schlüssel. Im Detail werden drei verschiedene Services beschrieben und spezifiziert: *Key Information Service Specification (X-KISS)* beschreibt, wie Anwendungen das digitale Zertifikat eines Benutzers sowie

gegebenenfalls weitere über ihn gespeicherte Informationen abfragen können, *Key Registration Service Specification (X-KRSS)* definiert ein Protokoll für die Registrierung von Schlüsseln und optional weiteren zugehörigen Informationen, und *Trust Assertion Service (X-TASS)* ist schließlich eine Architektur und ein Protokoll für die Verknüpfung von Vertrauensbeziehungen mit öffentlichen Schlüsseln.

1.4.7 Web Services Security: SOAP Message Security

Nachdem nun klar ist, wie XML-Dokumente im Allgemeinen verschlüsselt und signiert werden können, stellt sich natürlich die Frage, wie die vorliegenden Lösungen in die Welt der Webservices integriert werden können. Es liegt nahe, die Erweiterbarkeit von SOAP auszunutzen. Zu diesem Zweck veröffentlichten IBM, Microsoft und VeriSign im April 2002 eine gemeinsame Spezifikation unter dem Namen WS-Security. Dann wurde WS-Security an *Organization for the Advancement of Structured Information Standards (OASIS)* abgegeben, wo es weiterentwickelt und als Standard verabschiedet werden soll. OASIS ist ein Zusammenschluss von über 170 Mitgliedern aus der Industrie, die anwendungsübergreifende Standards zum Austausch von strukturierter Informationen erstellt. Unter den Mitgliedern befinden sich Firmen wie Airbus, Deutsche Post, Hewlett Packard, IBM, Intel, Microsoft, Oracle, SAP und SUN. Bei Erstellung dieser Diplomarbeit war der Standard *Web Services Security: SOAP Message Security 1.1* [35], die aktuelle Version.

Diese Spezifikation beschreibt Erweiterungen des SOAP-Protokolls, mit denen die gewünschte Vertraulichkeit und Integrität einer SOAP-Nachricht sowie eine Authentifizierung des Kommunikationspartners durch Verwendung bestehender Standards wie XMLenc und XMLdsig erreicht werden können. Diese Spezifikation ist flexibel und gilt als Basis für eine sichere Nutzung der Webservices innerhalb verschiedener Sicherheitsmodelle wie *Public Key Infrastructure (PKI)*, Kerberos und SSL. Daneben bietet WS-Security einen allgemein gehaltenen Mechanismus, um sogenannte *security token* in SOAP-Nachrichten einzubetten. *security token*, auf die im folgenden Abschnitt eingegangen wird, können beliebige Formen von sicherheitsrelevanten Informationen sein (z.B. zum Identitätsnachweis), WS-Security erfordert jedoch keinen spezifischen Typ. Die Spezifikation beschreibt, auf welche Weise Binärdaten enthaltende *security token* wie X.509-Zertifikate oder Kerberos-*tickets* für den Transport in SOAP-Nachrichten zu kodieren sind.

WS-Security kann keine Sicherheit garantieren oder gar eine vollständige Sicherheitslösung darstellen. Vielmehr dient es als Baustein, der zusammen mit anderen SOAP-basierten Spezifikationen und Protokollen sowie mit anwendungsspezifischen Protokollen zu sicheren Architekturen führen soll. WS-Security beschreibt zum Beispiel, wie digitale Signaturen in SOAP-Nachrichten eingebettet werden können. Ob die Signatur sicher gegen Angriffe ist, ist abhängig von dem kryptographischen Algorithmus, mit dem sie erstellt wurde.

Der SOAP-Header <Security>

WS-Security führt ein neues SOAP-Header-Element namens `wsse:Security` ein. Es dient dazu, sicherheitsrelevante Informationen zu transportieren, die für einen bestimmten Empfänger (*SOAP actor*) bestimmt sind. Dies kann entweder der endgültige Empfänger der Nachricht oder eine Zwischenstation sein. Der Header einer SOAP-Nachricht kann demnach mehrere `wsse:Security`-Elemente enthalten, falls Informationen für verschiedene Empfänger transportiert werden sollen. Ein Attribut namens `actor` zeigt an, für welchen der Empfänger ein jeweiliges Element bestimmt ist. Dabei gilt, dass niemals mehrere `wsse:Security`-Elemente

den gleichen Empfänger haben dürfen, und dass bei genau einem `wsse:Security`-Element das `actor`-Attribut weggelassen werden darf, die darin enthaltenen Informationen sind dann für beliebige Empfänger bestimmt. Die Zwischenstationen dürfen nach Erhalt einer solchen SOAP-Nachricht weitere `wsse:Security`-Elemente hinzufügen oder die in bereits bestehenden `wsse:Security`-Elementen enthaltenen Informationen ergänzen. Das Beispiel in Abbildung 1.13 veranschaulicht den geschilderten Aufbau.

```
<Envelope
  xmlns:S11="..."
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <S11:Header>
   ...
    <wsse:Security S11:actor="..."  S11:mustUnderstand="...">
     ...
    </wsse:Security>
     ...
  </S11:Header>
  ...
</Envelope>
```

Abbildung 1.13: `wsse:Security`-Element in einem `Envelope`

Innerhalb des Elements `wsse:Security` können nun *security token*, Signaturen oder Informationen zur Verschlüsselung der Nachricht enthalten sein. Wie bereits erwähnt, ist WS-Security offen und erweiterbar ausgelegt. Das bedeutet, dass prinzipiell beliebige sicherheitsrelevante Informationen dort eingebettet werden können. Die Spezifikation macht jedoch einige Vorschläge, von denen erwartet wird, dass sie, neben vielen anderen, zum Einsatz kommen werden.

1.4.7.1 Security Tokens

In der Spezifikation werden vier Beispiele im Zusammenhang mit *security token* beschrieben: `UsernameToken`, `BinarySecurityToken`, `EncryptedDataToken` und `SecurityTokenReference`.

`UsernameToken` ist für einfache Authentifizierungsinformationen gedacht. Es kann den Benutzernamen und optional auch ein Passwort des Absenders transportieren. So ein `UsernameToken` ist in Abbildung 1.14 dargestellt. Wird das Passwort weggelassen, geht man davon aus, dass der Empfänger dieses kennt. Wird das Passwort dagegen mitgeschickt, sollte ein sicheres Transportprotokoll wie zum Beispiel SSL zum Einsatz kommen.

Während XML-basierte *security token* also sehr einfach in das `Security`-Element eingebettet werden können, müssen Token mit binärem (z.B. X.509 oder Kerberos) oder einem anderen Nicht-XML-Format auf einem möglichst standardisierten Weg in XML dargestellt werden, um ebenfalls eingefügt werden zu können. Hierzu dient das `BinarySecurityToken`.

Dieses Element besitzt zwei Attribute, die verwendet werden, um seinen Inhalt zu interpretieren: `ValueType` zeigt an, welcher Typ von *security token* kodiert ist (z.B. ein X.509-Zertifikat), `EncodingType` enthält die Information, auf welche Weise das *token* kodiert wurde (z.B.

```
<Envelope xmlns:S11="..." xmlns:wsse="...">
  <S11:Header>
   ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Nutzer</wsse:Username>
        <wsse:Password>MyTestPasswd</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
     ...
  </S11:Header>
</Envelope>
```

Abbildung 1.14: Verwendung vom `UsernameToken`-Element

```
<wsse:BinarySecurityToken
  xmlns:wsse="..."
  Id="MyCertificate" ValueType="wsse:X509v3"
  EncodingType="wsse:Base64Binary">
  MIIEZzCCA9CgAwIBAgIQEmtJZc0hf739ut30t803z3
  nf938HF139fvbfGtfCUvc427jF03eo2k1WDFhj2...
</wsse:BinarySecurityToken>
```

Abbildung 1.15: Verwendung vom `BinarySecurityToken`-Element

Base64-Kodierung oder hexadezimale Darstellung [26]). Abbidung 1.15 zeigt ein X.509-Zertifikat, welches als *security token* kodiert wurde.

Manchmal ist es erwünscht, dass das *security token* in verschlüsselter Form übertragen werden soll. Dafür kann das Element `EncryptedDataToken` genutzt werden, das das *token* in sich kapselt und in den `wsse:Security` Element einfügt. Abbildung 1.16 zeigt ein `EncryptedDataToken`.

```
<wsse:EncryptedDataToken
  MIIEZzCCA9CgAwIBAgIQEmtJZc0hf739ut30t803z3
  nf938HF139fvbfGtfCUvc427jF03eo2k1WDFhj2...
</wsse:EncryptedDataToken>
```

Abbildung 1.16: Verwendung vom `EncryptedDataToken`-Element

Das Element `SecurityTokenReference` dient schließlich dazu, auf *security token* zu verweisen, die nicht zusammen mit der SOAP-Nachricht transportiert werden, sondern stattdessen vom Empfänger auf anderem Wege beschafft werden sollen. Alternativ kann das Element auch verschachtelt innerhalb des Elementes `KeyInfo` aus der XML-Signature-Spezifikation verwendet werden, um auf ein *security token* innerhalb der gleichen SOAP-Nachricht zu verweisen, das für eine Signatur oder Verschlüsselung verwendet wurde. In Abbildung 1.17 ist ein `SecurityTokenReference`-Element dargestellt.

Es können generell beliebige sicherheitsrelevante Informationen als *security token* transportiert werden, zum Beispiel auch SAML-Informationen.

```
<wsse:SecurityTokenReference
    wsu:Id="...", wsse11:TokenType="...",
wsse:Usage="...", wsse:Usage="...">
</wsse:SecurityTokenReference>
```

Abbildung 1.17: Verwendung vom `SecurityTokenReference`-Element

1.4.7.2 Signierte SOAP-Nachrichten

Für das digitale Signieren von SOAP-Nachrichten wird der bestehende Standard XMLdsig eingesetzt. Eine SOAP-Nachricht kann dabei mehrere Signaturen enthalten, die jeweils für verschiedene Teile der Nachricht erstellt wurden. Dies ist insbesondere wichtig für Anwendungen, in denen die Nachrichten durch mehrere Zwischenstationen fließen. In dem typischen Szenario zwischen Kunde, Händler und Kreditinstitut müsste beispielsweise zunächst der Kunde seine Bestellung signieren. Der Händler würde der Bestellung eine Bestellnummer hinzufügen und die Nachricht dann an das Kreditinstitut weiterreichen, welches die nötige Zahlung verbucht und der SOAP-Nachricht eine Transaktions-ID hinzufügt. Schließlich wird das Kreditinstitut die erfolgreiche Durchführung dieser Transaktion bestätigen, indem es seinerseits die Kombination von Bestellnummer und Transaktions-ID signiert. Der Händler kann nun sicher sein, dass sowohl der Kunde tatsächlich eine Bestellung aufgegeben hat und dass ihm ein entsprechender Betrag gutgeschrieben wurde. So kann er mit dem Versand der Ware beginnen.

Das in Abbildung 1.18 dargestellte Beispiel zeigt eine komplette SOAP-Nachricht mit einer fiktiven DVD-Bestellung. Das SOAP-Element `Body` besitzt hier ein Attribut namens `Id` mit dem Wert `MsgBody`. Dieser Bezeichner wird innerhalb der Signatur durch das Attribut `URI` im Element Reference referenziert. Es wurde demnach der gesamte SOAP-Body signiert.

Das zur Signatur gehörende Element `KeyInfo` zeigt an, welcher Schlüssel zur Anfertigung der Signatur verwendet wurde. Dies geschieht durch Verwendung eines `SecurityTokenReference`-Elementes, das auf das Element `BinarySecurityToken` am Anfang des Headers verweist. Dort ist ein X.509-Zertifikat abgelegt.

1.4.7.3 Verschlüsselte SOAP-Nachrichten

WS-Security erlaubt die Verschlüsselung beliebiger Kombinationen von Elementen des SOAP-Bodys, des SOAP-Headers oder des gegebenenfalls angehängten Attachments. Dabei kann entweder ein symmetrischer Schlüssel zum Einsatz kommen, der sowohl dem Sender als auch dem Empfänger bereits zuvor bekannt ist, oder aber der Schlüssel wird selbst verschlüsselt und innerhalb der Nachricht transportiert. Für den gesamten Mechanismus greift WS-Security, ähnlich wie bei den Signaturen, mit XMLenc auf einen bestehenden Standard zurück und beschreibt nur noch, wie die dort beschriebenen Elemente in eine SOAP-Nachricht einzubetten sind. Im Einzelnen handelt es sich um die Elemente `ReferenceList`, `EncryptedKey` und `EncryptedData`.

Wie in Abschnitt 1.4.5.2 bereits erläutert, enthält `EncryptedData` die verschlüsselten Daten. `ReferenceList` kann beim Einsatz von WS-Security, wie in Abbildung 1.19 dargestellt, dazu verwendet werden, alle verschlüsselten Anteile innerhalb des SOAP-Envelope aufzulisten.

```
<?xml version="1.0" encoding="utf-8"?>
  <S:Envelope
    xmlns:S="..." xmlns:ds="..." xmlns:wsse="..." xmlns:xenc="...">
    <S:Header>
      <wsse:Security>
        <wsse:BinarySecurityToken ValueType="wsse:X509v3"
          EncodingType="wsse:Base64Binary" Id="X5O9Token">
          MIIEZzCCA9CgAwIBAgIQEmtJZcOrgrKh5i...
        </wsse:BinarySecurityToken>
        <ds:Signature>
          <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm=".../xml-exc-c14n\#"/>
            <ds:SignatureMethod Algorithm="...\#rsa-sha1"/>
            <ds:Reference URI="\#MsgBody">
              <ds:Transforms>
                <ds:Transform Algorithm=".../xml-exc-c14n\#"/>
              </ds:Transforms>
              <ds:DigestMethod Algorithm="...\#sha1"/>
              <ds:DigestValue>EULddytSol...</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>BL8jdfToEb11/vX... </ds:SignatureValue>
          <ds:KeyInfo>
            <wsse:SecurityTokenReference>
            <wsse:Reference URI="#X509Token"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S:Header>
  <S:Body Id="MsgBody">
    <m:orderDVD xmlns:m="Some-URI">
      <title>The Lord of the Rings</title>
    </m:orderDVD>
  </S:Body>
</S:Envelope>
```

Abbildung 1.18: Eine SOAP-Nachricht mit einem signierten `Body`-Element

Für jeden einzelnen verschlüsselten Block ist dann jeweils ein `DataReference`-Element vorzusehen, welches auf die entsprechende Stelle verweist. Obwohl `ReferenceList` in der Spezifikation von XMLenc eigentlich dafür gedacht war, als Kindelement von `EncryptedKey` auf all jene Stellen zu verweisen, die mit dem gleichen Schlüssel chiffriert wurden, erlaubt WS-Security in diesem Zusammenhang, dass die in der `ReferenceList` aufgeführten Anteile mit verschiedenen Schlüsseln chiffriert wurden. Welche Schlüssel das jeweils sind, kann innerhalb von `EncryptedData` mit dem Element `KeyInfo` angezeigt werden.

1.5 Authentifikation

Die Überprüfung der Authentizität von Objekten und Subjekten ist die Voraussetzung für die Realisierung weiterer Sicherheitsanforderungen wie Integrität und Vertraulichkeit. Gemäß der Definition von Authentizität in Abschnitt 1.4.2 erfordert eine korrekte Authentifikation eines

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig\#"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc\#">
  <S:Header>
    <wsse:Security>
      <xenc:ReferenceList>
        <xenc:DataReference URI="\#bodyID"/>
      </xenc:ReferenceList>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <xenc:EncryptedData Id="bodyID">
      <ds:KeyInfo>
        <ds:KeyName>Key A</ds:KeyName>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </S:Body>
</S:Envelope>
```

Abbildung 1.19: Auflistung der verschlüsselten Elemente im `ReferenceList`-Element

Objekts bzw. Subjekts dessen eindeutige Charakterisierung durch wohldefinierte Eigenschaften, so dass über diese zweifelsfreien Merkmale eine eindeutige Identifizierung möglich ist. Die Aufgabe der Sicherheitsgrundfunktion der Authentifikation besteht darin, mit geeigneten Maßnahmen die Korrektheit einer behaupteten Identität zu kontrollieren, d.h. diese muss von den Objekten bzw. Subjekten nachgewiesen werden. Ganz allgemein spricht man in diesem Zusammenhang häufig von *credentials*, die ein Subjekt zum Nachweis seiner Identität vorzulegen hat. Beispiele für derartige *credentials* sind Benutzerkennung und zugehöriges Passwort oder *tickets*, die von vertrauenswürdiger Stelle ausgestellt werden und die Identität des Besitzers bestätigen. Der Personalausweis wäre ein Beispiel für ein solches *ticket* in nicht-technischen Umgebungen [12].

Es werden Mechanismen benötigt, die eine hohe Fälschungssicherheit der Identität garantieren. Im vorliegenden Abschnitt werden Authentifizierungsmechanismen und -maßnahmen beschrieben, wobei nur auf die Authentifikation von Subjekten eingegangen wird. Die gängigen Authentifikationsmechanismen für Objekte wurden mit den kryptographischen Hashfunktionen und Signaturen bereits in Abschnitt 1.4.3 behandelt.

In heutigen IT-Systemen sind die Subjekte in der Regel Benutzer, Prozesse wie beispielsweise WWW-Server oder Hardwarekomponenten wie PCs. Prozesse werden entweder explizit von den Benutzern des Systems oder implizit bei der Systemnutzung gestartet. In Betriebssystemen werden für jeden Prozess Datenstrukturen verwaltet, die den Prozess eindeutig identifizieren und Informationen enthalten wie z.B. welche Berechtigungen dem Prozess gewährt werden.

In heutigen Systemen greift eine Vielzahl parallel ausgeführter Prozesse sowohl konkurrierend als auch gezielt kooperierend auf gemeinsame Ressourcen wie zum Beispiel Dateien zu. Da solche Zugriffe auch über ein Netz, also als entfernte Zugriffe, abgewickelt werden können, ist die

herkömmliche Authentifikation im Rahmen der Zugangskontrolle vieler Betriebssysteme nicht mehr ausreichend. Sie beschränken sich nämlich nur darauf, einen Benutzer beim Systemzugang zu authentifizieren. Nach dessen erfolgreicher Authentifizierung werden automatisch alle seine Prozesse, die er im Verlauf der Login-Sitzung erzeugt, als authentisch betrachtet.

In Service-orientierten Systemen bieten Server wie beispielsweise Dateiserver oder Datenbankserver Dienste an, die Clientrechner unter Verwendung von Kommunikationsnetzen nutzen. Der Zugriff auf diese Dienste sollte natürlich nur authentischen Subjekten gewährt werden. Um die Authentizität von Client-Prozessen nachzuweisen reicht es aber nicht aus, lediglich eine Identität anzugeben. In offenen Systemen können Identitätsangaben sehr einfach gefälscht und Spoofing-Angriffe (Identitäsübernahme) durchgeführt werden. Aus diesem Grund sind Authentifikationsprotokolle zwischen anfragendem Client und anbietendem Server abzuwickeln.

Bei den in der Praxis eingesetzten Authentifikationstechniken unterscheidet man Maßnahmen, die auf der Kenntnis eines spezifischen Wissens, auf einem persönlichen Besitz oder auf biometrischen Merkmalen basieren. Diese Klassen von Authentifizierungsmaßnahmen können sowohl in zentralen, geschlossenen, als auch in offenen und verteilten Systemen eingesetzt werden.

Zur Realisierung von Authentifikationsverfahren werden häufig Kombinationen von mehreren Authentifikationstechniken verwendet, um auf diese Weise die Sicherheit zu erhöhen und um die Vorteile der unterschiedlichen Techniken gezielt auszunutzen. Bei einer Kombination aus zwei Techniken spricht man häufig von einer Zwei-Faktor-Authentifikation, die in der Regel darauf basiert, dass man etwas wissen und etwas besitzen muss. Ein allgemein geläufiges Beispiel dafür ist die Authentifikation eines Bankkunden an einem Geldautomat. Der Kunde beweist seine Identität zum einen durch den Besitz seiner EC-Karte und zum anderen durch die Kenntnis seiner PIN (spezifisches Wissen). Da sich in diesem Szenario nur der Kunde gegenüber dem Geldautomaten authentifiziert, sieht man schon ein klassisches Beispiel für eine fehlende wechselseitige Authentifikation, das in der IT-Welt noch einfacher ausgenutzt werden kann als an einem Geldautomaten.

1.5.1 Authentifikation durch Wissen

Techniken zur Authentifikation auf der Basis von spezifischem Wissen sind in der Praxis noch immer am häufigsten anzutreffen. In der Regel wird dazu ein Verfahren verwendet, bei dem sich der Benutzer oder ein Gerät durch die Kenntnis eines Geheimnisses authentifizieren muss. Dazu verwendet man so genannte *Challenge-Response-Verfahren (CR-Verfahren)*. Der in der Praxis nach wie vor am häufigsten eingesetzte Spezialfall dieser Authentifizierungsmethoden stellt das Passwort-basierte Verfahren dar.

Passwortverfahren

Mittels eines Passwortverfahrens authentifiziert sich ein Subjekt, indem es die Kenntnis eines mit dem System vereinbarten Geheimnisses nachweist. Passwortbasierte Authentifikation wird in nahezu allen Betriebssystemen für PCs eingesetzt. Das System hat die Passwörter sicher zu verwalten, so dass keine unautorisierten Zugriffe darauf möglich sind.

Ablauf einer Passwort-basierten Zugangskontrolle:

1. Das System verlangt die Identifikation/Kennung des Benutzers, z.B. durch ein *Login-Prompt Login*:

2. Der Benutzer identifiziert sich durch seine Kennung,
   z.B. mit *Login: Nutzer*

3. Das System verlangt die Authentifikation, das der Benutzer
   z.B. über den Prompt *Passwort: passwort* eingibt.

4. Das System vergleicht das Passwort mit dem in der Passwortdatei unter der Benutzeridentifikation abgespeicherten, wofür ggf. das eingegebene Passwort zunächst verschlüsselt werden muss.

5. Im Fehlerfall wird der Benutzer in der Regel informiert, z.B. mit *Login failed*. Bei einer erfolgreichen Authentifizierung, ist der Benutzer *eingeloggt*.

Verwendet nun der Benutzer ein einfach zu bestimmendes Passwort, so ist es für einen Angreifer nicht schwierig, sich unter der Benutzerkennung seines Opfers Zugang zum System zu verschaffen. Die Probleme und Risiken des Passwortverfahrens sind bereits seit langer Zeit allgemein bekannt. So lieferte eine in [32] durchgeführte Untersuchung an 3289 von Benutzern frei gewählten Passworten das in Tabelle 1.1 zusammengefasste Ergebnis.

Anzahl	Prozent	Eigenschaft
72	2%	Passwort aus zwei ASCII Zeichen
464	14%	Passwort aus drei ASCII Zeichen
477	14%	Passwort aus vier alphanumerischen ASCII Zeichen
706	21%	Passwort aus fünf Groß- oder Kleinbuchstaben
605	18%	Passwort aus sechs Kleinbuchstaben
492	15%	Passwort stand in einem Wörterbuch o.ä.

Tabelle 1.1: Schwächen benutzerwählbarer Passworte

Insgesamt fielen 86% aller Passworte in eine der in Tabelle 1.1 angegebenen Kategorien und waren somit sehr einfach zu brechen. Aufgrund der aus dieser speziellen Studie und in der IT-Welt gesammelten Erfahrungen gelten Passwort-basierte Verfahren als ziemlich unsicher.

1.5.2 Challenge-Response-Verfahren

Challenge-Response- (CR) oder auf deutsch Frage-Antwort-Verfahren sind Verallgemeinerungen von Authentifizierungsverfahren, die auf Wissen beruhen. Die wesentliche Idee, dass die Fragen und die passenden Antworten zwar auf einem sich nicht ändernden Geheimnis basieren, aber bei jedem Anmeldeversuch neu generiert werden, so dass einem Angreifer ein Ausspähen erschwert wird.

Im Folgenden wird ein allgemeiner CR-Ablauf basierend auf symmetrischen Verschlüsselungsverfahren erklärt. Es wird ein Szenario betrachtet, in dem eine Chipkarte sich gegenüber einem Arbeitsplatzrechner authentifizieren möchte. Dies lässt sich natürlich verallgemeinern, so dass anstelle der Chipkarte auch ein beliebiger Clientrechner und als authentifizierende Instanz ein Server in einer Client-Server-Architektur betrachtet werden kann.

CR mit symmetrischer Verschlüsselung

Voraussetzung für die Durchführung einer Authentifikation ist, dass die beteiligten Geräte den gleichen Verschlüsselungsalgorithmus E sowie den gleichen Schlüssel verwenden, der z.B. aus dem Passwort des Benutzers abgeleitet sein kann. Sei CID (*Card Identification*) die Kartenidentifikation der Karte und K_{CID} der verwendete Schlüssel, der auf der Karte gespeichert ist. Sei K_r der Schlüssel, den der Rechner lokal zur Kommunikation mit der Karte CID in einer Schlüsseldatenbank gespeichert hat. Ist die Karte authentisch, so gilt $K_{CID} = K_r$.

Beim Login muss der Benutzer seine Chipkarte in ein an den Arbeitsplatzrechner angeschlossenes Lesegerät legen, so dass die Kartenkennung CID ausgelesen werden kann. Der Rechner sucht den zu CID in seiner Schlüsseldatenbank abgelegten Schlüssel K_r. Falls ein Eintrag vorliegt, stellt der Rechner der zu authentifizierenden Chipkarte eine Frage (*challenge*), indem er ihr eine neu erzeugte Zufallszahl $RAND$ zusendet. Die Chipkarte muss $RAND$ mit dem vereinbarten Verfahren E und ihrem Schlüssel K_{CID} verschlüsseln. *response*: $C = \{RAND\}^{K_{CID}}$.

Das Ergebnis C dieser Verschlüsselung wird an den Rechner gesendet (*response*). Dieser verschlüsselt seinerseits die Zufallszahl $RAND$ mit dem ihm bekannten Schlüssel K_r = K_{CID}. Test: $C' = \{RAND\}^{K_r}$.

Die Chipkarte hat die *challenge* korrekt beantwortet, wenn es dem Arbeitsplatzrechner gelingt, die Gleichheit zwischen seinem Ergebnis C' und der von der Chipkarte gelieferten Kryptonachricht C festzustellen. Die Schritte sind nachfolgend noch einmal zusammengefasst. Da der Server in der Regel die Kryptonachricht C nicht entschlüsselt, um die Authentizität zu prüfen, werden an Stelle von symmetrischen Kryptoverfahren häufig auch *Message-Authentication-Code-Verfahren (MAC-Verfahren)* verwendet. Ein MAC ist ähnlich dem in Abschnitt 1.4.3.2 beschriebenen Hashwert mit dem Unterschied, dass in die Berechnung eines MAC ein geheimer Schlüssel einfließt. Da der geheime Schlüssel nur den Kommunikationspartnern bekannt ist, kann ein MAC nur von diesen auch erzeugt werden. Somit ermöglicht ein MAC Aussagen über die Authentizität des Urhebers der Daten.

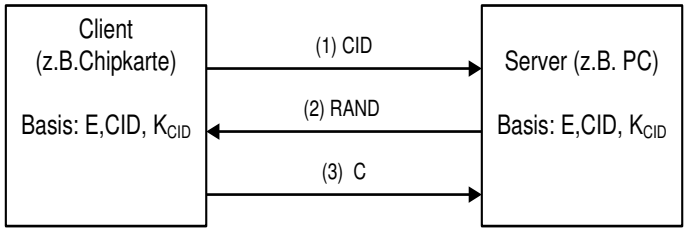

Abbildung 1.20: Ablauf eines symmetrischen Challenge-Response-Verfahrens

Der skizzierte Ablauf in Abbildung 1.20 kann noch erweitert und zur wechselseitigen Authentifikation eingesetzt werden, so dass sich der Server dann auch gegenüber dem Client authentifiziert. In diesem Fall stellt der Client die *challenge*, die vom Server korrekt zu beantworten ist.

Sicherheit

Beachtenswert an dem vorgestellten Verfahren ist, dass das Passwort des Benutzers zur Authentifikation nicht übermittelt werden muss. Da bei jeder Authentifizierung eine andere Zufallszahl $RAND$ verwendet wird, kann ein Angreifer einen abgefangenen und bereits zur Authentifikation verwendeten Kryptotext $\{RAND\}^{K_{CID}}$ nicht erfolgreich wieder einspielen, um sich als

CID zu maskieren. Es besteht aber die Gefahr, dass ein Angreifer die Verbindung zwischen dem zu authentifizierenden und dem authentifizierenden System kontrolliert. Da symmetrische CR-Verfahren heutzutage vielfach auch in offenen Umgebungen eingesetzt werden, erfolgt der Austausch von *challenges* und zugehöriger *responses* häufig über unsichere und sehr einfach abzuhörende Funkschnittstellen. Dies eröffnet vielfältige Möglichkeiten für Known-Plaintext-Angriffe. Hierbei zeichnet der Angreifer die Klartext-Nachricht $RAND$ sowie den zugehörigen Kryptotext, die *response*, auf und versucht mittels kryptoanalytischer Techniken den verwendeten geheimen Schlüssel abzuleiten.

Mit den abgehörten Klartext-Kryptotext-Paaren lassen sich aber auch Wörterbuchangriffe durchführen, wie man sie in herkömmlichen Passwort-Systemen kennt. Dazu rät der Angreifer den verwendeten geheimen Schlüssel, verschlüsselt den $RAND$-Wert mit diesem geratenen Schlüssel und vergleicht sein Ergebnis mit der *response* des berechtigten Benutzers. Zur Abwehr derartiger Angriffe kann man zum Beispiel die Protokolle erweitern, so dass die *challenge* verschlüsselt übermittelt wird.

1.5.3 Smartcard

Besitzbasierte Authentifikationstechniken stützen sich in der Praxis meist auf den Besitz von Smartcards ab, wobei zunehmend aber auch andere Formfaktoren wie USB-Tokens eingesetzt werden. In diesem Abschnitt wird nur auf Smartcards eingegangen, da die meisten Smartcard-Konzepte, auch auf andere Formfaktoren übertragbar sind.

Als Geburtsstunde der Chipkarte gilt das Jahr 1974, in welchem dem französischen Wirtschaftsjournalisten R. Moreno ein Patent für ein „System zur Speicherung von Daten in einem unabhängigen, tragbaren Gegenstand“ erteilt wurde. Die heutige Chipkarte ist eine Plastikkarte, die in unterschiedlichen Ausprägungen vorkommt. Die einfachsten Typen sind reine Speicherkarten, besitzen also keine CPU und sind monofunktional. Derartige Karten können sehr preiswert produziert werden und sind nicht wirklich „smart“. Beispiele für Speicherkarten sind Telefon- oder Krankenversicherungskarten [4].

Die nächste Stufe bilden die intelligenten Speicherkarten, die eine zusätzliche Sicherheitslogik besitzen, die meist für eine *Personal Identification Number (PIN)*-Speicherung und -Überprüfung verwendet wird. Weiterhin verfügen sie in der Regel über einen Zähler für Fehlversuche, um nach einer Anzahl solcher Fehlversuche die Karte zu sperren. Sind die Karten auch noch mit einem eigenen Mikroprozessor und einem programmierbaren Speicher ausgerüstet, dann spricht man wirklich von Smartcards. Ein Beispiel für solche Smartcards ist die JavaCard. Diese Smartcards dienen nicht nur als sicherer Information- oder Schlüsselspeicher, sie bieten zudem verschiedene Sicherheitsdienste wie Authentifikation, Verschüsselung, Signatur etc. an, die in einer vertrauenswürdigen Umgebung genutzt werden können. Die Möglichkeit eine Signatur auf der Smartcard zu erzeugen, bringt einen großen Vorteil, da die Schlüssel nie die Smartcard verlassen. Dadurch ist das Erspähen des privaten Schlüssel praktisch unmöglich.

Durch das schnell sich verbessernde Preis/Leistungsverhältnis von Smartcards kommt diesen bereits heute eine große Bedeutung als Sicherheitswerkzeug und insbesondere als Mittel zur Authentifikation zu. Smartcards sind in vielfältigen Bereichen einsetzbar, als Kreditkarte im Bankensektor, als Campus- und Firmen-Karte oder als SIM-Karte in Mobiltelefonen.

[4] www.die-gesundheitskarte.de

Smartcardbasierte Authentifikation

Eine Smartcard kann sehr gut als persönliches Sicherheitswerkzeug zur Authentifikation von Benutzern eingesetzt werden. Es wird zwischen drei Stufen der Authentifizierung unterscheiden. Im ersten Schritt authentifiziert sich der Benutzer gegenüber der Smartcard und erst im zweiten Schritt erfolgt die Authentifikation mit dem Zielsystem wie dem Kartenlesegerät oder einem PC. Der dritte Schritt umfasst die Authentifikation des Zielsystems gegenüber der Karte.

Schritt 1: Benutzerauthentifikation

Der Benutzer authentifiziert sich gegenüber der Karte in der Regel unter Nutzung einer vier- oder fünfstelligen PIN. Im Umfeld der Smartcards wird eine PIN auch häufig als *Card Holder Verification (CHV)* bezeichnet. Die PIN gibt der Nutzer am Kartenlesegerät ein. Wird die PIN über ein Kartenlesegerät an die Karte transferiert, so sendet das Lesegerät gleichzeitig das Kommando Verify CHV, um die Karte zur Authentifikation zu veranlassen. Die eingegebene PIN wird vom Mikroprozessor der Karte mit der dort abgespeichert PIN verglichen (vgl. Abbildung 1.21).

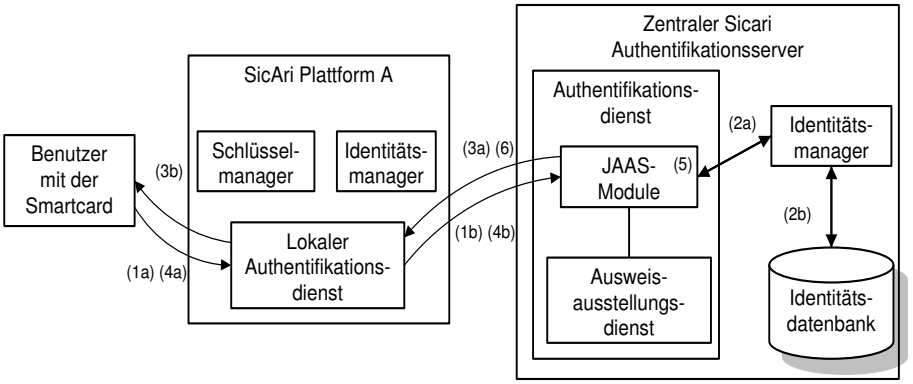

Abbildung 1.21: Authentifikation zwischen Nutzer, Kartenterminal und Karte

Der Chip führt zudem einen Zähler über die Anzahl der Fehlversuche. Falls eine kartenspezifisch festgelegte Maximalzahl erreicht ist, üblicherweise ist dies nach drei Fehlversuchen der Fall, wird die Karte blockiert. Manche Karten ermöglichen das Entsperren durch die Eingabe einer speziellen PIN, des PIN Unblocking Key (PUK). Für dessen Eingabe wird ebenfalls ein Zähler geführt. Mehrmalige Fehlversuche führen zu einer endgültigen Sperrung der Karte. Im nächsten Schritt authentifiziert sich nun die Karte gegenüber dem Lesegerät. Dazu besitzen beide ein Pre-Shared Seceret K. Das Terminal sendet der Karte eine $RAND$ und das Kommando, eine entsprechende *response* zu berechnen. Die Karte antwortet mit der verschlüsselten *response*, wobei zur Verschlüsselung die Kenntnis von K nachgewiesen wird.

Alternativ zur PIN-Authentifikation werden in zunehmendem Maß auch biometrische Informationen zum Identitätsnachweis verwendet. Da biometrische Techniken (siehe nächsten Abschnitt) spezielle Hardwarekomponenten wie beispielsweise Fingerabdruck-Scanner zur Erfassung der biometrischen Merkmale erfordern, enthalten die Smartcards häufig nur die Referenzwerte der zu identifizierenden Person. Der Abgleich zwischen Referenzdaten und aktuellen Daten erfolgt im Gegensatz zur PIN-Überprüfung häufig nicht in der Smartcard, sondern durch das Biometriegerät. Dies kann für die Sicherheit gravierende Konsequenzen haben. Musste man bei der PIN-Technik „nur“ darauf vertrauen, dass die Chipkarte zuverlässig und vor Angriffen

geschützt (*tamper resistant*) ihre Funktionen erfüllt, so fordert man diese Vertrauenswürdigkeit nun zusätzlich von den verwendeten biometrischen Geräten. Auf Risiken im Zusammenhang mit biometrischen Authentifikationstechniken gehe ich im nächsten Abschnitt näher ein.

Schritte 2 und 3: Karten-Zielsystem-Authentifikation

Im zweiten Schritt erfolgt die Authentifikation zwischen der Smartcard und dem Zielsystem. Dabei werden CR-Protokolle verwendet, wobei als symmetrisches Verfahren meist DES bzw. Tripel-DES und als asymmetrisches meist das RSA-Verfahren mit typischen Schlüssellängen von 512, 768 oder 1024 Bit eingesetzt wird. CR-Protokolle wurden bereits in Abschnitt 1.5.2 ausführlich erklärt. Soll sich auch das Zielsystem gegenüber der Karte authentifizieren, so wird dazu wiederum ein CR-Protokoll abgewickelt, wobei diesmal die *challenge* durch die Chipkarte gestellt wird.

Zusammengefasst stellt der Einsatz von Smartcards als Sicherheitswerkzeuge einen wesentlichen Beitrag zur Erhöhung der Systemsicherheit dar. Das immer günstiger werdende Preis/-Leistungsverhältnis, sowohl der Karten selber, als auch der zu ihrer Betreibung erforderlichen Lesegeräte, macht diese Technologie nun auch dem Massenmarkt zugänglich. Mit der Entwicklung der JavaCard wurde ein weiterer wichtiger Schritt in Richtung auf einen breiten Einsatz von Smartcards gemacht. Auf die Sicherheitseigenschaften der Programmiersprache Java selber sowie ihrer Ausführungsumgebung, der Java Virtual Machine (JVM), wird an dieser Stelle nicht weiter eingegangen.

Mit der zu erwartenden Verbesserung der Prozessorleistung und einer Vergrößerung der Hardwarekomponenten, so dass z.B. auch RSA-Schlüssel von 2048 Bit verwendbar sind, zusammen mit einer Ablösung der PIN-Überprüfung durch biometrische Techniken, eröffnet die Smartcard äußerst interessante Zukunftsperspektiven zur Steigerung der Systemsicherheit von IT-Systemen. Es sei aber noch einmal darauf hingewiesen, dass die Smartcards keine „Allheilmittel“ zur Lösung von Sicherheitsproblemen darstellen. Sie sind wichtige Bausteine einer Sicherheitskette, deren Sicherheit jedoch, von der Sicherheit ihres schwächsten Glieds abhängt.

1.5.4 Biometrie

In diesem Abschnitt wird auf Authentifikationstechniken eingegangen, die auf biometrischen Merkmalen (griechisch bios = Leben und metron = Mass) basieren. Unter einem solchen Merkmal versteht man physiologische oder verhaltenstypische Eigenschaften einer Person, die diese eindeutig charakterisieren wie z.B. Fingerabdrücke.

Einführung

Der Einsatzbereich biometrischer Technologie ist weit gefächert und reicht vom Bereich der Strafverfolgung, der Grenzkontrollen und der Terrorismusfahndung, über Eintrittskontrollen für Hochsicherheitsbereiche bis hin zur Kontrolle von Zugriffen auf Datenbestände, der Initiierung von Transaktionen (beispielsweise an einem Geldautomat), oder zur Aktivierung von Smartcards und Handys anstelle einer herkömmlichen PIN-Authentifikation.

Techniken zur eindeutigen Identifikation von Personen anhand biometrischer Merkmale haben bereits eine längere Entwicklungsgeschichte hinter sich. Ihre breite Akzeptanz im Massenmarkt wurde jedoch über lange Zeit durch die hohen Kosten, die mit der benötigten Geräteausstattung verknüpft waren, gebremst. Die Fortschritte bei der Entwicklung der Biometriegeräte sorgen aber mittlerweile dafür, dass diese Technologie auch beginnt, im Massenmarkt Fuß zu fassen.

Biometrische Techniken zur Authentifikation und Zugangskontrolle kann man grob in zwei Klassen einteilen. Man unterscheidet Verfahren, die auf physiologischen, statischen Eigenschaften einer Person beruhen und verhaltenstypische, dynamische Verfahren. Beispiele für physiologische Eigenschaften sind die Iris bzw. die Retina, das Gesicht oder Fingerabdrücke. In der Forschung werden bereits weitere physiologische Eigenschaften wie der Geruch oder der Hautwiderstand untersucht.

Verhaltenstypisch ist beispielsweise ein spezifisches Tippverhalten (z.B. Rhythmus), die Stimme oder die Dynamik einer handschriftlichen Unterschrift. Noch im Forschungsstadium befinden sich Untersuchungen anderer Verhaltensmuster wie beispielsweise das Sitzverhalten einer Person. In der Praxis sind heute noch überwiegend Geräte im Einsatz, die sich auf physiologische Merkmale beziehen. Im Zusammenhang mit kleinen mobilen Endgeräten wie PDAs oder Smartphones werden aber zunehmend auch handschriftliche Unterschriften oder auch die Stimme (multimodale Ein-/Ausgabeschnittstellen stehen z.B. bei Smartphones bereits zur Verfügung) als biometrische Erkennungsverfahren eingesetzt, da diese Geräte in der Regel über multimodale Benutzungsschnittstellen verfügen und bereits Techniken zur Schrift- oder auch Spracherkennung anbieten.

An ein biometrisches Merkmal, das zur Authentifikation von natürlichen Personen eingesetzt werden kann, werden folgende allgemeine Anforderungen gestellt.

- Universalität: Jede Person besitzt das biometrische Merkmal.
- Eindeutigkeit: Das biometrische Merkmal ist für jede Person verschieden.
- Beständigkeit: Das Merkmal ist unveränderlich.
- Quantitative Erfassbarkeit: Das Merkmal lässt sich mittels Sensoren quantitativ erfassen.
- Performance: Die Erfassung des Merkmals kann mit einer erforderlichen Genauigkeit erfolgen und der Vorgang ist leistungsfähig durchführbar.
- Akzeptanz: Die Verwendung des Merkmals wird von Benutzern akzeptiert.
- Fälschungssicherheit: Das Merkmal ist fälschungssicher.

Die körpereigenen Merkmale erfüllen die aufgelisteten Anforderungen in unterschiedlichem Maß, so dass die Qualität eines biometrischen Verfahrens auch von der Qualität abhängt, mit der die Anforderungen erfüllt werden.

Biometrische Authentifikation

Biometrische Techniken arbeiten alle nach dem gleichen Schema. Zunächst muss das Analysesystem Referenzwerte der zu analysierenden biometrischen Eigenschaften mittels Sensoren erfassen und digitalisieren. Diese Referenzwerterfassung wird auch als Lernen bezeichnet. Aus den Referenzwerten werden dann charakteristische Eigenschaften extrahiert und als personenbezogene, komprimierte Referenzdatensätze auf dem biometrischen Gerät selber, auf einer personenbezogenen Smartcard oder in anderen Speichermedien abgelegt. Zur Authentifikation einer Person benötigt man spezielle Geräte (u.a. Fingerabdrucksensoren oder Videokameras), die das betreffende biometrische Merkmal bei jeder durchzuführenden Authentifikation erfassen. Die aktuell erhobenen biometrischen Merkmale sind ihrerseits in Merkmalssätze digital zu transformieren und im abschließenden Schritt mit den gespeicherten Referenzdaten zu vergleichen.

Die datengesetzkonforme Verwaltung der Referenzdaten durch zuverlässige und vertrauenswürdige Systeme ist eine wesentliche Voraussetzung dafür, dass biometrische Techniken von Benutzern akzeptiert und eingesetzt werden. Biometrische Verfahren können dann erheblich zu einer Qualitätssteigerung auch in offenen IT-Systemen beitragen, wenn die Integrität und Authentizität der Referenzdaten mit hoher Zuverlässigkeit garantiert wird und Biometrie nicht als „Allzweckwaffe“ und Kryptographieersatz zweckentfremdet wird. Genauso, wie man auch nicht das gleiche Passwort für zwei unterschiedliche Rechner oder den gleichen Datenschlüssel für unterschiedliche Anwendungen verwenden sollte, so sollte auch ein und dasselbe biometrische Merkmal nicht zur Zugangskontrolle für ganz unterschiedliche Anwendungsbereiche wie private Daten, E-Mails und vertrauliche Patientenakten oder gleichzeitig zum Deaktivieren der Autowegfahrsperre und zum Aktivieren des Handys sowie der ec-Karte genutzt werden.

Wie auch schon an anderen Stellen ist auch hier natürlich wieder darauf hinzuweisen, dass die Eignung von Verfahren und Mechanismen unmittelbar davon abhängt, welche Sicherheitsanforderungen gestellt werden. Nicht für jeden Anwendungsbereich ist eine hohe Mechanismenstärke der biometrischen Verfahren notwendig. Trotz der zurzeit noch offensichtlichen Defizite sind diese aber durchaus für einige Bereiche sinnvoll einsetzbar.

Die Situation stellt sich jedoch anders dar, wenn beispielsweise biometrische Überprüfungen rechtsverbindlichen Charakter haben sollen. Aufgrund der fehlenden Standardisierung existieren neben verschiedener herstellerspezifischer Verfahren zur Repräsentation der Charakteristika biometrischer Merkmale auch proprietäre, zum Teil nicht offen gelegte Verfahren zum Abgleich von Referenz- und Verifikationsdaten. Welche Ergebnisse derartige Verfahren bei unterschiedlichen Repräsentationen ein und desselben Merkmals liefern, welche Rate an fälschlich akzeptierten Benutzern (*false positive*) oder an fälschlich abgelehnten berechtigten Benutzern (*false negative*) aufgrund derartiger Interoperabilitätsprobleme auftreten können, ist zurzeit noch völlig ungeklärt. Der Einsatz biometrischer Authentifizierungstechniken zum Beispiel als integraler Bestandteil von Personalausweisen über Ländergrenzen hinweg, wie er zurzeit insbesondere durch die USA sehr stark vorangetrieben wird, ist vor dem Hintergrund einer fehlenden Standardisierung deshalb noch sehr problematisch. Weiterhin sollte nicht vergessen werden, dass die Anzahl der biometrischen Merkmale eines Menschens beschränkt ist und sollte ein benutztes Merkmal wie z.B. der Fingerabdruck kompromittiert werden, ist es nicht möglich, dieses wie z.B. ein Passwort auszuwechseln.

1.5.5 PAM und die Authentifikation in UNIX

PAM steht für *Pluggable Authentication Modules* und wurde ursprünglich von der Firma Sun für ihr UNIX-Derivat Sun Solaris entwickelt. Der Name deutet dabei schon auf die Möglichkeiten zur Verwendung von unterschiedlichen Modulen (*Modules*) zur Authentifikation (Authentication) und eine hohe Konfigurierbarkeit (Pluggable) der Gesamtlösung durch den System-Administrator an. 1995 wurde das Konzept in der RFC 86.0 der Open Software Foundation (OSF) veröffentlicht [44, 30]. Seitdem wird diese Technologie schrittweise auch in anderen UNIX-Systemen eingeführt (beispielsweise in Hewlett-Packard’s HP-UX seit Version 10.20). Schon seit geraumer Zeit gibt es das Projekt Linux-PAM, das sich zur Aufgabe gemacht hat, ein freies PAM speziell für Linux aber auch für andere Systeme zu entwickeln. Ergebnisse liegen schon vor und sind auch schon in die meisten Linux-Distributionen eingeflossen (z.B. SuSE-Linux ab Version 6.1). Mit der Linux-üblichen Dynamik sind bereits sehr viele Module z.B. LDAP-Modul oder Kerberos-Modul und PAM-fähige Anwendungen verfügbar.

Das Problem in den existierenden UNIX-Systemen (aber auch in anderen Systemen) ohne PAM ist, dass es mehrere Möglichkeiten gibt, sich den Zugang zum System zu verschaffen, wie z.B. via ftp-*daemon*und der telnet-*daemon*. Diese sogenannten *system-entry* Services implementieren die Authentifikationsmechanismen selbst. Normalerweise handelt es sich um eine Abfrage des Namens und Passwortes. Es ist wohl offensichtlich, dass bei mehrfacher Implementierung der gleichen Funktionalität auch die Fehleranfälligkeit steigt und außerdem gibt es Probleme, wenn der zugrunde liegende Authenifizierungmechanismus geändert werden soll. Dann ist normalerweise ein Austausch der Applikation bzw. eine Änderung des Quellcodes, ein Neuübersetzen und Installieren notwendig.

PAM behebt die beschriebenen Probleme, indem es die einzelnen Anwendungen von den Mechanismen zur Benutzerauthentifizierung trennt. Das PAM-Framework besteht aus einer Menge von gemeinsam genutzten Bibliotheken (*shared libraries*). Diese Module, die jeweils einen bestimmten Mechanismus implementieren, liegen ebenfalls als *shared libraries* vor. Der Administrator kann nun (abhängig vom Sicherheitsbedürfnis, der Umgebung, den persönlichen Vorlieben etc.) für jede einzelne Anwendung festlegen, welche Mechanismen zum Einsatz kommen sollen (d.h. also, welche Module verwendet werden). Dabei können auch mehrere Module hintereinander zum Einsatz kommen (*module stacking*) und durch die Verwendung mehrerer Authentifizierungsmechanismen ein einheitliches Login für den Benutzer erreicht werden.

Wie Module, Framework und Applikationen zusammenarbeiten verdeutlicht noch einmal Abbildung 1.22.

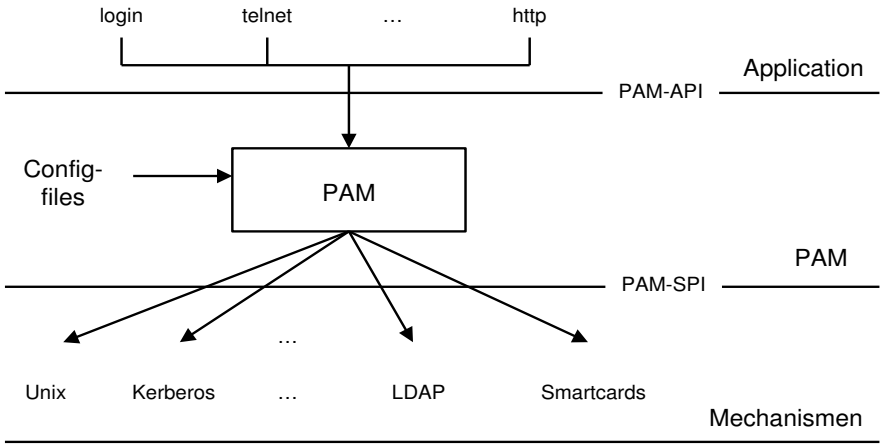

Abbildung 1.22: PAM-Architektur

1.6 Authentifikationsprotokolle

Die im vorherigen Abschnitt eingeführten Authentifizierungstechniken können sowohl alleine als auch in unterschiedlichen Ausprägungen und Kombinationen eingesetzt werden, um Subjekte in offenen Systemumgebungen zu authentifizieren. Die jeweiligen Maßnahmen sind Bestandteil von Authentifikationsprotokollen, die zwischen räumlich verteilten Endsystemen abgewickelt werden [12, 49].

Heutige vernetzte IT-Systeme sind noch vorwiegend durch Client-Server-Architekturen geprägt. Die Server bieten Dienste bzw. Services an, die von den Clients in Anspruch genommen werden können. Beispiele für solche Dienste sind Dateisystem- oder Datenbankdienste. Die Kommunikation bzw. Kooperation zwischen Client- und Serverprozessen erfolgt nachrichtenbasiert meist unter Nutzung des Remote-Procedure-Call-Konzepts (RPC-Konzepts). Im Folgenden wird zunächst mit dem HTTP-Protokoll ein weit verbreitetes, nachrichtenorientiertes Authentifizierungsprotokoll erklärt. Danach wird das RPC-Konzept im Groben eingeführt, so dass die erforderlichen sicherheitsspezifischen Erweiterungen des Basisprotokolls verständlich werden.

1.6.1 HyperText Transport Protocol (HTTP)

Dem Client wird es in HTTP ermöglicht, sich durch einen Header beim Server zu identifizieren. Im Folgenden wird kurz auf den Nachrichtenaustausch eingegangen. Zunächst sendet der Client eine Anfrage ohne *Authorization-Header*. Wenn die entsprechende Seite durch ein Passwort geschützt ist, schickt der Server eine Antwort, mit HTTP-Statuscode 401 und einem WWW-Authenticate-Header. Bei diesem Statuscode handelt es sich um einen Fehlercode, welcher besagt, dass der Client nicht die benötigte Berechtigung verfügt. Der Header teilt dem Client mit, für welchen Bereich (*realm*) er sich identifizieren muss, um an das entsprechende Dokument zu gelangen und mittels welchen Authentifikationsmechanismus er sich anmelden muss (normalerweise mit Benutzername und Passwort). Bei der nächsten Anfrage fügt der Client die benötigten Daten hinzu. Dadurch kann er als berechtigter Nutzer identifiziert werden und die entsprechenden Daten werden vom Server geschickt.

Das HTTP-Protokoll spezifiziert zwei verschiedene Authentifizierungsschemata: *Basic Authentication* und *Digest Authentication*, die in den nächsten Unterabschnitten erklärt werden [15].

Vorraussetzung für die beiden Methoden ist, dass der Server und Nutzer ein vorab vereinbartes Geheimnis (*preshared key*) in Form eines Passwortes besitzen.

Basic Authentication

Wenn in der Antwort des Servers in dem *WWW-Authenticate-Header* der Wert `Basic` steht, dann wird das *Basic Authenticate Schema* angewandt. Dieses Schema basiert darauf, dass der Nutzer sich für jeden Bereich mit Benutzername und Passwort authentifiziert. Der Benutzername und das Passwort werden über das Netz im Klartext verschickt. Genauer gesagt werden sie getrennt durch einen Doppelpunkt „**:**“ aneinandergehängt und dann mit `base64` kodiert. Obwohl es keinerlei Verschlüsselung gibt, kann *Basic Authentication* trotzdem sinnvoll eingesetzt werden, wie z.B. wenn es sich um Daten handelt, die zwar nicht der Öffentlichkeit zugänglich sein sollen, aber es sich dennoch nicht um wichtiges Datenmaterial handelt oder wenn die Registrierung der Nutzer anderen, nicht-sicherheitsbezogenen Zwecken dient.

Digest Authentication

Digest Authentication wurde entwickelt, um *Basic Authentication* zu ersetzen. Es ist sicherer und kompatibel zu *Basic Authentication*, dennoch bietet diese Methode bei weitem keine vollständige Lösung zur Authentifikation der Nutzer, weil keine kryptographische Mechanismen verwendet werden.

Wie die *Basic Authentication*, so basiert auch *Digest Authentication* auf dem einfachen CR-Mechanismus. Die *challenge* besteht aus einer *nonce (a number used once)*.

Die gültige Standard-Antwort (MD5 Verwendung) sieht wie folgt aus:

$$response = MD5(Zufallszahl, Name, Passwort, http - Methode, URI \ \ des \ \ Servers)$$

Bei dieser Vorgehensweise wird das Passwort nie im Klartext über das Netz geschickt, was den großen, aber auch den einzigen Vorteil gegenüber der *Basic Authentication* darstellt.

1.6.2 Remote Procedure Call (RPC)

Der RPC ist im Prinzip eine Übertragung des Prozedurkonzepts sequentieller Programmiersprachen auf eine verteilte Umgebung, indem der Client analog zu einem lokalen Prozeduraufruf einen entfernten Aufruf durchführt [48]. Dazu sendet er eine Anfragenachricht an den Server und wartet solange (synchron), bis dieser die Anfrage bearbeitet hat und seine Antwortnachricht zurücksendet. Damit der Datentransport, der über ein zugrunde liegendes Netzwerk erfolgt, weitestgehend transparent für den Anwendungsprogrammierer bleibt, werden so genannte *Stub*-Prozeduren verwendet. Dabei handelt es sich um Bibliotheksfunktionen, die sowohl dem Client- als auch dem Servercode hinzuzufügen sind. Zu den Aufgaben der Stubs gehören die Lokalisierung des Servers, der die gewünschte Prozedur implementiert, die Transformation eines Aufrufs in eine übertragbare Nachricht (*marshalling*) sowie das Versenden der Nachricht unter Nutzung der Protokolle eines Transportsystems (vgl. Abbildung 1.23).

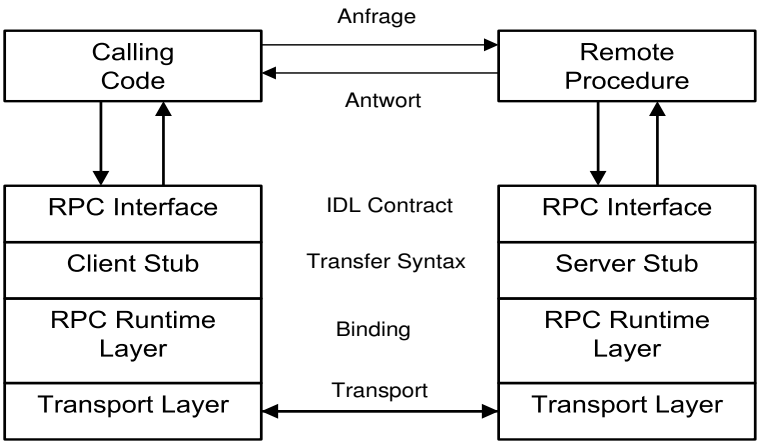

Abbildung 1.23: Remote Procedure Call

Zur automatischen Generierung von Stub-Prozeduren stellen heutige RPC-Realisierungen Schnittstellenspezifikationssprachen (*Interface Definition Language (IDL)*) wie z.B. WSDL sowie IDL-Compiler zur Verfügung. Die über einen entfernten Prozeduraufruf nutzbaren Dienste eines Servers sind mit den IDL-Sprachkonzepten zu spezifizieren, d.h. der Dienst ist zu benennen und die benötigten Parameter einschließlich ihrer Typen sind anzugeben. Aus dieser Schnittstellenspezifikation generiert der IDL-Compiler die erforderlichen Client- und Server-Stubs.

Der RPC-Mechanismus ist auf der Anwendungsschicht des ISO/OSI Referenzmodells anzusiedeln und setzt somit auf den darunter liegenden unsicheren Transport- und Kommunikationsprotokollen auf. Das bedeutet, dass die Anfrage- und Antwortnachrichten ungeschützt übertragen werden. Darüber hinaus findet auch keine Authentifikation zwischen Client und Server statt. Für

den zweiten Problembereich, also die Authentifikation, bietet der Secure RPC eine Lösung, das eine Erweiterung des Original-Protokolls der Firma Sun ist.

1.6.2.1 Secure RPC

Secure RPC ist eine Entwicklung der Firma Sun und wurde erstmals in der Version SunOS 4.0 in Zusammenhang mit *Network File System (NFS)/Network Information Service (NIS)* eingesetzt. Er ist nun Bestandteil des von Sun zur Verfügung gestellten RPC-Dienstes. Diese RPC-Implementierung bietet dem Client im Wesentlichen drei Wahlmöglichkeiten, einen Server zu nutzen, nämlich `AUTH_NULL`, `AUTH_UNIX` und `AUTH_DES`. Bei der Wahl von `AUTH_NULL` erfolgen keinerlei Authentifikationsschritte und mit `AUTH_UNIX` wird lediglich eine Unix-artige Zugriffskontrolle ohne Authentifikation durchgeführt.

Erweiterte Authentifikationsmaßnahmen werden erst bei der Wahl von `AUTH_DES` eingesetzt. Wie der Name schon vermuten lässt, basieren die im RPC-Protokoll verwendeten Verschlüsselungen auf dem DES-Algorithmus. Der gemeinsame DES-Schlüssel $K_{c,s}$ wird unter Nutzung des *Diffie-Hellman-Verfahrens (DH-Verfahrens)* sowohl vom Client als auch vom Server berechnet [43]. Da im Secure RPC das für das DH-Verfahren benötigte Modul (die Primzahl q) einen festen Wert der Länge 192 Bit besitzt, haben der private und der öffentliche Schlüssel sowie der Schlüssel $K_{c,s}$ ebenfalls eine Länge von 192 Bit. Dieser gemeinsame Schlüssel wird jedoch nur verwendet, um Sitzungsschlüssel $K_{c,s}^{session}$ vertraulich vom Client zum Server zu übertragen, die für die Dauer einer Login-Sitzung eines Benutzers gültig sind.

Schlüsselmanagement

Zu klären bleibt, wie die benötigten öffentlichen Schlüssel des Clients und NIS-Servers wechselseitig bekannt gemacht werden. Ferner ist zu klären, wie der Clientrechner, der im Auftrag eines Benutzers tätig ist, in den Besitz des privaten Schlüssels dieses Benutzers gelangt, um damit die Schlüsselberechnung durchführen zu können. Da einem Benutzer nicht zuzumuten ist, sich seinen 192-stelligen privaten Schlüssel zu merken und ihn bei Bedarf dem Clientrechner zu präsentieren, werden die benötigten Informationen in der NIS-Datenbank verwaltet. Für jeden Benutzer existiert darin ein Eintrag, der dessen Domänennamen, den öffentlichen Schlüssel sowie dessen privaten Schlüssel enthält. Der private Schlüssel ist DES-verschlüsselt abgelegt, wobei der dafür notwendige Schlüssel aus dem Benutzerpasswort abgeleitet wird.

Beim Login eines Benutzers auf einem Clientrechner, auf dem Secure RPC Log installiert ist, wird über das Login-Programm automatisch der entsprechende NIS-Eintrag des Benutzers angefordert. Mit dem entweder interaktiv angeforderten oder auf dem Clientrechner bereits vorhandenen Benutzerpasswort ist der Client dann in der Lage, den in dem Eintrag enthaltenen privaten Benutzerschlüssel zu entschlüsseln. Zusammen mit dem öffentlichen Schlüssel des Servers, der auch aus der NIS-Datenbank zu entnehmen ist, kann er dann den gemeinsamen 192-Bit Schlüssel $K_{c,s}$ berechnen. Um diesen als DES-Schlüssel einsetzen zu können, muss er aber noch auf 56 Bit reduziert werden. Dies erfolgt auf einfache Weise, indem man vom 192-Bit Schlüssel $K_{c,s}$ nur die mittleren acht Byte nutzt und so den reduzierten Schlüssel $K_{c,s}^{red}$ erhält.

RPC-Protokollablauf

Eine RPC-Anfrage eines Clients C an einen Server S enthält folgende Daten:

$$\mathrm{C} \rightarrow \mathrm{S} : (C, \{K_{c,s}^{session}\}^{K_{c,s}^{red}}, \{L\}^{K_{c,s}^{session}}, \{T, L-1\}^{K_{c,s}^{session}})$$

$K_{c,s}^{session}$ ist der vom Client C erzeugte Sitzungsschlüssel, der mit dem gemeinsamen Schlüssel $K_{c,s}^{red}$ verschlüsselt wird. Damit der Server die Clientauthentizität überprüfen kann, schickt dieser ihm seine Identität C zusammen mit einem verschlüsselten Zeitstempel T sowie einer Gültigkeitsdauer L für den Zeitstempel. Mit dem dritten Teil der Anfragenachricht weist der Client dem Server nach, dass er tatsächlich den gemeinsamen Schlüssel $K_{c,s}^{session}$ kennt, da er nur mit dieser Kenntnis den korrekten Wert $L - 1$ angeben kann.

Der Server berechnet nach dem Empfang dieser Anfragenachricht seinerseits den Schlüssel $K_{c,s}^{red}$ und entschlüsselt damit den Sitzungsschlüssel $K_{c,s}^{session}$ Mit diesem kann er anschließend den Zeitstempel T und die um eins dekrementierte Lebensdauer $L - 1$ in der Teilnachricht

$$(\{T, L-1\}^{K_{c,s}^{session}})$$

überprüfen. Sind diese korrekt und gültig, so erzeugt S einen Nicknamen $N_{c,s}$ für C und sendet diesen zusammen mit dem Nachweis (der korrekt dekrementierte Zeitstempel T), dass er den Sitzungsschlüssel kennt, in der Antwortnachricht an den Client zurück:

$$\text{S} \rightarrow \text{C: } (\{T-1\}^{K_{c,s}^{session}}, N_{c,s})$$

Der Client kann zur schnelleren Authentifikation in nachfolgenden RPC-Anfragen den Nicknamen verwenden, da der Server die Authentifikationsdaten des Clients unter diesem Namen im Cache für eine gewisse Zeit ablegt.

$$\text{C} \rightarrow \text{S} : (\{T\}^{K_{c,s}^{session}}, N_{c,s})$$

Sicherheit

Ein wesentlicher Pluspunkt von Secure RPC ist, dass zur entfernten Authentifikation das Benutzerpasswort nicht über das unsichere Netz übertragen wird. Problematisch ist jedoch, dass in diesem Protokoll der Clientrechner und nicht eine vertrauenswürdige Instanz den zu verwendenden Sitzungsschlüssel $K_{c,s}^{session}$ generiert. Der Server vertraut darauf, dass der ihm präsentierte Schlüssel frisch ist. Das kann ein Angreifer ausnutzen, indem er die Anfragenachricht eines Opfers abfängt, sie kopiert und erst danach an den autorisierten Empfänger weiterleitet. Anschließend versucht er, den in der Kopie enthaltenen verschlüsselten Sitzungsschlüssel zu brechen. Gelingt ihm das, so kann er die Anfragenachricht wiedereinspielen und dem Server einen bereits gebrochenen Schlüssel zur Kommunikation unterschieben.

Da der Schlüssel $K_{c,s}^{session}$ nur mit einem 56-Bit DES-Schlüssel verschlüsselt wird, ist beim heutigen Stand der Technik ein solcher Angriff mit vertretbarem Aufwand durchführbar. Dies gilt ebenfalls für Angriffe, auf die in der NIS-Datenbank gespeicherten, sensitiven Informationen. Auch hier wird zur Gewährleistung der Vertraulichkeit nur ein DES-Schlüssel verwendet. Das Brechen dieses DES-Schlüssels hat aber noch gravierendere Konsequenzen, da dadurch auch der private Schlüssel des Benutzers offen gelegt ist. Damit kann ein Angreifer nicht nur eine Kommunikationsverbindung zwischen dem Benutzer und einem Server abhören, sondern es ist ihm sogar möglich, alle Secure RPC-basierten Kommunikationsverbindungen des Benutzers zu allen Servern S' abzuhören, da die Schlüssel $K_{c,s'}^{red}$ unter Kenntnis des privaten Benutzerschlüssels berechenbar sind.

Eine weitere Schwachstelle bleibt die nach wie vor vorhandene passwortbasierte Authentifikation eines Benutzers. Gelingt es einem Angreifer, ein solches Passwort erfolgreich zu brechen, so bietet der Secure RPC keinen weiteren Schutz mehr. Schließlich ist noch zu beachten, dass in dem Protokoll keine Maßnahmen zur Sicherstellung der Integrität von Nachrichten, also keine MACs oder Signaturen, vorgesehen sind.

Trotz dieser Problembereiche und Beschränkungen stellt der Secure RPC ohne Zweifel einen wichtigen Fortschritt gegenüber herkömmlichen RPC-Implementierungen dar. Weitere Verbesserungen ließen sich dadurch erzielen, dass Client und Server das zu verwendende Verschlüsselungsverfahren nach Bedarf aushandeln dürfen, die Sitzungsschlüssel durch zertifizierte, vertrauenswürdige Server erstellt werden und durch MACs die Integrität von Nachrichten geschützt, sowie durch die Verwendung digitaler Signaturen die Verbindlichkeit durchgeführter Aktionen gewährleistet wird. Ferner sollte auf die Speicherung des privaten Benutzerschlüssels in der NIS-Datenbank verzichtet werden, da diese in der jetzigen Form einen sehr interessanten Angriffspunkt bietet.

1.6.3 Andere Protokolle

Simple Authentication and Security Layer (SASL)

Authentifizierung ist wesentliche Grundfunktion einer Anwendung. Entsprechend oft muss sie *credentials* eines Benutzers nachfragen. Zieht man diese Funktion aus der Anwendung heraus, entsteht der Bedarf nach einer standardisierten Abstraktionsschicht und einem Verhandlungsmechanismus. Genau das bietet SASL, und zwar für alle verbindungsorientierten Protokolle wie beispielsweise das *Simple Mail Transfer Protocol (SMTP)*, *Internet Message Access Protocol (IMAP4)* und das *Lightweight Directory Access Protocol (LDAP v3)*. SASL gibt diesen verbindungsorientierten Protokollen die Möglichkeit, eine Authentifizierung anzufordern, nachdem beide Seiten über die Verwendung eines der zugelassenen Authentifizierungsmechanismen wie beispielsweise HTTP Digest, Kerberos v5 oder S/Key, bzw. SSL für externe Anfragen (immer häufiger auch der SSL-Nachfolger TLS) verhandelt und sich geeinigt haben. SASL ist in [34] beschrieben.

Transport Layer Security (TLS) / Secure Sockets Layer (SSL)

TLS arbeitet auf der Transportschicht des Protokollstacks und wird immer dann verwendet, wenn sich der Rechner oder die Geräte eines Benutzers mit einem Server im Internet verbindet, der eine sichere Verbindung anfordert. TLS/SSL nutzt ein Handshake-Verfahren, um den Server und optional auch den Client mit X.509 konformen Zertifikaten zu authentifizieren, die Verschlüsselungsform zu vereinbaren und um einen Sitzungsschlüssel auszutauschen. SSL (oder der Nachfolger TLS [10]) ist die am weitesten verbreitete Form der sicheren Kommunikation im Web. Dennoch gilt zu beachten, dass SSL v3 [16] (Client-Authentifizierung) in seiner heutigen Verwendungsform kein Authentifizierungsverfahren für den Benutzer des Clients ist, denn mit dem Client-Schlüssel auf dem lokalen Rechner wird lediglich der Rechner und dessen Software authentifiziert, nicht aber der Benutzer. SSL wird erst dann Teil eines Zwei-Faktoren-Authentifizierungsverfahrens, wenn der Client-Schlüssel außerhalb des Rechners auf einer PIN-gesicherten Smartcard liegt.

Secure Shell (SSH)

SSH ist ein sicheres Protokoll und ein Toolset für die Authentifizierung von Benutzern auf einem entfernten Server und für den Zugang zu diesem Server [52]. Eine sichere SSH-Verbindung zwischen einem Client und einem Server wird über einen Tunnel generiert, die an beiden Enden an einen festen Port geknüpft ist. Auf diese Weise können lokale Anwendungen entfernt genutzt werden.

Das SSH-Protokoll besteht aus 3 Komponenten:

- Das *Transport Layer Protocol* für die Server-Authentifizierung.
- Das User *Authentication Protocol* für die Client-Authentifizierung.
- Das *Connection Protocol* für die Gliederung des Kommunikationstunnels in Kanäle.

Ähnlich wie bei SSL/TLS setzt das *User Authentication Protocol* voraus, dass der Client im Besitz des öffentlichen Server-Schlüssels ist.

1.7 Single Sign On

Eine Einmalanmeldung bzw. *Single-Sign-On (SSO)* bedeutet, dass ein Benutzer nach einer einmaligen Authentifizierung auf alle Rechner und Dienste zugreifen kann, für die er berechtigt ist, ohne sich jedes Mal neu anmelden zu müssen. Ein Benutzer besitzt immer genau eine einzige Identität in der realen Welt. Innerhalb eines Systems kann er aber unter verschiedenen Benutzerkennungen oder ihm zugewiesenen Rollen gespeichert sein. Ziel vom SSO ist es, dass sich der Benutzer nur einmal unter Zuhilfenahme eines Authentifizierungsverfahrens (z.B. durch Passworteingabe) identifiziert. Danach übernimmt der SSO-Mechanismus die Aufgabe, den Anwender zu identifizieren [12, 18, 21, 19, 7].

Alle Lösungen haben eines gemeinsam, nämlich dass sie die Erstauthentifizierungsinformationen des Anwenders an die angeschlossenen Systeme weiterreichen müssen. Das muss in einer sicheren Weise geschehen: Ein SSO-Mechanismus wäre bedenklich, wenn er die Sicherheit des Erstauthentifizierungsverfahrens herabsetzen würde. Das naive Vorgehen, gar keine Authentifizierung des Benutzers durchzuführen, ist damit keine SSO-Lösung.

Vorteile von SSO sind Zeitersparnis, da nur noch eine einzige Authentifizierung notwendig ist, um auf alle Systeme zugreifen. Außerdem braucht man in diesem Modell nur einen Authentifizierungsserver oder -dienst, der für die Verwaltung der Nutzerdaten und Authentifizierung zuständig ist. Die Dienstserver, wie z.B. der E-Mail- oder online-shop-Server muss dann nur noch die Zugriffskontrolle durchführen und kann sich so die Zeit und Arbeit für Authentifizierung sparen. Ein möglicher Ablauf von SSO ist in Abbildung 1.24 dargestellt.

Nachteile sind aber auch vorhanden: Kann ein Angreifer die Identität eines Benutzers entwenden, so stehen ihm sofort alle Systeme zur Verfügung, auf die dieser Benutzer Zugriff hat.

Es wird zunächst beschrieben, welche Lösungsansätze in heutigen Softwaresystemen existieren, um einen SSO zu realisieren. Danach wird auf Mechanismen für einen SSO im Internet eingegangen. Auch die jeweiligen Sicherheitsrisiken der Mechanismen werden angesprochen. In einer Schlussbetrachtung werden die beschriebenen Mechanismen miteinander verglichen.

1.7.1 Kerberos-Authentifikationssystem

Das Kerberos-Authentifikationssystem wurde ursprünglich (Versionen 1-4) im Rahmen des Athena Projektes (Start 1983) am MIT in Kooperation mit IBM und DEC entwickelt [12, 6, 25]. Das Protokoll wurde nach dem Dreiköpfigen Hund Zerberus (lat. Cerberus) benannt, der gemäß

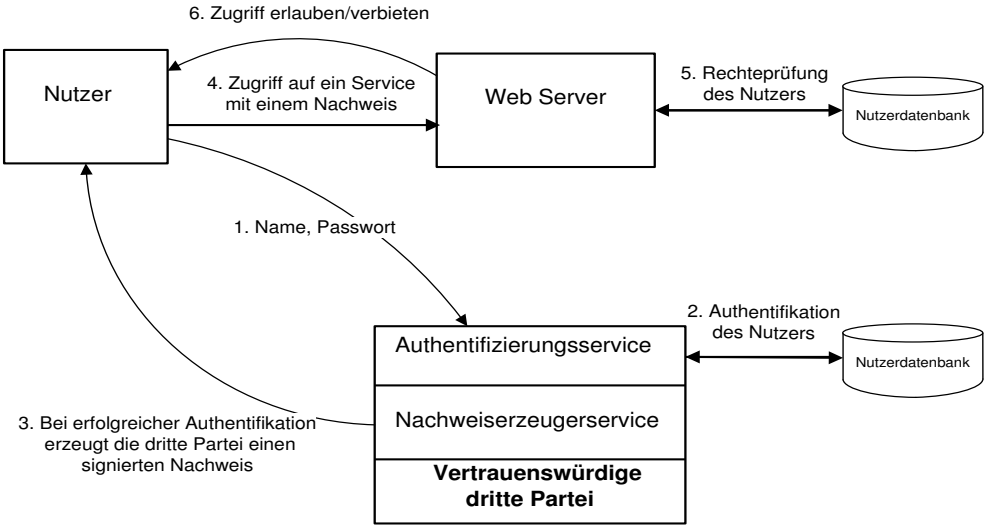

Abbildung 1.24: Ablauf von Single Sign On

der griechischen Mythologie den Eingang zur Unterwelt bewacht und jeden in die Unterwelt hinein- und niemanden mehr heraus lässt. Das Kerberos-Protokoll bewacht, im Gegensatz zum Mythos, den Eingang in das zu schützende System und kümmert sich nicht um dessen Verlassen. Auf die Verwendung von asymmetrischen Verfahren wurde verzichtet, da in den USA zur Zeit der Entwicklung von Kerberos die Public-Key-Verschlüsslung noch unter patentrechtlichen Beschränkungen lag. In einem Draft der IETF2 wurde aber bereits ein Konzept vorgeschlagen, um die Verwendung von asymmetrischen Verfahren mit Kerberos zu kombinieren. Dieses Konzept wurde von Microsoft für ihre proprietäre Kerberos-Version implementiert.

Eine Reihe von Mängeln, die noch bis einschließlich der Version 4 auftraten, sind in der aktuellen Version 5 beseitigt. Die verwendeten Verschlüsselungsverfahren sind nicht festgelegt und werden als zusätzliche Information der versandten Nachricht hinzugefügt. Außerdem können in Version 5 Authentifikationsinformationen an andere Subjekte weitergereicht werden, so dass ein Server mit den Authentifikationsmerkmalen eines Benutzers in dessen Auftrag (Delegations-Prinzip, siehe Abschnitt 1.9) tätig sein kann. Die Authentifikationsinformationen werden dezentral durch eine Hierarchie von Authentifikationsservern verwaltet, die jeweils in ihrem Verantwortungsbereich autonom agieren können und miteinander kooperieren, um bereichsübergreifende Zugriffe von Benutzern für diese transparent zu authentifizieren. Auf diese Weise wird ein SSO-Konzept realisiert, das verschiedene Verwaltungsbereiche überspannt. Das heißt, dass sich auch in einem verteilten System ein Benutzer nur einmal beim Zugang zu einem der vernetzten Rechner wie z.B. einem Datei-Server authentifizieren muss. Der authentifizierte Zugang zu anderen Rechnern des verteilten Systems wird automatisch, ohne weitere Interaktion mit dem Benutzer, durch die „kerborisierten“ Clients und Server abgewickelt.

Kerberos erfüllt zwei Aufgaben: Es stellt Hilfsmittel zur Verfügung, um (1) Subjekte (u.a. Arbeitsplatzrechner, Benutzer oder Server, sie heißen im Kerberos-Kontext Principals) zu authentifizieren und (2) Sitzungsschlüssel auszutauschen. Der Authentifikationsdienst wird von einem vertrauenswürdigen Server (*trusted third party*) erbracht. Der Authentifikationsdienst basiert auf symmetrischen Verfahren und geht davon aus, dass die beteiligten Principals ihre geheimen Schlüssel sicher verwalten. Da das Protokoll Zeitstempel mit einer globalen Zeit einsetzt, wird

darüber hinaus von den beteiligten Clientrechnern erwartet, dass deren Systemuhren *lose* mit den Uhren des Authentifizierungsdienstes sowie der Server, die von den Clients genutzt werden sollen, synchronisiert sind. Auf diese Weise wird sichergestellt, dass ihre Uhren sich nur um einen festgelegten Toleranzwert unterscheiden können.

Kerberos ist also ein reiner Authentifikations- und Schlüsselaustauschdienst ist. Die Vergabe von Rechten zur Nutzung von netzwerkweiten Diensten sowie die Kontrolle der Zugriffe gehört nicht zum Funktionsumfang.

Kerberos im Überblick

Kerberos basiert auf einem Client-Server-Modell (vgl. Abbildung 1.25), in dem ein Principal einen Dienst eines Servers S nur dann in Anspruch nehmen darf, wenn er dafür dem Server eine Authentizitätsbescheinigung (ein sogenanntes *ticket*) vorweisen kann, das ein vertrauenswürdiger Kerberos-Dienst ausgestellt hat. Kerberos ist zusammen mit herkömmlichen Netzwerkdiensten (z.B. *New Technology File System (NTFS)*, *Remote Login* oder *Electronic Mail*) verwendbar. Dazu müssen diese Dienste, d.h. die Server, die diese Dienste anbieten, um die nachfolgend beschriebenen Fähigkeiten zur Überprüfung von Authentizitätsnachweisen erweitert werden.

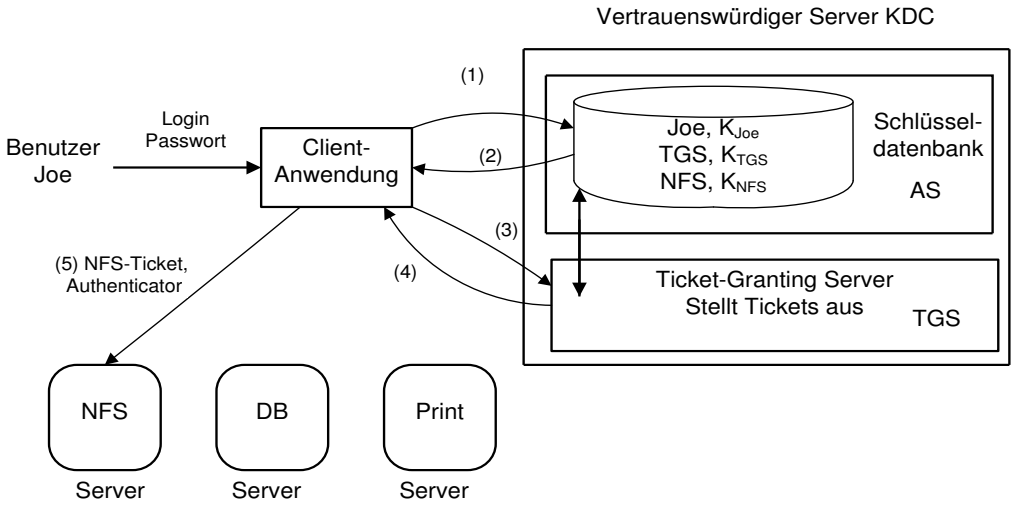

Abbildung 1.25: Kerberos-Grobarchitektur

Die Benutzer eines um Kerberosdienste erweiterten Systems melden sich bei ihrem Arbeitsplatzrechner (in Abbildung 1.25 ist das der Client) über die Vorlage eines Passwortes wie üblich an. Die für die entfernten Server-Zugriffe erforderlichen Authentizitätsbescheinigungen werden danach transparent für den Benutzer vom Clientrechner, beim vertrauenswürdigen *Authentifikationsserver (AS)* angefordert. Dieser arbeitet mit einem weiteren vertrauenswürdigen Server, dem *Ticket-Granting Server (TGS)*, zur Ausstellung der *tickets* zusammen. Die Dienste des AS und TGS werden häufig in einem Server zusammengefasst, der als *Key Distribution Center (KDC)* bezeichnet wird. In einem großen System können mehrere derartige Server vorhanden sein.

Jeder Principal A hat vorab mit dem Kerberos-Schlüsselverteilungs-Server KDC einen geheimen Master-Schlüssel (*master key*) K_A vereinbart, der vom KDC in einer Schlüsseldatenbank verwaltet wird. Für einen Benutzer ist dieser Schlüssel aus dessen Passwort in der Regel unter

Verwendung des MD5-Verfahrens abgeleitet, so dass der KDC nur die Hashwerte der Benutzerpasswörte sicher verwalten muss und der jeweilige Client-Rechner den Master-Schlüssel bei Bedarf, d.h. beim Login des Benutzers, berechnen kann. Der Benutzer ist von einer Verwaltung dieser Master-Schlüssel entlastet. Der Benutzer präsentiert beim Login-Vorgang wie gewöhnlich sein Passwort, das der Client-Rechner verwendet, um den Master-Schlüssel für die Kommunikation mit dem KDC zu bestimmen. Ein KDC verwaltet die Master-Schlüssel sicher, indem er sie mit seinem eigenen Schlüssel verschlüsselt. Zur Verschlüsselung wird in der Regel der DES verwendet, wobei die Kerberos in Version 5 auch den Einsatz anderer Verschlüsselungsverfahren ermöglicht. Dazu ist ein Datenfeld in den Datenpaketen vorgesehen, in das der Identifikator des zu verwendenden Kryptoverfahrens eingetragen werden kann. Alle Master-Schlüssel der Principals werden mit dem gleichen Schlüssel des KDC, dem KDC-Master-Schlüssel, verschlüsselt.

Im KDC wird jeder Principal durch das Tupel $(Name,\ Instanz,\ Bereich)$ beschrieben. Falls der Principal ein Benutzer ist, so bedeutet der Eintrag `Name` die Benutzerkennung und die Instanz ist entweder `null` oder sie repräsentiert spezifische Attribute, wie beispielsweise *root*. Handelt es sich bei dem Principal um einen Service, so bezeichnet `Name` dessen Namen und die Instanz ist die IP-Adresse des Rechners, auf dem der Dienst implementiert ist. Der Bereich wird verwendet, um zwischen unterschiedlichen Authentifikationsbereichen differenzieren zu können.

Bei einer erfolgreichen Anmeldung wird dem Benutzer ein *ticket* ausgestellt.

Ticket

Ein vom TGS ausgestelltes *ticket* ist jeweils nur für einen bestimmten Server gültig. *Tickets* dienen nur zur authentifizierten Nutzung eines Dienstes und führen zu keiner Zugriffsberechtigung. Jedes *ticket* $T_{c,s}$ enthält im Wesentlichen folgende Informationen:

$$T_{c,s} = (S, C, addr, timestamp, lifetime, K_{c,s}).$$

S ist der Name des Servers, der in Anspruch genommen werden soll, C ist der Name des anfordernden Clients, präziser, es ist der Name des Principals, der authentifiziert wurde, und addr ist dessen IP-Adresse. Das *ticket* enthält ferner den Sitzungsschlüssel $K_{c,s}$, der vom KDC für die Kommunikation zwischen S und C erzeugt wird und der auch C bekannt gegeben wird (siehe Nachricht (2) bzw. (4) des Protokolls in Abbildung 1.26). Solange die Lebenszeit des *tickets*, angegeben durch einen Anfangs- und einen Endwert, nicht abgelaufen ist, kann der Client das *ticket* für Zugriffe auf den Server S verwenden. Das *ticket* wird mit dem Master-Schlüssel K_s des Servers S verschlüsselt, $\{T_{c,s}\}^{K_s}$, und so vom TGS dem Client zugestellt, der es seinerseits bei Bedarf S präsentiert.

Da der TGS ebenfalls ein Server ist, dessen Service (nämlich die *ticket*-Ausstellung) nur durch Vorlage eines gültigen *ticket* in Anspruch genommen werden darf, wird der initiale *Ticket-Granting-Ticket (TGT)* benötigt, das der Clientrechner nach dem erfolgreichen Login eines Benutzers automatisch beim KDC anfordert. Die Verwendung solcher TGTs hat den Vorteil, dass der KDC im Wesentlichen zustandslos arbeiten kann, da er nach einer Authentifizierungsanfrage alle Antwortdaten in das *ticket* verpackt und keine Zustandsinformationen über den Client anlegen und verwalten muss.

Authenticator

Ein vom TGS ausgestelltes *ticket* soll innerhalb seiner Gültigkeit vom Client wiederverwendet werden, um damit für Entlastung des Netzes zu sorgen, weil dadurch der Nachrichtenaustausch

Von	An	Nachricht
1. Client	→ KDC	$Joe, TGS, Nonce1$
2. KDC	→ Client	$\{K_{Joe,TGS}, Nonce1\}^{K_{Joe}}$, $\{T_{Joe,TGS}\}^{K_{TGS}}$
3. Client	→ TGS	$\{A_{Joe}\}^{K_{Joe,TGS}}$, $\{T_{Joe,TGS}\}^{K_{TGS}}, NFS, Nonce2$
4. TGS	→ Client	$\{K_{Joe,NFS}, Nonce2\}^{K_{Joe,TGS}}$, $\{T_{Joe,NFS}\}^{K_{NFS}}$
5. Client	→ NFS	$\{A_{Joe}\}^{K_{Joe,NFS}}$, $\{T_{Joe,NFS}\}^{K_{NFS}}$

Abbildung 1.26: Kerberos-Protokollablauf (Version 5)

reduziert wird. Andererseits öffnet diese Eigenschaft einem Angreifer die Möglichkeit eines Wiedereinspielungs-Angriffs, d.h. wenn der Angreifer ein gültiges *ticket* in die Hände bekommt und dann dieses *ticket* S als eigenes vorzeigt. Um dies zu verhindern und die Authentizität desjenigen Principals zu verifizieren, der einem Server ein *ticket* $T_{c,s}$ vorweist, muss der Principal einen sogenannten *authenticator* erzeugen und diesen zusammen mit dem *ticket* versenden. Die Erzeugung des *authenticator* erfolgt, ebenfalls transparent für den Benutzer, durch dessen Clientrechner. Mit dem *authenticator* beweist der Principal, also der Client, dass auch er im Besitz des Sitzungsschlüssels $K_{c,s}$ ist. Um Wiedereinspielungen abzuwehren, muss der *authenticator* einen aktuellen Zeitstempel enthalten. Jeder *authenticator* darf nur einmal von einem Client verwendet werden und enthält im Wesentlichen folgende Informationen:

$$A_c = (c, addr, timestamp).$$

Der Client verschlüsselt den *authenticator* mit dem gemeinsamen Schlüssel $K_{c,s}$ bevor er ihn dem Server S vorlegt. Diesen gemeinsamen Schlüssel erhält der Client (siehe Protokollschritte unten) vom TGS, indem dieser ihn mit einem Schlüssel verschlüsselt, der zwischen dem TGS und dem Client vereinbart ist. Beim initialen Schritt ist dies der Master-Schlüssel des Principals.

Protokollschritte

Die Protokollschritte zur Erlangung des initialen *ticket* für einen Principal namens `Joe` (Schritte (1) und (2)) sowie zur Ausstellung eines *ticket* (Schritte (3) bis (5)) zur Nutzung eines Servers, hier des NFS-Servers, sind in Abbildung 1.26 zusammengefasst. Die Nummerierung entspricht den bereits in Abbildung 1.25 angegebenen Protokollschritten. Der in dem Protokoll angegebene Client C ist ein Prozess, der im Auftrag eines Benutzers tätig ist, in dem Beispiel ist es der Benutzer namens `Joe`.

Zur Erlangung des initialen *ticket* sendet der Client in Nachricht (1) den Namen des zu authentifizierenden Principals, hier `Joe`, zusammen mit einer *nonce* zum KDC. Falls der Benutzer in der Datenbank des KDC registriert ist, stellt dieser ein TGT $T_{Joe,TGS}$ für Joe aus und verschlüsselt es mit dem geheimen Schlüssel K_{TGS} des Servers TGS

$\{T_{joe,TGS}\}^{K_{TGS}}$.

Das *ticket* enthält den vom KDC erzeugten Sitzungsschlüssel $K_{Joe,TGS}$. Dieser wird in Schritt (2) zusammen mit dem *ticket* an den Client zurückgesandt. Damit diese Information vertraulich bleibt, ist sie mit dem Master-Schlüssel von `Joe` verschlüsselt, also

$\{K_{Joe,TGS}, Nonce1\}^{K_{Joe}}$.

Die verwendeten *nonces* dienen zur Erkennung von Replay-Attacken. Um das TGT entschlüsseln zu können, benötigt der Clientrechner den Master-Schlüssel K_{Joe} des Benutzers `Joe`. Da dieser aus dem Benutzerpasswort abgeleitet wird, berechnet der Client ihn aus den ihm vorliegenden bzw. interaktiv vorzulegenden Passwortinformationen. *Tickets* und Sitzungsschlüssel sind von den Clientrechnern sicher zu verwalten. Der Client hält alle verwendeten *tickets* im Cache und löscht sie nach dem Logout des Benutzers.

Um mit dem TGS bzw. KDC Kontakt aufnehmen zu können, muss der Clientrechner diesen lokalisieren. Die entsprechende Information kann zum Beispiel manuell in einer Konfigurationsdatei eingetragen sein. In der Regel wird hierfür die Datei `krb5.conf` verwendet. Alternativ kann ein KDC seine Dienste aber auch über DNS bekannt machen.

Der TGS stellt *ticket* auf Anfragen gemäß Nachricht (3) aus. In der Anfrage ist neben dem Namen des Principals und einer *nonce* auch der Name des Servers angegeben, für den ein *ticket* benötigt wird. Im Beispiel ist dies der NFS-Server. Über die Vorlage des TGT, hier $T = \{T_{Joe,TGS}\}^{K_{TGS}}$, und eines *authenticator* dafür, hier $A = \{A_{Joe}\}^{K_{Joe,TGS}}$ weist der Client seine Authentizität gegenüber dem TGS nach. Der TGS entschlüsselt das *ticket* T unter Anwendung seines Schlüssels K_{TGS} und besitzt danach ebenfalls den Sitzungsschlüssel $K_{Joe,TGS}$. Mit diesem kann er im nächsten Schritt den *authenticator* A entschlüsseln und die Aktualität der darin enthaltenen Zeitmarke überprüfen. Veraltete *authenticators* werden als ungültig zurückgewiesen.

Nach einer erfolgreichen Authentifikation erzeugt das KDC einen Sitzungsschlüssel, hier ist es $K_{Joe,NFS}$, für die Kommunikation zwischen dem Benutzer `Joe` und dem NFS-Server. Der TGS sendet dann in Schritt (4), analog zu Schritt (2), ein *NFS-ticket* sowie diesen Schlüssel zurück.

Wird eine wechselseitige Authentifikation gefordert, so ist das angegebene Protokoll um einen Schritt (4') zu erweitern. Der Server authentifiziert sich gegenüber dem Client, indem er den um eins erhöhten Zeitstempel aus dem *authenticator* $\{A_{Joe}\}^{K_{Joe,NFS}}$ verschlüsselt mit dem gemeinsamen Schlüssel $K_{Joe,NFS}$ zurückschickt. Damit hat der Server bewiesen, dass er den Schlüssel $K_{Joe,NFS}$ sowie den geheimen Schlüssel K_{NFS} kennt.

Sicherheit von Kerberos

Kerberos stellt ohne Zweifel einen erheblichen Fortschritt in Bezug auf die Sicherheit in offenen Netzen dar, da auf netzwerkweite Dienste authentifiziert zugegriffen werden kann und Benutzerpasswörter dazu nicht über das Netz übertragen werden müssen. Dennoch weist das Protokoll einige Defizite auf. Die meisten Mängel, deren Ursache in der ursprünglichen Ausführungsumgebung, dem Athena-Projekt, zu finden waren, sind mit der Version 5 bereits behoben worden. Nach wie vor problematisch ist jedoch die Verwaltung von Sitzungsschlüsseln, die in der Verantwortung eines jeden Clientrechners liegt, sowie die Verwendung von IP-Adressen. Im Athena-Projekt wurden nur Einbenutzerrechner verwendet, so dass die sichere Speicherung von Sitzungsschlüsseln und *tickets* keine spezifischen Maßnahmen erforderte. Dies trifft bei offenen Mehrbenutzerumgebungen natürlich nicht mehr zu, so dass die sichere Verwaltung dieser Informationen nun eine wesentliche Voraussetzung für einen sicheren Betrieb von Kerberos ist. Da

tickets automatisch im /tmp-Verzeichnis abgelegt werden, ist es in Mehrbenutzerumgebungen sehr einfach, *tickets* anderer Benutzer zu stehlen und zu missbrauchen.

Um zu verhindern, dass ein Angreifer eine Offline-Attacke auf einen Master-Schlüssel eines Benutzers durchführen kann, indem der Angreifer sich einfach ein *ticket* für diesen Benutzer ausstellen lässt, werden durch den KDC/TGS keine *tickets* für Benutzer ausgestellt.

Die Authentizität eines Clients wird anhand des *authenticator* und der darin enthaltenen IP-Adresse überprüft. Da IP-Adressen fälschbar sind, könnte ein Angreifer einen *authenticator* abfangen und in eine maskierte TCP/IP-Verbindung einschleusen. Während die Version 4 hierfür keine Abwehrmaßnahmen vorsieht, bietet Version 5 ein CR-Protokoll an. Dies ist aber lediglich eine wählbare Option, die für einen sicheren Betrieb aber unbedingt genutzt werden sollte.

Wesentlicher Schwachpunkt von Kerberos ist sicherlich das Passwortkonzept, das die Basis für die initiale Vergabe von *tickets* an Benutzer und Server bildet. Mit erfolgreichen Passwort-Cracking-Angriffen auf Clientrechner lässt sich der gesamte Authentifikationsprozess unterlaufen. Wer es geschafft hat, sich auf einem Clientrechner unter einer falschen Identität einzuloggen, kann unter dieser Maske auch netzwerkweit aktiv werden. Aber auch die Master-Schlüssel-Verwaltung auf den KDCs ist problematisch. Der KDC verschlüsselt alle diese Schlüssel mit dem gleichen Server-Master-Schlüssel, der ebenfalls auf der Festplatte gespeichert ist. Falls der KDC oder ein Backup kompromittiert ist, müssen alle Passworte von Benutzern erneuert werden.

Die Verwendung der *authenticators* zur Erkennung von Replay-Angriffen ist ebenfalls nicht ganz unproblematisch, da die Erkennung auf der Basis der Aktualität von Zeitstempeln erfolgt. Kerberos basiert darauf, dass sich die beteiligten Rechner bezüglich einer globalen Zeit synchronisieren. Falls es aber einem Angreifer gelingt, einem Server eine falsche Zeit (eine bereits abgelaufene Zeit) als aktuelle Zeit „unterzuschieben“, so ist es ihm möglich, bereits verwendete *authenticators* für Maskierungsangriffe zu missbrauchen. Der Server kann dann nämlich beim Testen der Aktualität des *authenticator* nicht mehr erkennen, dass es sich um eine Wiedereinspielung handelt. Da viele Rechner auf nicht vertrauenswürdige Synchronisationsprotokolle für ihre Uhren zurückgreifen, sind solche Angriffe keineswegs unrealistisch. Abhilfen könnten hier CR-Schritte unter Verwendung von $nonces$ schaffen. Diese sind aber auch in der Version 5 nicht vorgesehen.

Zu beachten ist zudem, dass die vom KDC erzeugten Sitzungsschlüssel keine Sitzungsschlüssel im eigentlichen Sinn sind. Vielmehr kann ein solcher Schlüssel von einem Client für jede Kommunikationsverbindung mit dem Server verwendet werden, für den das *ticket* gilt. Da *tickets* unter Umständen eine lange Gültigkeit haben, unter Version 4 beträgt die maximale Gültigkeitsdauer 21 Stunden, während unter Version 5 eine maximale Dauer sogar bis zum 31.12.9999 möglich ist, wird in diesem Fall auch der Sitzungsschlüssel über einen sehr langen Zeitraum verwendet und ist währenddessen möglichen Angriffen ausgesetzt. Version 5 sieht jedoch eine Option vor, einen Sitzungsschlüssel zu erneuern bzw. *tickets* mit langer Lebensdauer zurückzurufen. Von dieser Möglichkeit sollte bei *tickets* mit langer Gültigkeit unbedingt Gebrauch gemacht werden.

In Version 5 wird das Problem dadurch gelöst, dass es eine Hierarchie von KDCs gibt, die das *Forwarding* transparent und insbesondere ohne eine Passwortübertragung durchführen. Die KDCs realisieren somit ein SSO. Dies ist einerseits für eine einfache Handhabung des Systems sicher sehr wünschenswert und fördert die Akzeptanz von Sicherheitsmaßnahmen beim Endbenutzer; andererseits entfallen durch das automatisierte Vorgehen Interaktionen mit dem Benutzer, die für eine erneute Überprüfung der Authentizität nützlich sein könnten. Ein erfolgreicher

Maskierungsangriff in einem SSO-System kann somit einen großen Wirkungsbereich haben, so dass der Zugriff auf sensible Daten durch zusätzliche, explizite Kontrollen differenziert geschützt werden sollte.

Analog zum Secure RPC verzichtet auch Kerberos auf Maßnahmen zur Gewährleistung der Nachrichtenintegrität und der nicht abstreitbaren Zuordenbarkeit von Nachrichten zu Absendern, was durch den Einsatz digitaler Signaturen erreichbar wäre. Es ist zu hoffen, dass in Folgeversionen diese Problembereiche behandelt werden. Durch den sich rasant verbreitenden Einsatz von Smartcards ist zu erwarten, dass in nachfolgenden Kerberos-Versionen das sicherheitskritische Passwortverfahren durch ein Smartcard-basiertes CR-Protokoll mit öffentlichen Schlüsseln ersetzt wird. Trotz der aufgezeigten Mängel stellt Version 5 des Kerberos-Protokolls eine deutliche Verbesserung der Sicherheit in vernetzten IT-Systemen dar.

1.7.2 Microsoft Passport-Protokoll (MSP)

Die dem Passport-Konzept von Microsoft [12] zu Grunde liegende Idee besteht darin, einem Internetbenutzer eine Online-Identität zu verschaffen. Diese Identität erhält er nach einer einmaligen Authentifizierung bei einem zentralen Microsoft-Dienst, dem Passport-Dienst. Mit dieser Identität wird dem Nutzer ermöglicht, auf Web-Seiten von Anbietern zuzugreifen, die Passport lizenziert haben, ohne sich erneut authentifizieren zu müssen. Die zur Authentifizierung benötigten Zugangsdaten werden in einer zentralen Datenbank in der `.passport.com`-Domäne verwaltet, auf die mehrere dedizierte Passport-Server in dieser Domäne Zugriff haben [12, 40, 39].

Das Microsoft Passport-Modell

Bei Passport handelt es sich um einen Web-basierten Dienst, der auf der Verwendung von Standardtechniken, wie SSL, *HTTP-Redirect* [14], Cookies und Server-seitigen Skripten basiert. Zur Nutzung des Dienstes muss also beim Client neben dem Browser keine spezielle Software geladen oder installiert werden.

Das Passport-Modell unterscheidet drei Komponenten:

- Den Benutzer, der zum Internetzugriff einen Web-Browser verwendet und der sich in der Regel vorab beim Passport-Server als Nutzer bzw. Kunde registriert hat,
- die Anbieter von Web-Inhalten, so genannte Partner-Sites und
- die Passport Login-Server sowie die zentrale Datenbank, die Authentifizierungsdaten der Kunden sowie Daten über deren jeweiliges Profil speichert. Kunden authentifizieren sich einmal gegenüber einem der Login-Server und eine korrekte Authentifikation wird von allen anderen Login-Servern anerkannt. Über einen Login-Server können Kundenprofile an Anbieter weitergegeben werden, wenn ein Kunde dies gestattet. Die kundenspezifisch gespeicherten Daten werden unterschieden nach Kundenprofilen (u.a. Adresse, Alter) und *Wallet*-Informationen (u.a. Kreditkarteninformationen).

Die beteiligten Komponenten wickeln entweder ein *Sign-in*-Protokoll zur Authentifikation oder ein *Wallet*-Protokoll zum Online Shopping ab. Das *Sign-in*-Protokoll wird ausführlich dargestellt, während das *Wallet*-Protokoll aufgrund der Ähnlichkeit hier nicht beschrieben wird.

Standard Sign-in-Protokoll

Greift ein Kunde auf die Web-Seite eines mit der Sicherheitsstufe Standard *Sign-in* registrierten Anbieters zu, so wird das in Abbildung 1.27 skizzierte Protokoll abgewickelt.

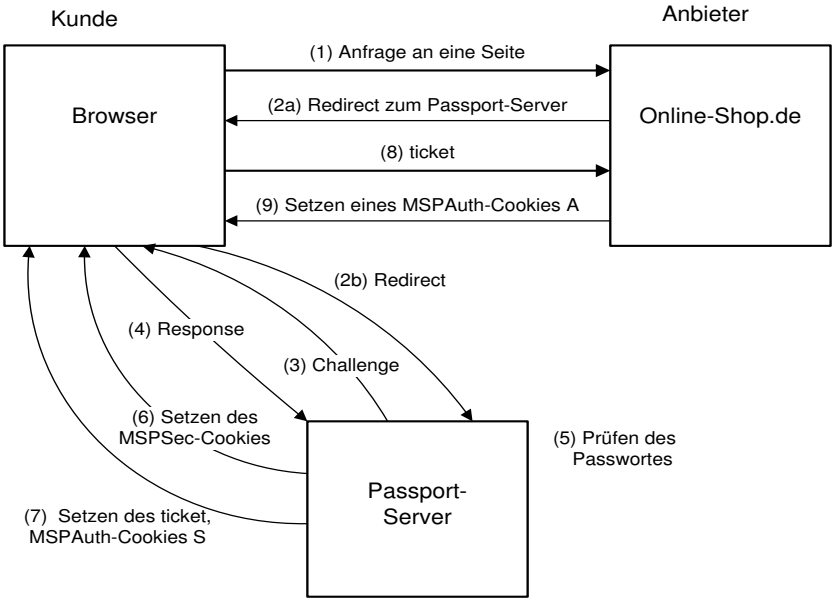

Abbildung 1.27: Passport Single Sign-in Protokoll

Der in Abbildung 1.27 dargestellte Ablauf:

1. Der Kunde stellt über einen *Web Browser* eine Anfrage zur Nutzung eines Services.
2. Der Service leitet den Kunden über ein *HTTP-Redirect* zur Anmeldung beim Passport-Dienst an eine Passport-Anmeldeseite eines Passport-Servers aus der Domäne `.passport.com` weiter. Dazu werden über URI-Parameter die eindeutige Identifikation des Anbieters sowie die Rückkehr-URL (das ist meist die Seite des Anbieters, von der aus der Kunde den Login-Prozess gestartet hat) über ein *HTTP-Redirect* an den Kunden-Browser und von da an den Passport-Server übermittelt.
3. Der Server prüft, ob die Anfrage von einem bei ihm registrierten Anbieter stammt. Ist dies der Fall, fragt der Server den Kunden nach dessen E-Mail und Passwort.
4. Die Antworten werden per SSL verschlüsselt vom Kunden zum Server übertragen.
5. Der Server prüft, ob es sich um gültige Daten handelt.
6. Ist dies der Fall, so setzt der Server ein Cookie, das `MSPSec`-Cookie, beim Browser des Kunden unter dem Pfad `/ppsecure`. In diesem Cookie werden die verschlüsselten Zugangsdaten, das Passwort des Kunden gespeichert, so dass er bzw. ein anderer Passport-Server der Domäne diesen bei nachfolgenden Anfragen identifizieren kann, ohne dass der Kunde erneut E-Mail und Passwort eingeben muss. Dafür wird ein symmetrischer Tripel-DES-Schlüssel verwendet, der allen Passport-Servern der Domäne bekannt ist.

7. Weiterhin erstellt der Server ein Zugriffsticket, einen Passport (u.a. enthält das *ticket* eine *Personal User ID (PUID)*) in Form eines Cookies, das ist das `MSPAuth`-Cookie `S`, für den Anbieter. Der Inhalt dieses Cookies wird mittels eines symmetrischen Tripel-DES-Schlüssel verschlüsselt. Dieser Schlüssel muss dem Passport-Server sowie allen Anbietern bekannt sein, damit die verschlüsselten Daten in den Cookies entschlüsselt werden können. Die verschlüsselten Passport-Informationen werden als *Query*-Parameter an die Rückkehr-URL angehängt und der Kunde wird zu dieser URL, also der Seite des Anbieters, weitergeleitet.

8. Über das `MSPAuth`-Cookie `S` authentifiziert sich der Kunde gegenüber dem Anbieter und kann auf dessen Daten zugreifen.

9. Der Anbieter extrahiert die Daten aus den Parametern, entschlüsselt sie und kann den Kunden anhand der übermittelten PUID in seinem eigenen Datenbestand ermitteln. Weiterhin setzt der Anbieter seinerseits ein, mit einem eigenen, geheimen, symmetrischen Tripel-DES-Schlüssel verschlüsseltes `MSPAuth-Cookie A` im Browser des Kunden, um bei nachfolgenden Zugriffen den Kunden zu identifizieren. Für unterschiedliche Anbieter werden abhängig davon, wie der Benutzer die Profilweitergabe spezifiziert, verschiedene Passports ausgegeben. Das `MSPAuth`-Cookie enthält neben der eindeutigen Benutzerkennung PUID mindestens auch zwei Zeitstempel, die zum einen den Zeitpunkt der letztmaligen Cookieerneuerung und zum anderen den Zeitpunkt der letztmaligen manuellen Authentifikation festhalten. Anbieter können beim Anmelden von Kunden Restriktionen festlegen, so dass beispielsweise Kunden nur dann akzeptiert werden, wenn ihre letztmalige manuelle Authentifikation nicht zu lange zurückliegt.

Meldet sich ein Benutzer wieder von dem Passport-Dienst ab, dies geschieht z.B. durch das Anklicken eines Abmeldebuttoms auf der Web-Seite des Anbieters, so ermittelt der Passport-Server die Liste aller Web-Anbieter, bei denen sich der abzumeldende Kunde angemeldet hat. Dies geschieht einfach dadurch, dass bei jedem Anmelden bei einem Anbieter ein neues Cookie beim Kunden gesetzt wird, das diese aktuelle Liste enthält und das beim Abmelden an den Passport-Dienst zurückgereicht wird. Der Server startet bei jedem in der Liste vermerkten Anbieter ein Skript, durch das die Cookies, die die Anbieter beim Kunden gesetzt haben, wieder gelöscht werden.

Der Passport-Server kann dies nicht selber machen, da Cookies nur von demjenigen Server wieder gelöscht werden dürfen, der sie gesetzt hat. Die im Rahmen des Passport-Protokolls gesetzten Cookies haben nur eine begrenzte Lebensdauer, so dass sie nach deren Ablauf oder bei der Beendigung der Browsersitzung automatisch gelöscht werden. Dieses ist jedoch nicht der Fall, wenn der Benutzer die Option automatische Anmeldung (*Sign-me in automatically*) gewählt hat. In diesem Fall sind die Cookies persistent.

Sicherheit des Passport-Konzepts

Im Gegensatz zu dem SSO-Ansatz, der durch Kerberos umgesetzt wird, sind die an dem Authentifikationsprozess beteiligten Partner Anbieter von Web-Diensten und -Inhalten, die sich unter ganz unterschiedlicher administrativer Kontrolle befinden. Demgegenüber befinden sich die Kerberos-Server, für die ein ticketbasiertes SSO realisiert wird, unter der gleichen, administrativen Kontrolle bzw. sie sind in hierarchische Verwaltungsdomänen untergliedert. Das Passport-Konzept ist der Versuch, die Heterogenität der Web-Welt zu überwinden, den Benutzern einen einfachen Zugriff auf zugangskontrollierte Web-Dienste zu ermöglichen und gleichzeitig die

Sicherheit des Authentifizierungsvorgangs zu gewährleisten. Im Folgenden werden einige Problembereiche im Zusammenhang mit dem Konzept diskutiert.

Einige Probleme, die im Passport-Konzept auftauchen können:

1. Der Authentifizierungsserver und der Dienstserver verwenden für Verschlüsselung immer denselben Schlüssel. Ist dieser gebrochen, sind alle Nutzer davon betroffen, weil die Daten nicht mehr vertraulich gesendet werden können.
2. Die zentrale Verwaltung der Nutzerprofile durch den Server macht ihn zum attraktiven Angriffspunkt.
3. Das Passport-Protokoll verwendet Session- und persistente Cookies, um die verschlüsselte Benutzerdaten im Browser zu speichern. Browser mit einer Konfiguration *disable cookies* können das Passport-Protokoll nicht nutzen.
4. Weiterhin gefährden Cookies die Privatsphäre des Nutzers, wenn sie z.B. kombiniert mit JavaScript benutzt werden [17, 13]. Wenn ein Nutzer vergisst, sich auf den öffentlichen Rechnern auszuloggen, kann es passieren, dass er gültige Authentifizierung-Cookies nicht löscht und sie von nachfolgenden Benutzern missbraucht werden können.

1.7.3 Single-Sign-On mit SAML

In diesem Abschnitt wird die Spezifikation der SAML-Assertions und ihre Nutzung im *Web Browser SSO Profile* beschrieben, das im [23] definiert ist. Die SAML-Assertions und die ausgetauschten Anfrage-Antwort Nachrichten selbst sind in [21] definiert und die Spezifikation wie SAML mit den Kommunikationsprotokollen wie z.B. SOAP, HTTP etc. eingesetzt werden kann, wird in [22] definiert und wird hier nicht weiter beschrieben.

Beteiligte Komponenten:

- Client des Nutzers ist z.B. ein *Web Browser*.
- Identity-Provider ist eine Authentifizierungsstelle, die die Nutzerdaten verwaltet und die Authentifizierung vollzieht. Bei erfolgreicher Authentifizierung wird dem authentifizierten Nutzer eine signierte Assertion, d.h. ein Ausweis über die erfolgreiche Authentifizierung, ausgestellt. Die Nutzer sollen sich einen Identity-Provider aussuchen, dem sie und auch der Service-Provider vertrauen, da nur dann die ausgestellte Assertion wirklich aussagekräftig ist.
- Service-Provider bietet und verwaltet geschützte Informationen und Dienste. Der Zugriff durch den Nutzer erfolgt nur mit einer gültigen Assertion eines vertrauten Identity-Provider. Die vom Nutzer übermittelte Assertion dient hauptsächlich der Identifizierung. Die Autorisierung wird durch lokale Politiken durchgeführt.

SAML Assertions:

Eine Assertion ist ein Ausweis, der vom Identity-Provider ausgestellt wird und Informationen über ein Subjekt enthält. Diese Informationen können nun von einem Service-Provider dazu verwendet werden, um Zugriffskontrolle durchzuführen.

Die SAML-Spezifikation definiert drei verschiedene SAML-Assertions, um passende Informationen oder Aussagen über das Subjekt darzustellen. Diese drei Assertions müssen nicht, können aber signiert werden. Nun zu den einzelnen Assertions:

- Authentifizierungs-Assertion: Das Subjekt in der Assertion wurde authentifiziert und die Assertion enthält Informationen über die Authentifizierung des Subjektes wie z.B. Authentifizierungsmethode. Abbildung 1.28 zeigt eine mögliche Authentifizierungs-Assertion.

```
<saml:Assertion
 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
 MajorVersion="1" MinorVersion="1"
 AssertionID="..."
 Issuer="https://idp.org/saml/"
 IssueInstant="2002-06-19T17:05:37.795Z">
 <saml:Conditions
   NotBefore="2002-06-19T17:00:37.795Z"
   NotOnOrAfter="2002-06-19T17:10:37.795Z"/>
 <saml:AuthenticationStatement
   AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
   AuthenticationInstant="2002-06-19T17:05:17.706Z">
   <saml:Subject>
     <saml:NameIdentifier Format=
       "urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
       user@mail.idp.org
     </saml:NameIdentifier>
     <saml:SubjectConfirmation>
       <saml:ConfirmationMethod>
         urn:oasis:names:tc:SAML:1.0:cm:artifact
       </saml:ConfirmationMethod>
     </saml:SubjectConfirmation>
   </saml:Subject>
 </saml:AuthenticationStatement>
</saml:Assertion>
```

Abbildung 1.28: Eine Authentifizierungs-Assertion

- Attribute-Assertion: Das Subjekt in der Assertion ist mit den, in der Assertion enthaltenen Attributen assoziiert. Ein Beispiel für Attribute-Assertion ist in Abbildung 1.29 dargestellt.
- Autorisations-Assertion: Diese Assertion gibt an, ob eine Anforderung eines Zugriffversuchs des Subjektes auf ein geschütztes Objekt erlaubt oder verboten wurde. Diese Assertion wird in Fällen verwendet, in denen die Zugriffskontrolle delegiert wird. In Abbildung 1.30 dargestellten Autorisations-Assertion wird ein Zugriff auf die Seite https://sp.-org/confidential_report.html erlaubt.

Die äußere Struktur einer Assertion ist generisch. Innerhalb der Assertion befindet sich eine Menge von Elementen, die die Authentifizierung, Attribute, Entscheidung über Autorisation oder auch selbstdefinierte Informationen oder Aussagen beschreiben. Erweiterungen der SAML-Assertion sind erlaubt.

```
<saml:Assertion
 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
 MajorVersion="1" MinorVersion="1"
 Issuer="https://idp.edu/saml/" ...>
 <saml:Conditions NotBefore="..." NotAfter="..."/>
 <saml:AuthenticationStatement
   AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X509-PKI"
   AuthenticationInstant="...">
   <saml:Subject>...</saml:Subject>
 </saml:AuthenticationStatement>
 <saml:AttributeStatement>
   <saml:Subject>...</saml:Subject>
   <saml:Attribute AttributeName=
     "urn:mace:dir:attribute-def:eduPersonScopedAffiliation"
     AttributeNamespace=
     "urn:mace:shibboleth:1.0:attributeNamespace:uri">
     <saml:AttributeValue Scope="idp.edu">
       member
     </saml:AttributeValue>
     <saml:AttributeValue Scope="idp.edu">
       student
     </saml:AttributeValue>
   </saml:Attribute>
 </saml:AttributeStatement>
</saml:Assertion>
```

Abbildung 1.29: Eine Attribute-Assertion

Einblick in das Web Browser Profile

Abbildung 1.31 zeigt eine Protokollschablone mit der SSO erreicht werden kann. Die einzelnen Schritte werden im folgenden Abschnitt beschrieben. Innerhalb jedes Schrittes können, abhängig von der Implementierung und genutzter Protokolle, eine oder mehrere Nachrichten ausgetauscht werden.

Protokollschritte:

1. Der Nutzer schickt eine HTTP-Anfrage an den Service-Provider, um auf eine geschützte Ressource zugreifen zu können, ohne dass er sich davor beim Authentifizierungsdienst authentifiziert hat.

2. Der Service-Provider stellt fest, bei welchem Identity-Provider der Nutzer registriert ist, und leitet ihn zu diesem weiter, damit sich der Nutzer dort zuerst authentifiziert. Die Vorgehensweise des Servers ist abhängig von der Implementierung. Die SAML-Spezifikation definiert *SAML identity provider discovery profile* [23], um den Nutzer zum richtigen Identity-Provider zu leiten.

3. Der Nutzer leitet die vom Service-Provider ausgestellte Nachricht `<AuthnRequest>` an den Identity-Provider, bei dem er registriert ist.

4. Der Nutzer wird vom Identity-Provider authentifiziert.

```
<saml:Assertion
 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
 MajorVersion="1" MinorVersion="1"
 Issuer="https://idp.org/saml/" ...>
 <saml:Conditions .../>
 <saml:AuthorizationDecisionStatement
   Decision="Permit"
   Resource="https://sp.org/confidential_report.html">
   <saml:Action>read</saml:Action>
   <saml:Subject>...</saml:Subject>
 </saml:AuthorizationDecisionStatement>
</saml:Assertion>
```

Abbildung 1.30: Eine Attribute-Assertion

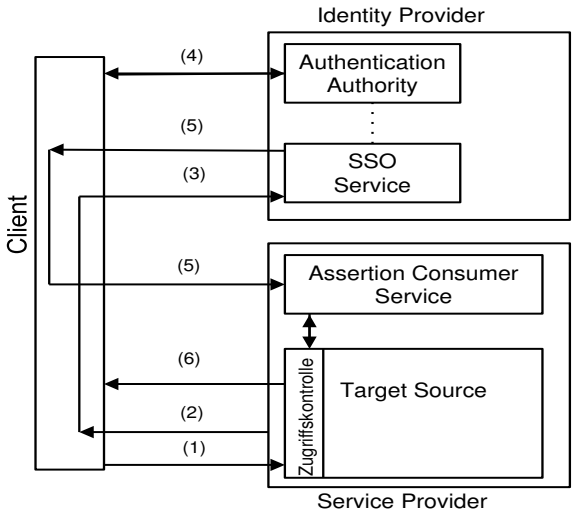

Abbildung 1.31: Web Browser SSO Profie

5. Der Identity-Provider stellt eine Antwortnachricht für den Service-Provider aus, die vom Nutzer zugestellt wird. Diese Nachricht enthält entweder eine Fehlermeldung oder mindestens eine Assertion.

6. Der Service-Provider kann dem Nutzer entweder mit eigener Fehlermeldung antworten oder einen Sicherheitskontext für den Nutzer erzeugen und den Zugriff auf die gewünschte Ressource erteilen.

Sicherheit des Web Browser Profile

Die Sicherheit des *Web Browser Profile* ist schwer zu bewerten, weil der Standard keinerlei Einschränkungen vorgibt und viele Möglichkeiten bietet, Sicherheitselemente einzubauen. Da aber alle diese Elemente optional sind, liegt es bei den Entwicklern, den passenden Sicherheitsgrad zu wählen. Zuverlässige Produkte basierend auf dieser Spezifikation werden erst in nächster Zukunft zum Einsatz kommen.

SAML und das *Web Browser Profile* setzen auf den XML-basierenden Sicherheitsstandards wie XMLdsig, XMLenc, XACML und XKMS auf (siehe Abschnitt 1.4.5 vorgestellt wurden) und können deshalb als zukunftsorientierte und zukunftssichere Technologien eingestuft werden.

1.8 Zugriffskontrolle

Dieser Abschnitt stellt die wichtigsten Mechanismen und Verfahren zur Rechteverwaltung und Zugriffskontrolle vor. Nach einer kurzen Einleitung werden die in der Praxis am weitesten verbreiteten softwarebasierten Kontrollmechanismen eingeführt, die vom Betriebsystemkern bzw. von Serverprozessen in Client-Server-Architekturen zur Verfügung gestellt werden und die Basis für Sicherheitsmodelle bilden.

1.8.1 Einleitung

Die Aufgaben der Rechteverwaltung und der Zugriffskontrolle eines IT-Systems bestehen darin, Mechanismen zur Vergabe von Zugriffsrechten bereitzustellen sowie bei Zugriffen auf zu schützende Objekte die Autorisierung zu prüfen [12, 49].

Die Rechteverwaltung hat zu gewährleisten, dass alle Subjekte und Objekte eindeutig und fälschungssicher identifiziert werden und dass jedes Objekt, für das Zugriffsbeschränkungen festgelegt sind, von der Rechteverwaltung erfasst wird. Mit geeigneten Überprüfungen sollten Widersprüche bei der Rechtevergabe frühzeitig erkannt und aufgelöst werden. Eine frühzeitige Erkennung verhindert, dass das System im Verlauf seiner Ausführung in einen inkonsistenten Rechtezustand eintritt, aus dem sich Sicherheitsprobleme und Informationsverluste durch Prozessabbrüche ergeben können. Die Informationen, die für die Zugriffskontrolle verwendet werden, insbesondere auch die Zugriffsrechte, müssen vor unautorisierter Manipulation besonders geschützt und fälschungssicher gespeichert werden.

Von der Zugriffskontrolle wird gefordert, dass sie jeden Zugriffsversuch kontrolliert und dass diese Kontrolle nicht umgangen werden kann. Zwischen der Rechteüberprüfung und der Ausübung der Rechte sollte keine Aktion möglich sein, die den Rechteentzug zur Folge hat, da sonst unzulässige Rechtezustände eintreten können. Ist es dennoch möglich, dass ein Subjekt weiterhin Rechte wahrnehmen kann, obwohl ihm diese zwischenzeitlich entzogen worden sind, so ist festzulegen, wie lange diese Rechtewahrnehmung dem Subjekt noch möglich sein darf.

1.8.2 Schutz der Objekte

Für den Zugriff auf Objekte können unterschiedlich komplexe Operationen bzw. Methoden zusammen mit Bedingungen festgelegt sein, die den Zugriff objektspezifisch kontrollieren. Aus diesem Grund lassen sich Zugriffskontrollen nicht mehr nur auf die einfache Kontrolle von Lese-, Schreib- oder Ausführungsrechten reduzieren. Erforderlich sind vielmehr individuelle, objekt- bzw. typspezifische Kontrollen. Die benötigten Kontrollen werden von der Objektverwaltung eines Objekttyps, dem Objekt- oder Referenzmonitor (vgl. Abbildung 1.32) realisiert. In heutigen Systemen sind die zu verwaltenden Objekte hauptsächlich Dateien bzw. Verzeichnisse.

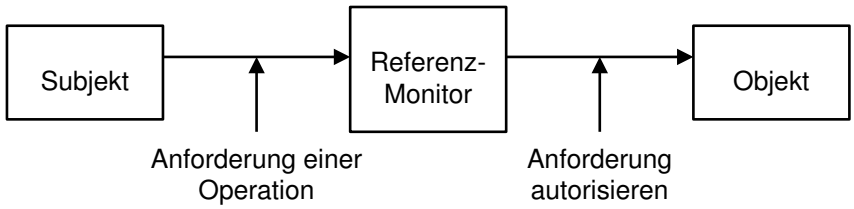

Abbildung 1.32: Allgemeines Modell für die Zugriffskontrolle auf Objekte

Der formalisierte Ausgangspunkt für die Realisierung eines Objektschutzes ist eine Spezifikation der Sicherheitsstrategie mittels eines Zugriffsmatrix-Modells. Die Zugriffsmatrix ist in der Regel nicht sehr dicht besetzt (*sparse*), so dass eine zweidimensionale Implementierung ineffizient ist.

In heutigen IT-Systemen werden im Wesentlichen zwei Konzepte bzw. auch Kombinationen aus diesen zur Implementierung einer Zugriffsmatrix eingesetzt, nämlich Zugriffskontrolllisten (*Access Control List (ACL)*) und Zugriffsausweise (*capability*). Mit Zugriffskontrolllisten wird eine objektbezogene Sicht auf die Zugriffsmatrix realisiert, während Zugriffsausweise eine subjektbezogene Sicht einnehmen. Abbildung 1.33 spiegelt diese beiden Sichten wider.

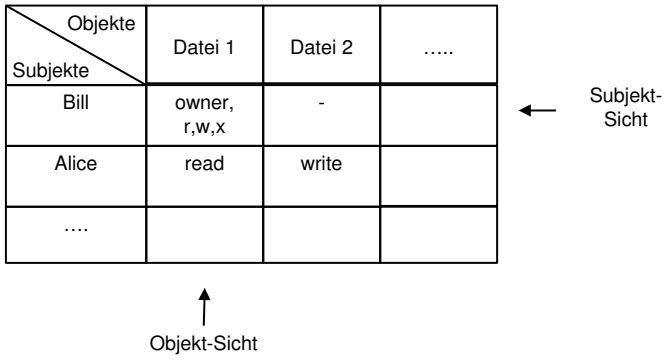

Abbildung 1.33: Realisierungssichten einer Zugriffsmatrix

Zugriffskontrolllisten

Abbildung 1.33 verdeutlicht, dass Zugriffskontrolllisten eine Zugriffsmatrix spaltenweise implementieren. Wie der Name schon sagt, ist eine Zugriffskontrollliste eine listenartig organisierte Datenstruktur, die einem zu schützenden Objekt zugeordnet wird. In Betriebssystemen wie z.B. Unix sind Dateien sowie Verzeichnisse, die zu schützende Objekte und Benutzer, Benutzerguppen sowie Prozesse die Subjekte. Dabei wird jeder Benutzer mindestens einer Gruppe zugeordnet. Jeder Listeneintrag in Unix identifiziert ein Subjekt, meist einen Benutzer oder eine Benutzergruppe, sowie die Rechte, die dem Subjekt an dem Objekt eingeräumt werden. Ein Eintrag enthält also die Felder `Subjekt_Name` und/oder `Gruppen_Name` sowie Zugriffsrechte. Zur Vereinfachung der Notation ist es in der Regel möglich, *wild cards* als Platzhalter zu verwenden, so dass damit einem beliebigen Subjekt oder einer beliebigen Gruppe Rechte erteilt werden können. So bedeutet zum Beispiel in einem Unix-ACL-Eintrag $(Joe, *, rw)$, dass der Benutzer `Joe`, egal in welcher Gruppe er gerade tätig ist, das Lese- und Schreibrecht `rw` an dem

der ACL assoziierten Objekt besitzt. Der ACL-Eintrag $(*, stud, r)$ besagt, dass alle Mitglieder der Gruppe `stud` ein Leserecht besitzen.

Vor- und Nachteile von ACLs

Die Vorteile des Konzepts der Zugriffskontrolllisten liegen in der einfachen Verwaltung der Zugriffsrechte, insbesondere in der einfachen und effizienten Realisierung einer Rechterücknahme. Dazu müssen nur die entsprechenden Einträge in der Zugriffskontrollliste aktualisiert werden. Außerdem ist es sehr einfach, für ein spezifisches Objekt zu bestimmen, welche Subjekte welche Zugriffsrechte an dem Objekt besitzen. Demgegenüber ist es aus dem Blickwinkel eines Subjekts sehr aufwändig, die Menge seiner aktuellen Rechte zu ermitteln.

Problematisch ist, dass die Zugriffskontrolle bei langen Listen aufwändig und ineffizient ist, da bei jedem Zugriff auf ein Objekt dessen Zugriffskontrollliste zu durchsuchen ist. Dies hat zur Folge, dass die meisten Systeme nur sehr einfache Listenstrukturen zulassen und/oder nicht jeden Zugriff kontrollieren. Vielmehr beschränken sie sich darauf, die Berechtigung beim erstmaligen Zugriff, zum Beispiel beim Öffnen einer Datei, zu überprüfen und für nachfolgende Zugriffe eine Berechtigungsbescheinigung auszustellen. Deren Überprüfung erfordert dann keine aufwändigen Kontrollen mehr.

Für den Einsatz in verteilten Systemen mit einer Vielzahl von Subjekten, die unterschiedliche Rechte an Objekten besitzen können, ist das ACL-Konzept aufgrund seiner schlechten Skalierbarkeit eher ungeeignet. Eine Lösung für diesen Problembereich bieten rollenbasierte Zugriffsmodelle (siehe Abschnitt 1.8.3.1).

Zugriffsausweise

Zugriffsausweise sind unverfälschbare *tickets*, die den Besitzer zum Zugriff auf das in dem *ticket* benannte Objekt berechtigen. Neben dem Objektnamen enthält ein Zugriffsausweis auch die Berechtigungen, die dem Besitzer des Ausweises an dem Objekt eingeräumt werden. Das Konzept der Zugriffsausweise ermöglicht die zeilenweise Realisierung einer Zugriffsmatrix. Dazu wird jedem Subjekt eine Liste von Paaren (Objekt, Zugriffsrechte) zugeordnet. Diese Liste, genannt *capability list (C-List)*, spiegelt die Einträge in der dem Subjekt zugeordneten Zeile der Matrix wider.

In Client-Server-Architekturen werden Zugriffsausweise zunehmend von Serverprozessen erzeugt und können ohne zusätzliche Maßnahmen einfach manipuliert werden. Mögliche Maßnahmen bestehen darin, die Zugriffsausweise zu signieren, um eine Manipulation zu verhindern oder zu verschlüsseln, damit die Ausweise vertraulich bleiben und nur von den berechtigten Servern, die den passenden Entschlüsselungsschlüssel haben, eingesehen werden können.

Vor- und Nachteile von *capabilities*

Im Vergleich zu Zugriffslisten bietet das Konzept der *capabilities* eine deutlich größere Flexibilität. Beim Ausstellen einer *capability* werden die für das Subjekt zum Ausstellungszeitpunkt geltenden Rechte an dem betreffenden Objekt in die *capability* eingetragen. Ein Subjekt darf stets dann auf ein Objekt zugreifen, wenn es im Besitz einer gültigen *capability* dafür ist. Dadurch können die Zugriffskontrollen wesentlich vereinfacht werden, da nur eine Zulässigkeitskontrolle durchgeführt werden muss. Das unter Umständen aufwändige Durchsuchen einer langen Kontrollliste bzw. das Befragen eines entfernten, vertrauenswürdigen Zugriffsservers entfällt.

Weiterhin ermöglicht das Konzept der *capabilities* ein dezentrales Sicherheitsmanagement, da die Verwaltung der sicherheitskritischen Informationen und die zur Ausstellung von fälschungssicheren *capabilities* notwendigen, vertrauenswürdigen Managementkomponenten von den Komponenten, die die Zulässigkeitskontrollen durchführen, getrennt werden können. Letztere lassen sich als Bestandteile von Servern der Anwendungsebene implementieren. Zugriffsausweise sind somit als Realisierungskonzept auch besonders für dezentrale Client-Server-Architekturen geeignet.

Da Zugriffsausweise nicht an Subjekte gebunden sind (sie enthalten keinen Subjektidentifikator), können sie auf einfache Weise, ggf. mit gezielten Einschränkungen, an andere Subjekte weitergegeben werden. Eine kontrollierte Weitergabe von Berechtigungen im Sinne einer Delegation von Rechten wird insbesondere in Client-Server-Architekturen benötigt, wenn der Server im Auftrag eines Clients aktiv sein soll (vgl. Abschnitt 1.9). Ferner ist es durch eine einfache Erweiterung der *capability*-Datenstruktur möglich, die Gültigkeitsdauer eines Zugriffsausweises zu beschränken oder einen Zugriff nur unter Vorlage einer Kombination aus mehreren Ausweisen zu erlauben. Damit lassen sich Sicherheitsanforderungen erfassen, die man mit dem Stichwort des „Mehraugenprinzips“ charakterisieren kann. Das heißt, dass ein erfolgreicher Zugriff auf ein derartig geschütztes Objekt gegebenenfalls das koordinierte Zusammenwirken von Subjekten erfordert, die nur jeweils einen Teil der benötigten Berechtigungen besitzen.

1.8.3 Sicherheitsmodelle

1.8.3.1 Rollenbasierte Zugriffsmodelle

Bei einer rollenbasierten Modellierung werden die Berechtigungen zur Nutzung geschützter Komponenten direkt an Rollen und damit an Aufgaben geknüpft. Die durch Rollen modellierten Aufgaben werden von Subjekten durchgeführt, so dass das Sicherheitsmodell festlegen muss, welche Subjekte welche Aufgaben durchführen, d.h. in welchen Rollen sie agieren dürfen. Führt ein Subjekt Aktionen in einer Rolle aus, so ist es in dieser Rolle aktiv. Rollenbasierte Zugriffsmodelle (*role-based access control (RBAC)*) wurden 1996 von R. Sandhu [47] eingeführt und in [46] weiter ausgearbeitet. Die Definition des RBAC-Modells fasst Komponenten und Abbildungen eines rollenbasierten Sicherheitsmodells zusammen.

Definition des RBAC-Modells

Ein rollenbasiertes Sicherheitsmodell wird durch ein Tupel:
$RBAC = (S, O, RL, P, sr, pr, session)$ definiert, wobei,

- S die Menge der Benutzer oder Subjekte des Systems,
- O die Menge der zu schützenden Objekte und
- RL die Menge von Rollen ist. Jede Rolle beschreibt eine Aufgabe und damit die Berechtigungen der Rollenmitglieder.
- P ist die Menge der Zugriffsberechtigungen.
- Die Abbildung sr modelliert die Rollenmitgliedschaft eines Subjektes, d.h. in welchen Rollen ein Subjekt aktiv sein kann.

- Über die Berechtigungsabbildung pr wird jeder Rolle die Menge derjenigen Zugriffsrechte zugeordnet, die die Mitglieder der Rolle während ihrer Rollenaktivitäten wahrnehmen dürfen.
- Eine $session$ gibt an, welche Rollen ein Subjekt aktuell aktiviert hat.

In einem RBAC-System kann ein Subjekt gleichzeitig in unterschiedlichen Sitzungen aktiv sein und es besitzt potentiell alle Berechtigungen aller Rollen, in denen es aktiv ist. Um Berechtigungen wahrnehmen zu können, muss ein Subjekt aktiv in einer Rolle sein, für die die Berechtigung erteilt wurde. Abbildung 1.34 veranschaulicht die Zusammenhänge zwischen den eingeführten Modellierungskonzepten und Relationen.

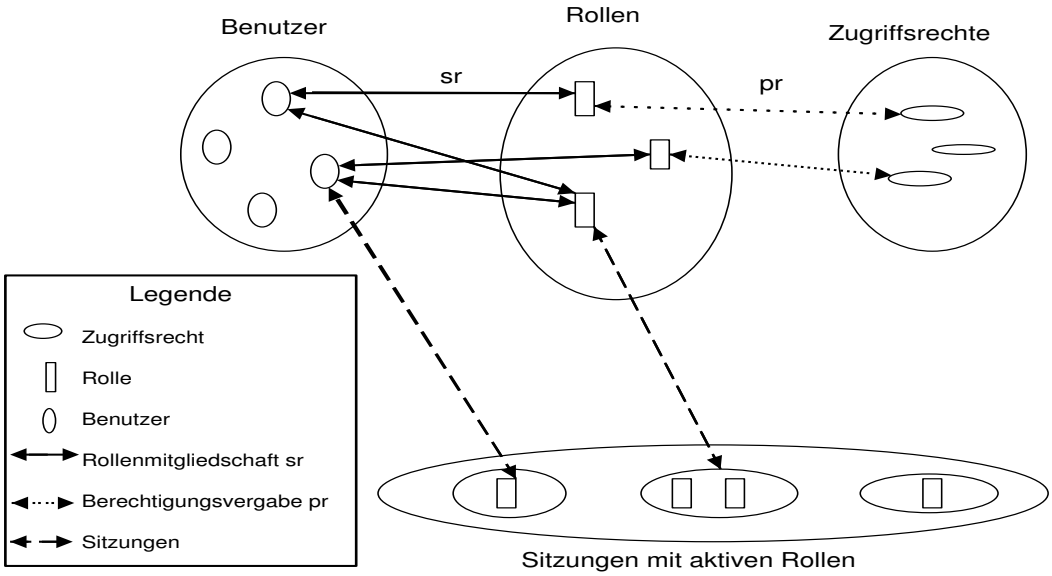

Abbildung 1.34: Zusammenhänge in einem RBAC-Modell

In praxisrelevanten Anwendungsumgebungen wie Unternehmen und Behörden werden Aufgaben und Zuständigkeiten innerhalb von Abteilungen oder Projekten häufig hierarchisch geordnet. Durch eine Erweiterung des RBAC-Modells um eine partielle Ordnung auf den Rollen lassen sich auch hierarchische Berechtigungsstrukturen durch RBAC-Modelle erfassen. Über die Rollenhierarchie können Rechte vererbt werden. Seien $R1$ und $R2$ Rollen mit $R1$ kleinerer Ordnung als $R2$, dann besitzen die Mitglieder von $R2$ mindestens auch alle Rechte der Mitglieder von $R1$. Diese Rechtevererbung kann jedoch auch problematisch sein, da dadurch ein Subjekt unter Umständen mehr Rechte erhält, als es zur Erfüllung seiner Aufgaben benötigt.

Beschränkte Rollenmitgliedschaft

In einem RBAC-Modell ist es häufig sinnvoll bzw. notwendig, Regeln zur Beschränkung von Rollenmitgliedschaften zu formulieren. Diese Beschränkung kann man im RBAC-Modell auf zwei Weisen realisieren, nämlich durch Regeln zur statischen bzw. zur dynamischen Aufgabentrennung.

Regeln zur statischen Aufgabentrennung (*separation of duty*) betreffen den wechselseitigen Ausschluss von Rollenmitgliedschaften. Dadurch wird für die Subjekte festgelegt, in welchen

Rollen sie nicht Mitglied sein dürfen, z.B. weil sich die den Rollen assoziierten Aufgaben wechselseitig ausschließen.

Statische Aufgabentrennung wird unabhängig von aktiven Sitzungen formuliert. Diese statische Festlegung ist jedoch für viele Anwendungsbereiche zu starr. Manchmal reicht es aus, die gleichzeitige Aktivität von Subjekten in unterschiedlichen Rollen zu untersagen, ohne jedoch die Rollenmitgliedschaft generell zu verbieten. Entsprechende Beschränkungen können durch Regeln zur dynamischen Aufgabentrennung formuliert werden.

Dynamische Aufgabentrennung besagt, dass ein Subjekt nicht gleichzeitig in solchen Sitzungen aktiv sein darf, deren assoziierte Aufgaben sich wechselseitig ausschließen. Ein Beispiel dafür wäre, dass ein Kundenbetreuer einer Bank auch selbst ein Kunde dieser Bank sein darf, aber nicht gleichzeitig sein eigener Kundenbetreuer.

Bewertung

Mit einem RBAC-Modell sind Objekte anwendungsspezifisch modellierbar und die Berechtigungen zum Zugriff auf die Objekte werden aufgabenorientiert vergeben. Mit einer RBAC-Strategie lassen sich die Nachteile klassischer, objektbezogener Sicherheitsstrategien vermeiden, da sich die Berechtigungsprofile von Rollen nur selten ändern. Auf diese Weise lassen sich die Probleme der mangelnden Skalierbarkeit objektbezogener Strategien bei sich dynamisch ändernden Subjektmengen stark reduzieren. RBAC-Modelle eignen sich gut für den Einsatz in verteilten Systemen. Die Rollenzuordnung bzw. deren Entzug ist Aufgabe des System- oder des Sicherheitsadministrators.

Das Modell trifft keine a priori Festlegungen hinsichtlich der Menge der beim Einsatzbereich vergebenden Zugriffsrechte, die universell oder objektspezifisch modelliert werden können. Die Regeln zur Formulierung von statischen und dynamischen Beschränkungen von Rollenaktivitäten ermöglichen eine Rechtevergabe gemäß des *need-to-know* Prinzips und unterstützen eine systematische Aufgabenteilung. Die eingeführten Regeln zur Erfassung unzulässiger Abhängigkeiten zwischen Rollen sowie die Möglichkeit, Rollenhierarchien zu modellieren, sind unmittelbar auf existierende Organisations- und Verwaltungsstrukturen in Unternehmen und Behörden übertragbar, so dass in diesem Umfeld RBAC-Modelle sehr gut für große Klassen von Anwendungssystemen geeignet sind.

1.8.3.2 Bell-LaPadula Modell

Das Bell-LaPadula Modell [12, 28, 27, 3] ist das bekannteste Sicherheitsmodell. Es gilt als das erste vollständig formalisierte Modell. Dem Bell-LaPadula Modell liegt ein dynamisches Zugriffsmatrix-Modell $M_t(s, o)$ zugrunde, wobei die Zugriffsrechte durch die Menge der universellen Rechte *read-only*, *append*, *execute*, *read-write*, *control* vorgeschrieben sind. Das *read-only*-Recht erlaubt einen reinen Lesezugriff, das *append*-Recht berechtigt dazu, Daten an ein vorhandenes Objekt anzufügen, das *read-write*-Recht erlaubt den Lese- oder Schreibzugriff und das *control*-Recht ermöglicht die Rechteweitergabe bzw. -rücknahme. Das *execute*-Recht berechtigt zur Ausführung einer ausführbaren Datei.

Als Erweiterung des $M_t(s, o)$ wird eine geordnete Menge von Sicherheitsklassen SC eingeführt, um den zu verarbeitenden Informationen sowie den agierenden Subjekten unterschiedliche Vertraulichkeitsstufen zuzuordnen. Ein Element $X \in SC$ wird durch ein Paar $X = (A, B)$ beschrieben, wobei A eine Sicherheitsmarke und B eine Menge von Sicherheitskategorien (*compartement*) ist.

Jedem Subjekt s wird eine Sicherheitsklasse, die so genannte Clearance $sc(s) \in SC$, und jedem Objekt o wird eine Sicherheitsklassifikation, die sogenannte Classification $sc(o) \in SC$, zugeordnet. Die Clearance $sc(s)$ eines Subjekts s ist die maximale Sicherheitsstufe, die s einnehmen darf. Bei der Anmeldung (*login*) muss ein Subjekt seine aktuelle Clearance $sc_{akt}(s)$ angeben, wobei gilt: $sc_{akt}(s) < sc(s)$.

Systembestimmte Zugriffsbeschränkungen

Der Zugriff auf Objekte wird durch zwei systembestimmte Regeln, die Simple-Security und die *-Eigenschaft, beschränkt. Die Simple-Security, auch bekannt als *no-read-up* Regel, besagt, dass ein Lese- oder Ausführungszugriff auf ein Objekt o durch ein Subjekt s nur dann zulässig ist, wenn s das entsprechende Zugriffsrecht r besitzt und die Objektklassifikation kleiner oder gleich der Subjekt-Clearance ist. D.h. es muss gelten: $r \in M_t(s, o)$ und $sc(s) > sc(o)$.

Die *-Eigenschaft, auch bekannt als *no-write-down* Regel, besagt, dass ein append-Zugriff auf ein Objekt o durch ein Subjekt s nur zulässig ist, wenn die Objektklassifikation mindestens so hoch ist, wie die Clearance des Subjekts, also $append \in M_t(s, o)$ und $sc(s) < sc(o)$, und dass ein Lese-Schreib-Zugriff auf ein Objekt o nur zulässig ist, wenn die Objektklassifikation gleich der Clearance des Subjekts ist, d.h. $read - write \in M_t(s, o)$ und $sc(s) = sc(o)$.

Durch die beiden systembestimmten Regeln sind einfach zu überprüfende Bedingungen formuliert. Diese gewährleisten, dass Informationsflüsse höchstens von unten nach oben entlang der partiellen Ordnung < oder innerhalb einer Sicherheitsklasse auftreten können. Abbildung 1.35 veranschaulicht die zulässigen Flüsse für ein System mit den linear geordneten Sicherheitsmarken unklassifiziert, vertraulich, geheim, und streng geheim.

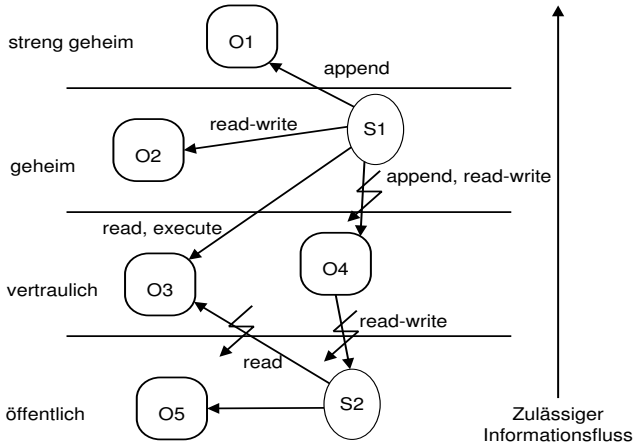

Abbildung 1.35: Zulässige Flüsse im Bell-LaPadula Modell

Bei jedem Zugriff auf ein geschütztes Objekt sind die *Simple-Security*-Eigenschaft sowie die *-Eigenschaft des Bell-LaPadula Modells einzuhalten. Die Zugriffskontrolle überprüft zunächst, ob das zugreifende Subjekt das nötigte Zugriffsrecht besitzt. Im zweiten Schritt wird bei der Ausführung der Operationen überprüft, ob das Subjekt die notwendige Clearance besitzt. So ist für Zugriffe, die nur lesenden Charakter haben, wie das Durchsuchen eines Verzeichnisses *(search directory*) gefordert, dass das zugreifende Subjekt mindestens die Sicherheitseinstufung des Objekts besitzt. Auf diese Weise wird gewährleistet, dass keine hoch klassifizierte Information zu einem niedriger eingestuften Subjekt fließen kann (*Simple Security Property*).

Das Bell-LaPadula Modell wird in der Praxis sehr häufig eingesetzt, obwohl es einige gravierende Mängel aufweist, von denen drei nachfolgend diskutiert werden.

(1) **Beschränkte Ausdrucksfähigkeit**

Es existieren Klassen von Anwendungsszenarien, deren informelle Sicherheitsanforderungen die intendierte Informationsfluss-Strategie erfüllen, die aber dennoch nicht modellierbar sind.

(2) **Problem des blinden Schreibens**

Durch die systembestimmten Regeln ist es möglich, dass ein Subjekt schreibend auf ein höher eingestuftes Objekt zugreifen, aber anschließend die von ihm verursachten Veränderungen nicht lesen darf, da sonst die *no-read-up* Bedingung verletzt wäre. Dieses blinde Schreiben ist im Hinblick auf Integritätsanforderungen sehr problematisch, da ein Subjekt ein Objekt beliebig (z.B. unabsichtlich fehlerhaft) manipulieren kann.

(3) **Problem des entfernten Lesens**

Wendet man die Bell-LaPadula Regeln in einem verteilten System an, so treten einige technische Probleme auf. Eines davon ist das Problem des entfernten Lesens. Betrachten wir wieder die geordnete Menge von Sicherheitsmarken. Betrachten wir ferner einen als geheim eingestuften Rechner A mit einem ebenso eingestuften Subjekt, das entfernt lesend auf einen als vertraulich eingestuften Rechner B zugreifen will. Nach Bell-LaPadula ist dieser Informationsfluss zulässig, da $vertraulich < geheim$ gilt und ein Lese-Zugriff „nach unten“ erlaubt ist. Um den entfernten Zugriff durchzuführen, muss aber eine Verbindung zwischen A und B aufgebaut werden. Ein Verbindungsaufbau erfordert den Nachrichtenaustausch zwischen A und B. D.h. A sendet zunächst eine Aufforderung zur Etablierung einer Verbindung an B. Das Problem besteht darin, dass diese Anfrage von Rechner A eine Schreib-Operation auf dem Rechner B zur Folge hat, so dass ein *write-down* von A zu B vorliegt, was gemäß der Bell-LaPadula Regeln aber nicht zulässig ist. Zur Lösung dieses Problems sind solche Anfragenachrichten speziell zu kontrollieren, so dass sichergestellt ist, dass kein unzulässiger Informationsfluss auftreten kann.

1.8.3.3 Chinese-Wall-Modell

Das Chinese-Wall-Sicherheitsmodell [45, 12], auch bekannt als Brewer-Nash-Modell, wurde entwickelt, um die unzulässige Ausnutzung von Insiderwissen bei der Abwicklung von Bank- oder Börsentransaktionen oder bei der Beratung unterschiedlicher Unternehmen zu verhindern. So soll zum Beispiel durch das IT-System verhindert werden, dass ein Unternehmensberater, der Insiderinformationen über ein Unternehmen besitzt, diese Informationen verwendet, um einem Konkurenzunternehmen Ratschläge zu erteilen. Die grundlegende Idee des Chinese-Wall-Modells basiert darauf, dass zukünftige Zugriffsmöglichkeiten eines Subjekts durch die Zugriffe, die es in der Vergangenheit bereits durchgeführt hat, beschränkt werden. Das heißt, dass die Zulässigkeit von Zugriffen auch von der Zugriffshistorie abhängt (vgl. RBAC mit *constraints* [37]).

1.9 Delegation von Rechten

In diesem Abschnitt wird beschrieben, was Delegation ist, wozu man sie in Webservice-orientierten Systemen braucht, welche Sicherheitsaspekte dabei betrachtet werden sollen und welchen Einfluss die lokalen Politiken darauf haben [49, 31, 38].

Was ist Delegation?

Allgemein bedeutet Delegation die Übertragung von Entscheidungskompetenzen von einer Instanz (Delegierender) an (meist) unterstellte Instanzen/Stellen (Delegationsempfänger oder Delegierte).

Eine mögliche Definition in Informatik könnte wie folgt lauten: Delegation gibt einem Nutzer die Möglichkeit, seine Rechte an einen Vermittler (z.B. einem Dienst) weiterzugeben, damit dieser gegenüber einem Dritten im Namen des Nutzers agieren kann.

Die beteiligten Komponenten bei einer Delegation sind also: der Delegierende, meistens der Nutzer oder ein Service, der Delegierte oder auch ein Zwischenknoten, der im Namen des Delegierenden agieren soll und ein Endsystem, das den Zugriff auf ihre Ressourcen kontrolliert.

In Abbildung 1.36 soll an einem einfachen Beispiel das Prinzip der Delegation veranschaulicht und später erklärt werden.

Beteiligte Komponenten im Beispiel sind:

- Nutzer der auf seine E-Mails zugreifen will, ist der Delegierende.
- Webmail-Server, der Delegierte, der zwischen dem Nutzer und der E-Mail Datenbank agiert.
- Das Endsystem ist die E-Mail Datenbank, in der sich die E-Mails des Nutzers befinden.

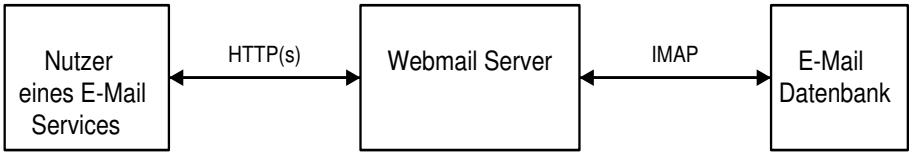

Abbildung 1.36: Einfache Delegation

Vereinfachter Ablauf des Beispiels:

1. Der Nutzer loggt sich bei seinem Webmail-Server ein.
2. Der Webmail-Server weist gegenüber der E-Mail Datenbank nach, dass sich ein Nutzer eingeloggt hat und darf somit im Namen des Nutzers agieren.

Wichtig ist in diesem Beispiel natürlich die Sicherheitspolitik der E-Mail Datenbank. Die erlaubt dem Webmail-Server erst dann auf die E-Mails eines Nutzers zuzugreifen, wenn er nachweisen kann, dass der Nutzer sich erfolgreich authentifiziert hat.

Die Art der Delegation hat großen Einfluss auf die Politiken des Endsystems, denn um eigene Politiken durchzusetzen, braucht man bestimmte Informationen, die in der Delegation enthalten sind. In den meisten Fällen entscheidet man sich deswegen erst, die Politiken zu entwerfen, und dann die Umsetzung der Delegation so anzupassen, dass sie genug Information enthält,

die Politiken sauber durchzusetzen. Beispiele für spezielle Umsetzungen können die SAML-Assertions sein. Auf eine allgemeine Umsetzungsmöglichkeit wird später in diesem Abschnitt unter „Realisierung der Delegation“ eingegangen.

Wie man in Abbildung 1.36 an einem einfachen Beispiel sieht, ist die Behandlung der Delegation von Rechten ein wichtiger Sicherheitsaspekt in Service-orientierten Architekturen. Es stellt sich die Frage, wie ein Service beweisen kann, dass er beauftragt wurde, oder wie überprüft werden kann, ob die Delegation über vertrauenswürdige Instanzen läuft. Auch muss ein System entscheiden, wie es Anfragen von Instanzen behandelt, die mehr oder weniger Rechte besitzen als die Stellen, von denen sie delegiert wurden. Für den Auftraggeber einer Delegation stellen sich ebenso Sicherheitsfragen: Werden die delegierten Rechte nicht missbraucht und kann die Delegation eingeschränkt werden? Diese Fragen sollen anhand verschiedener Konzepte hier näher untersucht werden.

Abhängigkeit der Delegation von der Sicherheitspolitik und die daraus entstehende Sicherheitsrisiken

Wie oben angedeutet, haben Sicherheitspolitiken großen Einfluss an den Inhalt der Nachrichten, die im Verlauf der Delegation ausgetauscht werden. Sicherheitspolitiken haben die Aufgabe, die Zugriffskontrolle für bestimmte Ressourcen durchzusetzen. In diesem Unterabschnitt werden einige Möglichkeiten näher betrachtet.

In vielen Sicherheitspolitiken wird Delegation nicht betrachtet. Bei manchen Datenbanken lautet die Sicherheitspolitik, dass der Zwischenknoten den vollen Zugang zum Endsystem hat und zwar nur mit einer festen Identität, die dem Server fest zugewiesen ist. In diesem Fall wird die Identität des Nutzers entweder gar nicht in die Sicherheitspolitik miteinbezogen oder sie kann der Anwendung als Parameter übergeben werden. Da der Server nicht im Namen eines Nutzers agiert, wird in diesem Szenario auch keine Delegation der Rechte des Nutzers benötigt.

In anderen, ähnlichen Sicherheitspolitiken wird die Identität des Nutzers auf der Ebene eines Sicherheitsmechanismus vom Zwischenknoten an das Endsystem zwar weitergereicht, aber dem Zwischenknoten wird nicht verboten, beliebige Identitäten vorzuweisen, weil z.B. kein Mechanismus zur Überprüfung vorhanden ist. In den Anwendungsprotokollen die das SASL-Framework nutzen, besteht für den Zwischenknoten die Möglichkeit während der eigenen Authentifizierung in das SASL-Feld `authorization id`, die Identität des Nutzer einzutragen. Der Inhalt dieses Feldes wird aber von dem Endsystem nicht überprüft, es wird also kein Nachweis einer Delegation von dem Zwischenknoten angefordert, der bestätigen könnte, dass der Zwischenknoten wirklich im Namen des Nutzers agiert.

Die beiden oben genannten Szenarien bringen dem Endsystem und ihrem Betreiber ein großes Risiko, wenn der Zwischenknoten kompromittiert wurde. Der Angreifer kann sich dann gegenüber dem Endsystem als jeder mögliche Nutzer präsentieren und dementsprechend auch agieren. Wenn sich bei diesem Szenario der Zwischenknoten und das Endsystem unter verschiedenen Sicherheitsdomänen befinden z.B. von verschiedenen Firmen betrieben werden, dann darf dieses Risiko nicht akzeptiert werden.

Mit dem Prinzip des minimalen Rechts (*principle of least-privilege*) kann man das Risiko reduzieren. Bei diesem Prinzip darf auf das Endsystem nur mit dem minimalsten Recht eines Mitglieds in der Delegationskette zugegriffen werden. Das heißt, dass der Zwischenknoten im Namen des Nutzers agiert und nur die Operationen ausführen darf, die auch der aktive Nutzer ausführen darf. Diese Einschränkung der „Macht“ des Zwischenknotens sollte von Sicher-

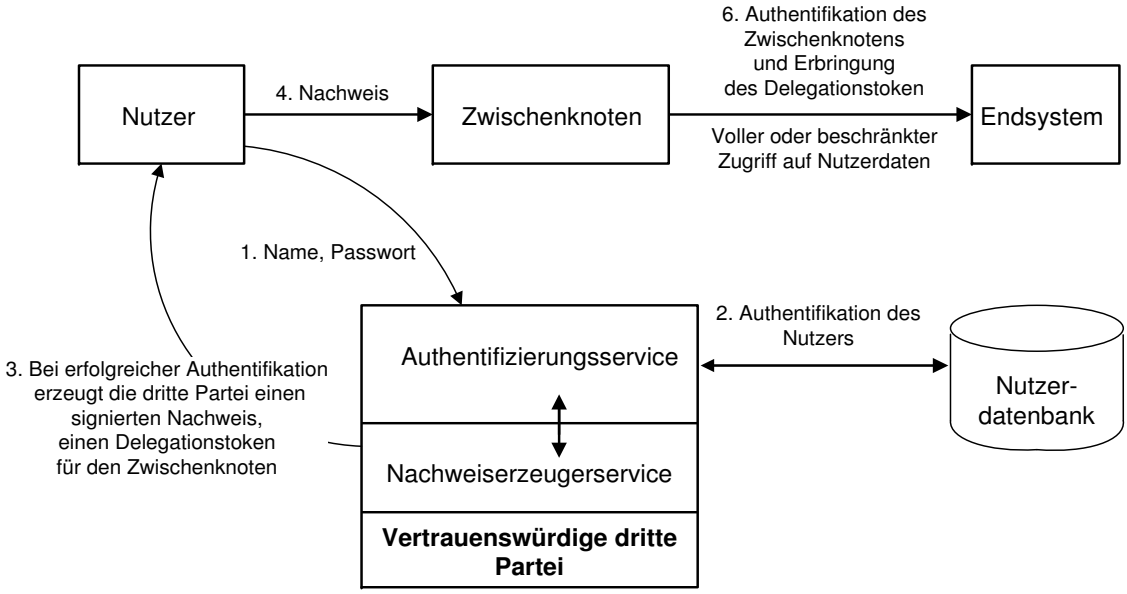

Abbildung 1.37: Authentifikation des Nutzers und eines Zwischenknotens

heitsmechanismen unterstützt werden. Diese Einschränkung kann in verschiedenen Variationen auftreten.

Eine Möglichkeit solche Einschränkung durchzusetzen, ist dem Zwischenknoten den Zugriff auf das Endsystem nur dann zu erlauben, wenn dieser im Namen eines ihm gegenüber authentifizierten Nutzers agiert und für die Authentifizierung des Nutzers auch einen Nachweis vorlegen kann. Es wird also ein Mechanismus zur Erbringung eines solchen Nachweises zwischen dem Zwischenknoten und dem Endsystem benötigt, indem entweder der Zwischenknoten direkt solch einen Nachweis erbringen kann oder einen Verweis, eine URL einer dritten vertrauenswürdigen Partei angibt, bei der das Endsystem die benötigten Informationen erhalten kann. Dieses Szenario ist in Abbildung 1.37 dargestellt.

Falls eine strengere Einschränkung vorgenommen werden soll, kann das Endsystem einen Nachweis vom Zwischenknoten verlangen, in dem explizit stehen soll, dass er vom Nutzer bevollmächtigt wurde, in seinem Namen zu agieren. Diese Einschränkung kann sogar weitere Bedingungen enthalten, wie z.B. die zusätzliche Einschränkung der Operationen, die der Zwischenknoten ausführen kann. Solche Einschränkungen limitieren auch die Möglichkeit eines kompromittierten oder böswilligen Zwischenknotens, unerwünschte Aktionen auszuführen.

Delegation mehrerer Zwischenknoten

Bis jetzt wurden Szenarien mit nur einem Zwischenknoten behandelt. In einer Service-orientierten Architektur kann die Kommunikation allerdings auch über mehrere Zwischenknoten stattfinden. Das Endsystem muss aufgrund seiner Sicherheitspolitik entscheiden, ob es nur dem Nutzer, dem Nutzer und dem letzten Glied oder jedem Glied in der Delegationskette bis zum Nutzer vertraut und dieses Vertrauen in seine Sicherheitspolitik aufnimmt. Dieses Szenario ist in Abbildung 1.38 dargestellt.

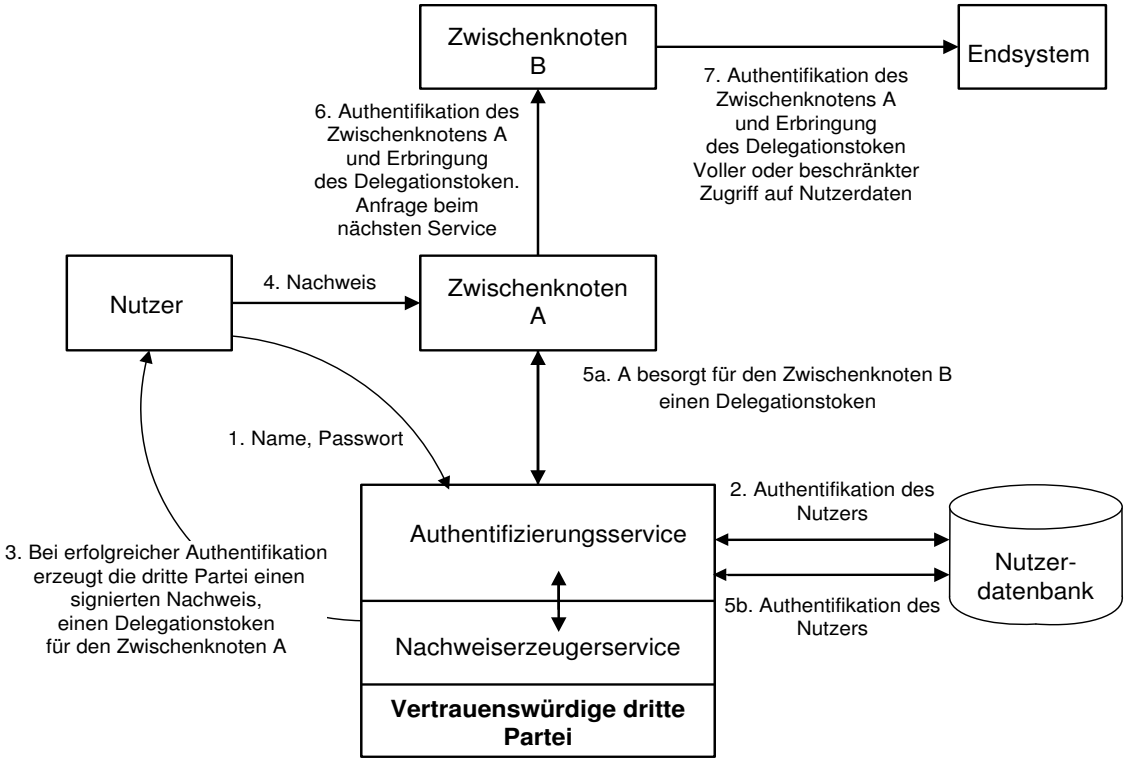

Abbildung 1.38: Delegation aller Kettenmitglieder

Realisierung der Delegation

Es gibt mehrere Möglichkeiten Delegation zu implementieren. Ein allgemeiner Ansatz, der auch in [36], beschrieben ist, verwendet ein *token*, das seinem Eigentümer erlaubt, mit denselben oder eingeschränkten Rechten und Berechtigungen zu arbeiten wie das Subjekt, das das *token* erteilt hat (vgl. Zugriffsausweise aus Abschnitt 1.8.2). Ein Prozess kann ein *token* mit höchstens denselben Rechten und Berechtigungen erzeugen, die er selbst besitzt. Wenn ein Prozess ein neues *token* basierend auf ein aktuelles *token* erzeugt, weist das abgeleitete *token* mindestens dieselben Einschränkungen wie das ursprüngliche auf, möglicherweise auch mehr [49].

Bevor wir das allgemeine Schema für die Delegation erklären, betrachten wir die beiden folgenden Ansätze. Erstens ist die Delegation relativ einfach, wenn eine Person Anna alle kennt, an die sie ihre Rechte weitergeben will. Wenn sie Rechte an Benny delegieren will, erzeugt sie einfach ein Zertifikat, das besagt „Anna weist Benny die Rechte R zu“. Wenn Benny einige dieser Rechte an Charly weitergeben will, fordert er Charly auf, sich an Anna zu wenden und sie um ein entsprechendes Zertifikat zu bitten.

Als zweites Beispiel kann Anna einfach ein Zertifikat erzeugen, das besagt „Der Besitzer dieses Zertifikats hat die Rechte R“. In diesem Fall müssen wir das Zertifikat jedoch gegen unerlaubtes Kopieren schützen. Das Schema von Neuman [36] berücksichtigt diesen Fall und vermeidet gleichzeitig das Problem, dass Anna jeden kennen muss, an den Rechte delegiert werden sollen.

Im Schema von Neuman besteht ein *token* aus zwei Teilen, wie in Abbildung 1.39 gezeigt. Sei A der Prozess, der das *token* erzeugt hat. Der erste Teil des *token* ist die Menge $C = \{R, S^{+}_{token}\}$, die aus einer Menge R von Zugriffsrechten besteht, die an A delegiert wurden,

zusammen mit einem öffentlich bekannten Teil eines Geheimnisses, das verwendet wird, um den Besitzer des Zertifikats zu authentifizieren. Die Verwendung von S^+_{token} wird nachfolgend erklärt. Das Zertifikat trägt eine mit dem privaten Schlüssel K_A erzeugte Signatur $\{C\}^{K_A}$ von A und ist damit gegen Veränderungen geschützt. Der zweite Teil enthält den anderen Teil des Geheimnisses, auch als S^-_{token} dargestellt. Es ist wichtig, S^-_{token} gegen Preisgabe zu schützen, wenn Rechte an einen anderen Prozess delegiert werden.

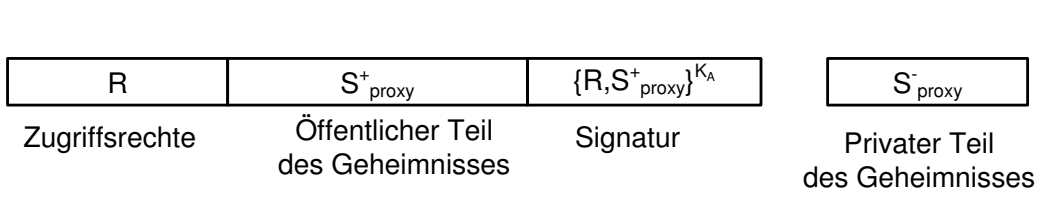

Abbildung 1.39: Die allgemeine Struktur eines *token*, das für Delegation verwendet wird

Man kann das *token* auch wie folgt betrachten: Wenn Anna einige ihrer Rechte an Benny delegieren will, legt sie eine Liste der R an, die Benny besitzen soll. Durch die Signatur der Liste verhindert sie, dass Benny sie abändert. Häufig ist es aber nicht ausreichend, nur eine signierte Liste mit Rechten zu verwenden. Wenn Benny seine Rechte durchsetzen will, muss er möglicherweise beweisen, dass er die Liste von Anna erhalten hat und sie nicht etwa von irgendjemand anderem gestohlen hat. Aus diesem Grund denkt sich Anna eine komplizierte Frage aus (S^+_{Token}), deren Antwort nur sie kennt (S^-_{Token}). Jeder kann die Korrektheit der Antwort ganz leicht überprüfen, wenn er die Frage kennt. Die Frage wird in die Liste eingefügt, unmittelbar bevor die Signatur von Anna erfolgt.

Für die Delegation ihrer Rechte gibt Anna die signierte Liste der Rechte zusammen mit der komplizierten Frage an Benny weiter. Dabei teilt sie Benny auch die Antwort mit, wobei sichergestellt wird, dass niemand sie abfangen kann. Jetzt hat Benny eine von Anna signierte Liste mit Rechten, die er beispielsweise gegenüber Charly beweisen kann, wenn das erforderlich ist. Charly stellt ihm die komplizierte Frage, die sich unten auf der Liste befindet. Wenn Benny die Antwort kennt, weiß Charly sicher, dass es tatsächlich Anna war, die die aufgelisteten Rechte an Benny delegiert hat.

Eine wichtige Eigenschaft dieses Schemas ist, dass Anna nicht konsultiert werden muss. Benny kann sogar entscheiden, die Rechte von der Liste (oder einige davon) an David weiterzugeben. Dafür teilt er David die Antwort auf die Frage mit, sodass David beweisen kann, dass ihm die Liste von jemandem übergeben wurde, der dazu berechtigt war. Anna muss David dazu überhaupt nicht kennen.

Abbildung 1.40 zeigt ein Protokoll für die Delegation und Ausübung von Rechten. Angenommen, Anna und Benny haben einen geheimen Schlüssel $K_{A,B}$, der für die Verschlüsselung von Nachrichten verwendet werden kann, die sie sich gegenseitig senden. Dann sendet zuerst Anna Benny das Zertifikat $C = \{R, S_{proxy}\}^{K_A}$ signiert mit K_A. Es ist nicht erforderlich, diese Nachricht zu verschlüsseln: sie kann als Klartext gesendet werden. Nur der private Teil des Geheimnisses muss verschlüsselt werden, in Abbildung 1.40 in der Nachricht 1 als $K_{A,B}(S^-_{proxy})$ dargestellt.

Angenommen, Benny will eine Operation für ein Objekt ausführen, das sich auf einem be-

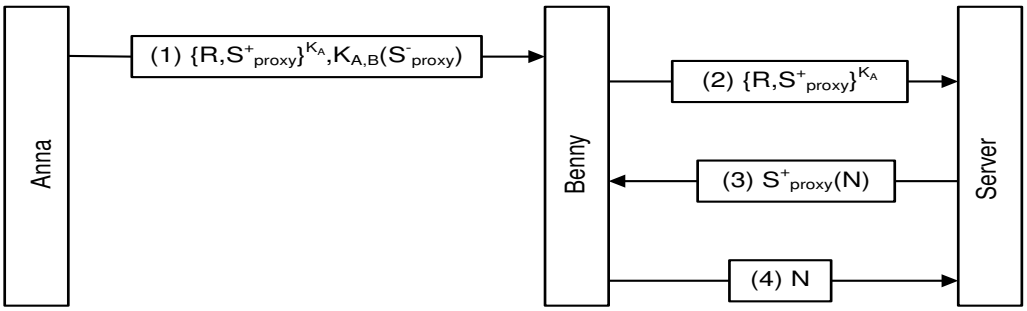

Abbildung 1.40: Delegation eines Tokens und das Challenge-Response-Verfahren

stimmten Server befindet. Nehmen wir weiterhin an, dass Anna die Berechtigung besitzt, diese Operationen auszuführen, und dass sie diese Rechte an Benny delegiert hat. Damit übergibt Benny seine Berechtigungen an den Server in Form des signierten Zertifikats C.

An dieser Stelle kann der Server überprüfen, dass C nicht verfälscht wurde: Jede Änderung an der Rechteliste oder der komplizierten Frage wird erkannt, weil beide gemeinsam von Anna signiert wurden. Der Server weiß jedoch nicht, ob Benny der ordnungsgemäße Eigentümer des Zertifikats ist. Um dies zu überprüfen, muss der Server das Geheimnis verwenden, das ihm zusammen mit C übergeben wurde.

Es gibt mehrere Möglichkeiten, S^+_{Token} und S^-_{Token} zu implementieren. Angenommen, S^+_{Token} ist ein öffentlicher Schlüssel, und S^-_{Token} ist der entsprechende private Schlüssel. Z kann dann Benny auffordern, ihm eine *nonce* N zu senden, die mit S^+_{Token} zu N_{enc} verschlüsselt wurde. Durch die Entschlüsselung von N_{enc} und die Rückgabe von N beweist Benny, dass er das Geheimnis kennt und damit der ordnungsgemäße Eigentümer des Zertifikats ist. Es gibt noch andere Möglichkeiten, die sichere Delegation zu implementieren, aber die grundlegende Idee bleibt immer dieselbe: Man zeigt, dass man ein Geheimnis kennt.

Kapitel 2

Die SicAri-Plattform

2.1 Überblick der SicAri-Plattform

Professionelle Nutzung heutiger Kommunikations- und Kollaborationsinfrastrukturen darf die Betrachtung passender Sicherheitsmechanismen nicht außer Acht lassen. Die Eigenschaften dieser Mechanismen sollen einerseits Sicherheit bieten, andererseits dürfen sie nicht lästig für den Benutzer sein, weil sie sonst nicht benutzt werden. Das allgemeine Ziel des SicAri-Projektes ist, ein Konzept einer Sicherheitsplattform und ihrer Werkzeuge für ubiquitäre Nutzung des Internets zu entwerfen und zu realisieren. Die SicAri-Plattform unterstützt eine Menge von Anwendungen, die den Nutzern in transparenter Weise verschiedene Sicherheitsdienste anbieten.

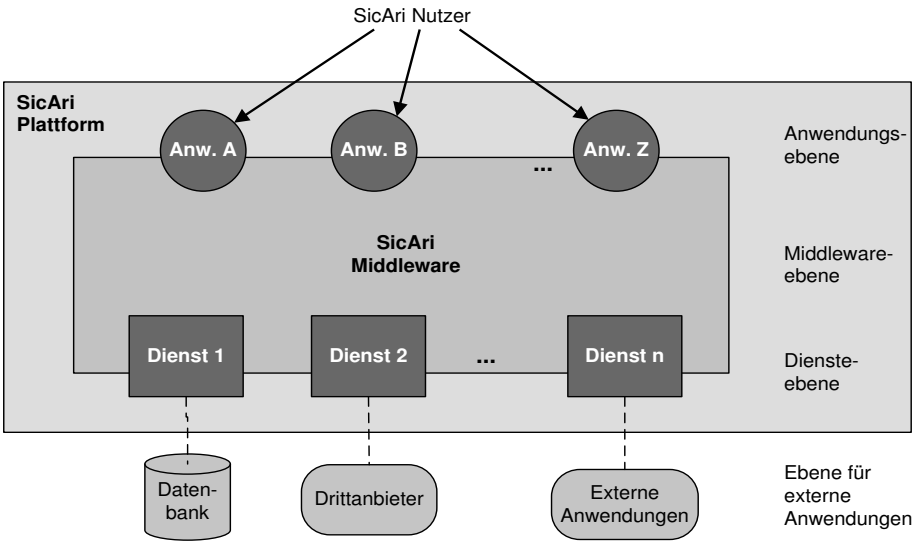

Abbildung 2.1: High-Level der SicAri-Plattform

Grundlage der SicAri-Architektur ist eine offene Middleware und Diensteplattform, deren Funktionalität sich aus den Anwendungsszenarien und speziellen Anforderungen ableitet. Dazu zählen insbesondere Sicherheit, funktionale Anforderungen, Integration externer Schnittstellen und

Wartbarkeit. Abbildung 2.1 gibt eine vereinfachte Übersicht über die generische, mehrschichtige SicAri-Plattform-Architektur.

In der obersten Schicht befindet sich die Anwendungsebene. Auf die Anwendungen, die in dieser Ebene definiert sind, können Nutzer direkt zugreifen. Die Anwendungen ihrerseits verwenden zur Erfüllung ihrer Aufgaben, die von der Service-Ebene angebotene Basis- und Anwendungsdienste. Diese Dienste integrieren auf der einen Seite externe Datenbanken und Software von Drittherstellern, auf der anderen Seite bieten sie die Basissicherheitsdienste wie z.B. Zugriffskontrolle und Authentifikation. Die Middleware-Ebene ist für sichere und transparente Integration und für Kommunikation zwischen den Anwendungen und Diensten zuständig. In den weiteren Unterabschnitten wird diese Architektur im Detail beschrieben.

2.2 Anwendungsbereich der SicAri-Plattform

Die Hauptfunktion der SicAri-Plattform ist, den Anwendungen der Nutzer eine Schnittstelle zu bieten, die der Anwendung ermöglicht, auf die von der Plattform angebotenen Services zuzugreifen. Zusätzlich bietet die Plattform zusammen mit der Middleware eine Kommunikationsinfrastruktur zwischen verteilten SicAri-Komponenten. Das heißt, dass externe Komponenten und Instanzen der Plattform flexibel hinzugefügt und wieder entfernt werden können.

Alle Sicherheitsaspekte der Plattform, wie z.B. Zugriffskontrolle auf lokale oder entfernte Services, werden durch eine Sicherheitspolitik (*security policy*) geregelt und überprüft. Bei der Interaktion mehrerer Plattformen untereinander, befinden sich alle Plattformen innerhalb einer Infrastruktur und haben eine gemeinsame Sicherheitspolitik. Jede Plattform hat eine eigene lokale Komponente, die für die Durchsetzung der Politik zuständig ist (*Policiy Enforcement Point (PEP)*).

Die SicAri-Plattform und ihre Komponenten verwenden Java-Technologie. Java-Technologie stellt Entwicklungswerkzeuge für mobile Geräte (J2ME), PCs (J2SE) bis hin zu skalierbaren, verteilten Umgebungen (J2EE) bereit. Die aktuelle Version der SicAri-Plattform setzt mindestens J2SE voraus, um den kompletten Funktionsumfang zu nutzen, weil die Unterstützung kryptographischer Funktionen auf mobilen Geräten (J2ME) noch sehr beschränkt ist.

2.3 Architektur der Plattform

Das Design der Plattformarchitektur wurde im Wesentlichen von den Anforderungen an die Plattform und ihren Werkzeugen beeinflusst, die für die Verwaltung und Durchsetzung der Sicherheitspolitiken zuständig sind. Abbildung 2.2 stellt eine andere Sicht auf die mehrschichtige Plattform dar.

Die Plattform hat den SicAri-Kernel als Basis, der auf die *Java Virtual Machine (JVM)* aufsetzt. Der Kernel besteht aus drei Komponenten, nämlich Shell, Environment und Security Manager, die im nächsten Abschnitt näher beschrieben werden. In Abbildung 2.2 kann man ebenfalls erkennen, dass die Anwendungen der SicAri-Nutzer in die Plattform integriert werden und auf der Plattform laufen können. Diese Anwendungen können auf die Basis- und Anwendungsdienste der Plattform zugreifen.

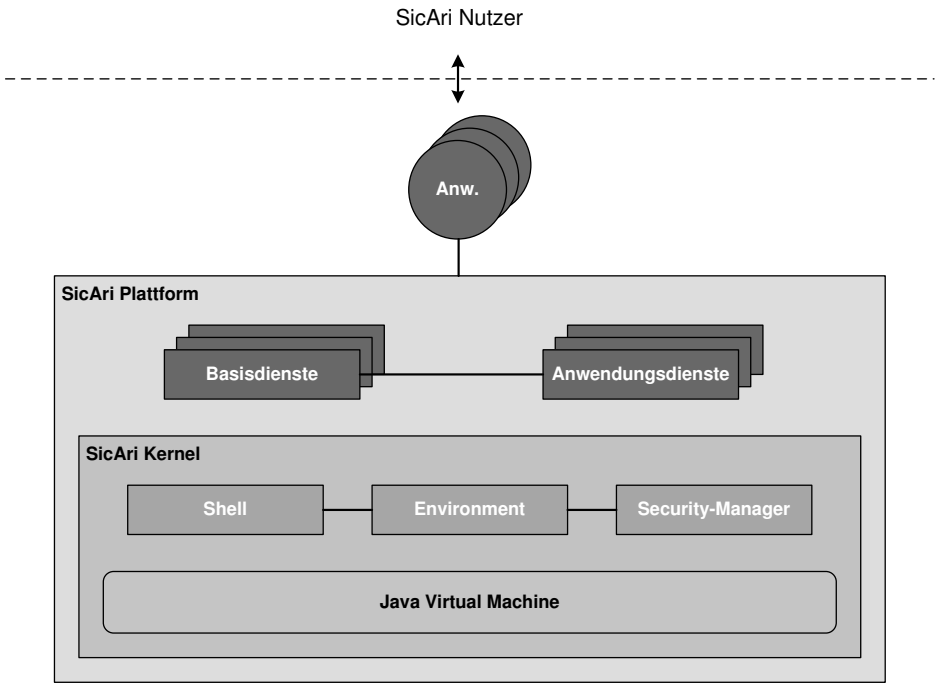

Abbildung 2.2: Schichten der SicAri-Plattform

Die SicAri-Plattform hat folgende positive Eigenschaften:

Minimalistisches Design: Das Design der Plattform wurde einfach und übersichtlich gestaltet, sodass kein allzu komplexes System entsteht. Dadurch wird die Verwaltung der Plattform vereinfacht und der Einsatz von Sicherheitsmechanismen minimiert.

Modularität und Wiederverwendbarkeit: Die Dienste sind unabhängige Komponenten des Systems. Sie können als Einzelkomponente oder als eine, in einem System integrierte Komponente genutzt werden. Diese Unabhängigkeit vereinfacht Integration neuer Dienste und ermöglicht schnelles Austauschen bereits bestehender Dienste.

Erweiterbarkeit und Wartbarkeit: Durch die Eigenschaft der Modularität kann die Funktionalität der Plattform leicht erweitert und gewartet werden.

Sicherheit: Da die Betrachtung der Sicherheit schon beim Beginn des Designs eine tragende Rolle gespielt hat, sind die Sicherheitsfunktionen in der Plattform transparent integriert.

2.3.1 SicAri-Kernel

Der Begriff Middleware ist ein Synonym für Kernel. Der SicAri-Kernel (oder einfach nur Kernel) besteht aus den drei in Java implementierten Komponenten Shell, Environment und Security Manager. Zusammen bieten sie Funktionalitäten wie Bootstrapping, Konfiguration, Verwaltung lokaler Dienste (Registrierung/Suchen) und eine konsistente Sicherheitsumgebung in Form von

Authentifikation und Zugriffskontrolle. Authentifikation dient als Basis für das Arbeiten mit der Plattform, weil die Plattform nur mit authentifizierten Subjekten kommuniziert und nur diesen ihre Dienste anbieten kann. Auf die Authentifikation wird später in Abschnitt Authentifikationsmanager 2.7 eingegangen. Hier werden die Komponenten beschrieben, aus denen der Kernel besteht.

Shell: die Shell ist die Interaktionsebene für den Administrator der Plattform. Die Shell stellt dem Administrator Variablen und die Syntax bereit, um Kontrollsequenzen wie Schleifen und Bedingungen, aber auch eigene Kommandos zu definieren. So wird dem Administrator ermöglicht, während der Laufzeit Dienste zu starten, zu konfigurieren und zu überwachen. Die Shell bietet auch Interaktion mit entfernten Rechnern an. Hierzu wird ein SSH-ähnlicher Dienst implementiert, der einem Client erlaubt, sich in die Plattform einzuloggen. Dieser Client authentifiziert implizit den Administrator der Plattform durch seinen privaten Schlüssel.

Environment: das Environment ist für die Verwaltung der Dienste zuständig. Das Environment ist ein hierarchisch strukturierter Namensraum, in dem Dienste unter einem bestimmten Pfad publiziert werden können. Analog zu einem Dateisystem kann auch hier der Zugriff auf Unterhierarchien oder Dienste beschränkt werden. Die Beschränkungen können durch Verteilung von Rechten oder bestimmten Genehmigungen, für Lesezugriffe (*lookup*) oder für zwei Arten von Schreibezugriffen (*publish* und *retract*) realisiert werden.

Das Environment ermöglicht Kommunikation zwischen Diensten untereinander, Kernel und Dienst sowie auch Dienst und Anwendung und ist damit der perfekte Ort für die Überprüfung und Durchsetzung der Sicherheitspolitik.

Eine weitere Eigenschaft des Environment ist, dem Administrator beim Publizieren eines Dienstes zu erlauben, einen speziellen Proxy für diesen Dienst zu spezifizieren, welcher die Dienstinteraktion übernimmt. Die angebotenen Proxytypen werden nun vorgestellt:

Plain Proxy: Die erste Art von Proxy heißt Plain Proxy. Dieser Proxy leitet einen Methodenaufruf zum eingekapselten Dienst-Objekt innerhalb des gleichen *thread* weiter. Sollte ein Dienst wieder entfernt werden, wird dieser *thread* durch ein privilegiertes Signal eingeleitet. Der Proxy löscht die Referenz auf das eingekapselte Objekt und gibt es für den *garbage collector* frei, falls keine weiteren Referenzen auf das Objekt bestehen. Weitere Methodenaufrufe führen zu einer entsprechenden Fehlermeldung, die von dem Proxy geworfen wird. Auf diese Weise kann der Dienst sauber entfernt und ein inkonsistenter Zustand während eines Dienstzugriffes verhindert werden, weil sich die Referenz auf das Proxy und nicht auf das eingekapselte Dienst-Objekt bezieht.

Asynchronous Proxy: Diese Art von Proxy erzeugt für die Bearbeitung der Methodenaufrufe je einen *thread* pro Methodenaufruf. Der *thread* des Anrufers wird blockiert, bis das Resultat der hervorgerufenen Methode vorhanden ist. Der Vorteil ist, dass für den *thread* des Aufrufers ein Timeout eingestellt werden kann, nach dessen Ende der *thread* beendet wird, selbst wenn kein Resultat vorhanden ist.

Security Proxy: Es wird noch ein dritter Proxy (der Sicherheitsproxy) angeboten, der für automatische und transparente Durchsetzung der Sicherheitspolitik verantwortlich ist. Dieser Proxy befindet sich vor sensitiven Objekten und dient als PEP. Durch diesen Proxy wird der Referenz-Monitor-Ansatz realisiert (vgl. Abschnitt 2.4.3).

Noch einmal kurz zusammengefasst: Die ersten beiden Proxies reduzieren die Anfälligkeit der Dienste durch *Denial of Service (DoS)*-Angriffen und bieten eine klare Trennung der Sicherheitskontexte zwischen dem Aufrufer und dem Aufgerufenen. Durch den dritten Proxy wird die Zugriffskontrolle realisiert.

Security Context (Sicherheitskontext): zusätzlich zu den Sicherheitsmechanismen, die von Shell und Environment angeboten werden, bietet der SicAri-Kernel einen impliziten und expliziten Sicherheitskontext für Dienste, die von der Shell gestartet wurden bzw. für die Benutzer, die sich in die Plattform eingeloggt haben. Der Sicherheitskontext ist die Voraussetzung für implizite Durchsetzung der Sicherheitspolitik. Es existieren zwei Möglichkeiten, einen Sicherheitskontext für einen Dienst einzurichten.

Die erste Möglichkeit ist, einen Dienst mit eigenem Sicherheitskontext zu starten, der einen eigenen `ClassLoader` enthält und falls benötigt, innerhalb einer neuen `ThreadGroup` läuft. Allgemein werden Dienste durch den bzw. im Namen des Administrators gestartet und erben somit seinen Sicherheitskontext in Form von Rechten und Genehmigungen. Mit diesem Sicherheitskontext eines Dienstes ist es grundsätzlich möglich, Zugriffsbeschränkungen zu definieren.

Die zweite Möglichkeit ist es, dass der SicAri-Sicherheitsmanager die Sicherheitspolitik implizit, in Abhängigkeit vom aufgerufenen Dienst des Nutzers, durchsetzt, indem er die Entscheidung an ein spezielles Modul delegiert (vgl. Abbildung 2.3).

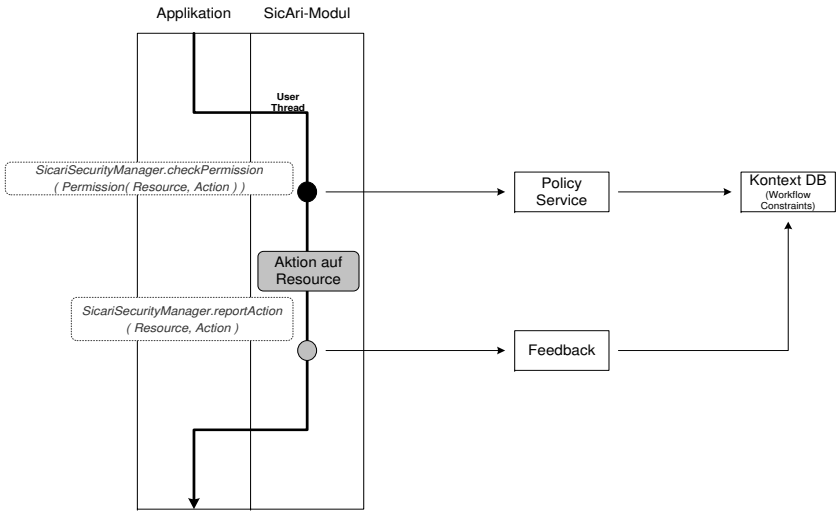

Abbildung 2.3: Zugriffskontrolle und Sessionkontext

2.4 Sicherheitsarchitektur der SicAri-Plattform

Die Sicherheitsarchitektur der SicAri-Plattform verwendet eine Menge von Software- und Hardwareeinheiten, welche die Nutzung sensitiver Daten schützen und kontrollieren. Diese Daten können Dateien oder Datenbanken, Services wie E-Mail oder Drucker sowie sicherheitsbezogene Informationen wie Passwörter und Zertifikate sein. Wie die meisten Sicherheitsarchitekturen ist die SicAri-Architektur mittels APIs beschrieben und versteckt die interne Struktur.

Da sicherheitsbezogene Funktionen, die mit sensitiven Daten arbeiten, die Architektur beeinflussen, muss Sicherheit in jedem Aspekt des Designs betrachtet werden. Dieser Abschnitt gibt einen Überblick über den Sicherheitsrahmen des SicAri-Kernels und seiner Subsysteme.

2.4.1 Überblick über die Komponenten des SicAri-Sicherheitsrahmens

Die SicAri-Plattform spezifiziert für alle sicherheitsbezogenen Angelegenheiten eine Sicherheitspolitik. Diese Politik umfasst z.B. Aussagen in Bezug auf Vertraulichkeit für bestimmte Zugriffsgenehmigungen oder für bestimmte Nachrichten. Alle sensitiven Operationen werden durch die Sicherheitspolitik überprüft. Zusätzlich können einzelne Komponente der Plattform selbst eine Abfrage der Sicherheitspolitik machen, um Rechte an Nutzer zu vergeben.

Der SicAri-Sicherheitsrahmen für die Politik besteht aus mehreren Komponenten wie Politikdurchsetzung (`Policy Enforcement`), Politikentscheidung (`Policy Decision`), Politikanfrage (`Policy Query`), Politikvermittlung (`Policy Provisioning`), Politikadministration (`Policy Administration`) und Politikspezifikation (`Policy Specification`), die in Abbildung 2.4 dargestellt sind. Dieser Ansatz wurde aufgrund der Anforderung gewählt, Politikmechanismen und Politikspezifikation zu trennen. Die genaueren Beschreibungen der Sicherheitskomponenten werden in Abschnitt 2.6 dargestellt.

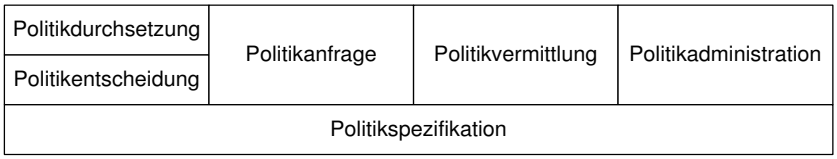

Abbildung 2.4: Komponenten des SicAris Sicherheitsrahmens

2.4.2 RBAC als Sicherheitsmodell

Der SicAri-Sicherheitsrahmen basiert auf zwei Basismechanismen, nämlich Authentifikation und Zugriffskontrolle, die in Abschnitt 1.5 und 1.8 bereits beschrieben wurden. Innerhalb von SicAri wird kein neues Zugriffsmodell vorgestellt. Es wird der RBAC-Mechanismus eingesetzt, der in Abschnitt 1.8.3.1 beschrieben wurde.

2.4.3 Referenz-Monitor-Ansatz

Der Referenz-Monitor-Ansatz wurde schon in Abschnitt 1.8.2 beschrieben. Dieser Ansatz besteht aus einer Politikentscheidungskomponente (*Policy Decision Point (PDP)*) und aus einer oder mehreren PEPs. In der SicAri-Plattform sind diese zwei Komponenten vollkommen unabhängig von der Politikspezifizierungskomponente (*Policy Specification Point (PSP)*). Dies ermöglicht das Modifizieren der PSP, ohne dabei den Kernel anpassen zu müssen. Abbildung 2.5 zeigt den SicAri-Referenz-Monitor.

Durch das Auftrennen des Politikmoduls in PDP und PEP, wird das Hinzufügen zusätzlicher PEPs für Anwendungen aller Art ermöglicht. Jedoch verwenden alle PEPs dieselbe Auswertungslogik einer PDP.

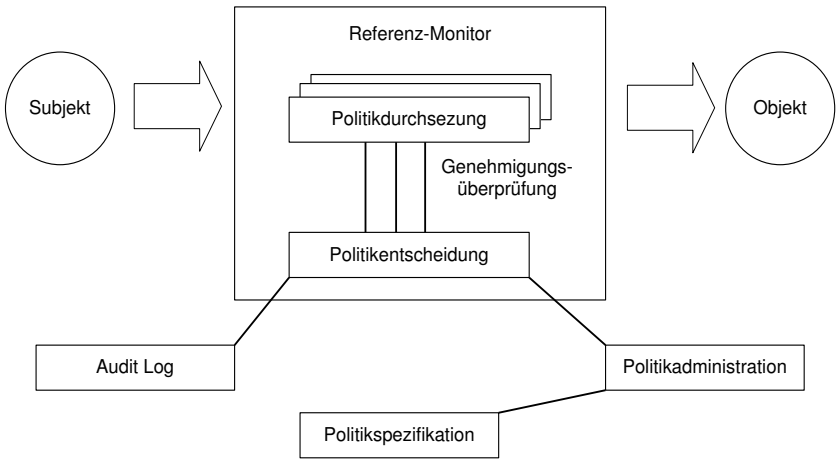

Abbildung 2.5: Zugriffskontrolle mittels Reference Monitor

In Abbildung 2.5 sind noch drei weitere in SicAri verwendete Komponenten dargestellt, die in Beziehung zum Referenz-Monitor stehen. Die *audit log* Komponente ist für das Logging zuständig. Geloggt werden alle Anfragen und die zugehörigen Entscheidungen, die von der PEP getroffen wurden. Die PSP enthält die maschinenleserliche Präsentation der Sicherheitspolitik. Da die Sicherheitspolitik modifiziert werden kann oder manchmal muss, gibt es die Politikadministrationskomponente *(Policy Administration Point (PAP)*), die für die Verwaltung der Sicherheitspolitik zuständig ist.

2.5 Kommunikation in der Plattform

Instanzen der SicAri-Plattform laufen auf vielen Computern. Dieser Abschnitt beschreibt die Interaktion zwischen den Instanzen. In einer typischen SicAri-Infrastruktur existieren verschiedene Arten von Instanzen, nämlich:

Computer, Laptop oder ein PDA der Benutzer, die beim Starten der SicAri-Plattform den SicAri-Kernel und ein paar Dienste installiert haben.

Auf mobilen Geräten wie einem Handy ist es nicht möglich, den SicAri-Kernel zu installieren, weil dafür die Ressourcen fehlen. Trotzdem kann man mit diesen Geräten SicAri-Dienste nutzen, die auf anderen Maschinen laufen.

Die dritte Gruppe bilden die Server, auf denen die zentralen Dienste laufen.

2.5.1 Überblick

Die Kommunikation zwischen den SicAri-Plattformen erfolgt immer in zwei Schritten. Im ersten Schritt wird der Dienst lokalisiert und im zweitem aufgerufen. Wenn der Dienst lokal zu finden ist, dann wird der erste Schritt von Environment übernommen. In einer verteilten Umgebung stützt sich SicAri auf das WS-Framework, das in Abschnitt 1.3 beschrieben wurde, in dem die SicAri-Dienste zu SicAri-Webservices werden. So wird ein UDDI-Verzeichnis für

die Registrierung und zum Aufsuchen der Dienste bzw. der Webservices verwendet. WSDL wird für die Beschreibung der Webservices verwendet und SOAP als Kommunikationsprotokoll eingesetzt. Die im Environment registrierten Dienste sollten automatisch als Webservices im UDDI-Verzeichnis mit einer automatisch erzeugten WSDL Beschreibung registriert werden. Das Ziel dieses Framework ist, den lokalen Servicemanager so zu erweitern, dass für den Nutzer die Verwendung der Webservices auf entfernten Rechnern keinen zusätzlichen Aufwand bedeutet. Die Nutzung der entfernten Webservices geschieht für den Nutzer transparent. Im nächsten Unterabschnitt wird beschrieben, wie eine Nachricht in dem SicAri-WS-Framework bearbeitet wird.

2.5.2 SicAri-WS-Framework und Apache Axis

Die Kommunikation zwischen den SicAri-Instanzen stützt sich auf eine Erweiterung der Apache-Axis-Implementierung [1] von SOAP. Axis ist ein erweiterbares Framework zur Bearbeitung von SOAP-Nachrichten. Module gliedern Axis in Teilsysteme, die verschiedene Aufgaben übernehmen. Die Aufteilung der Aufgaben in Module in Verbindung mit Konfigurationsdateien ermöglicht, eine Erweiterung ohne Änderungen an Axis selbst vorzunehmen. Von grundlegender Bedeutung sind Nachrichten und Handler.

Nachrichten durchlaufen in Form eines `MessageContext` die einzelnen Handler. Der `MessageContext` ist eine Struktur, die Referenzen auf die Anfrage- und Antwortnachricht sowie auf eine Reihe von Attributen beinhaltet (siehe Abbildung 2.6). Der Zugriff auf die Anfrage, die Antwort oder weitere Eigenschaften wie z.B. die Art des Transports, erfolgt in den Handlern über den `MessageContext`.

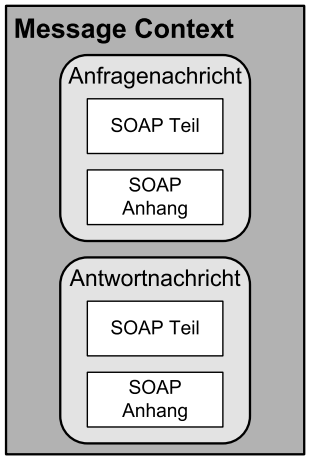

Abbildung 2.6: Der MessageContext

Die Verarbeitung der Anfragenachricht und das Erzeugen der Antwortnachricht wird von Handlern durchgeführt. Über die `invoke()` Methode bekommt ein Handler einen `MessageContext` übergeben, auf den er in seinem Verarbeitungsschritt zugreifen kann.

Mehrere Handler können in einer Kette zusammengefasst werden. Eine Handler-Kette bietet die gleiche Schnittstelle wie ein Handler und kann eine geordnete Menge von Handlern enthalten.

[1] http://ws.apache.org/axis/

Abbildung 2.7 zeigt eine Kette aus drei Handlern.

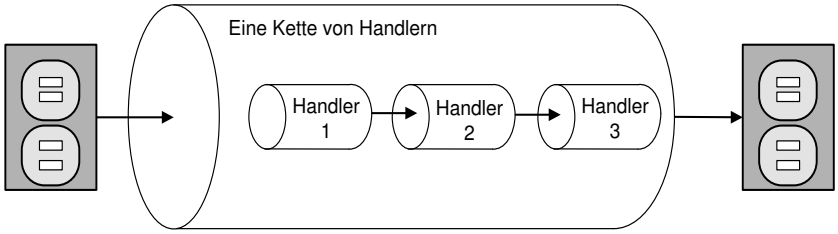

Abbildung 2.7: Eine Kette von Handlern

Die Verarbeitung von Anfragen erfolgt über Filterketten, die sich aus Handlern zusammensetzen. Client und Server bestehen immer aus einer *Axis engine*, welche die Handler-Ketten Service, Transport und Global umfasst.

Die Verkettung der Handler ist flexibel und kann über Konfigurationsdateien angepasst werden. Über selbst implementierte Handler kann in die Verarbeitung von SOAP-Nachrichten eingegriffen werden.

Die Infrastruktur auf Server- und Clientseite unterscheidet sich nur geringfügig. Abbildung 2.8 skizziert den Weg einer Nachricht von der *client engine* zur *server engine*. Im Folgenden wird der Weg, den ein synchroner Aufruf durchläuft, skizziert. Zu Beginn nimmt die Axis *engine* auf Clientseite einen Aufruf der Anwendung entgegen, erzeugt eine Nachricht, die dann die Ketten von Handlern auf der Clientseite durchläuft. Begonnen wird mit der Servicekette, auf welche die globale Kette folgt. Die Transportkette ist anschließend für den Versand der Nachricht zum Server zuständig.

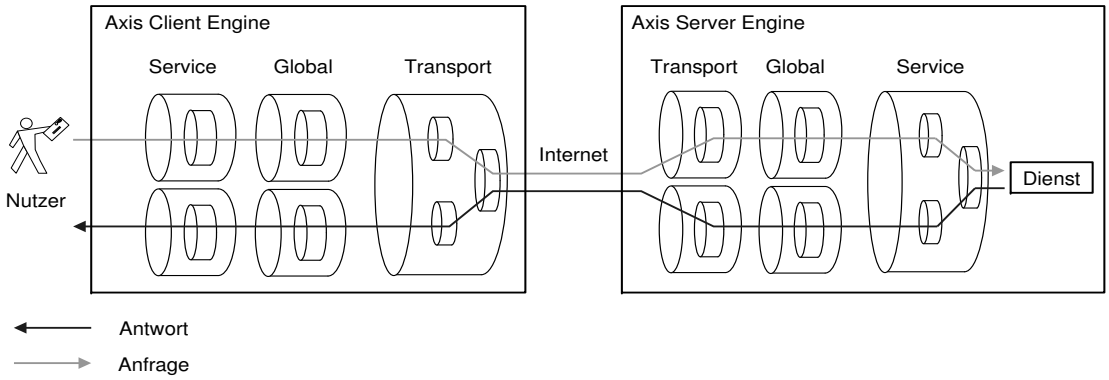

Abbildung 2.8: Der Pfad einer Nachricht

Auf der Serverseite nimmt ein Transport-Handler die Nachricht entgegen und packt diese wieder in einen `MessageContext`, damit die Nachricht die Handler des Servers durchlaufen kann.

Nachdem der Service seine Aufgabe vollbracht hat, kehrt sich der Nachrichtenfluss um. Im `MessageContext`, der zuvor nur die Anfrage beinhaltet hat, wird jetzt zusätzlich eine Antwort referenziert. Die Ketten werden in umgekehrter Reihenfolge durchlaufen. Der Transport Handler auf der Seite des Servers schickt die Antwort zum Client, bei dem ebenfalls der Transport Handler die Nachricht entgegennimmt. Verpackt als `MessageContext` geht es über die

globale Kette zurück zur Service-Kette und danach zum *client code*, durch den der Aufruf ursprünglich initiiert wurde.

2.6 Basisdienste der SicAri-Plattform

2.6.1 Politikdurchsetzung und der Sicherheitsmanager

In diesem Abschnitt wird der PEP der SicAri-Plattform und ihre Realisierung im Detail beschrieben. Zuerst werden die Eigenschaften dieser Komponente beschrieben und dann wird ein Verwendungsszenario dargestellt, an der diese Komponente beteiligt ist.

Bei Zugriffsversuchen auf sensitive Information bewertet der PEP, ob der agierende Nutzer für den Zugriff die benötigten Genehmigungen besitzt.

In SicAri läuft der PEP im Hintergrund und kann von keiner Komponente umgegangen werden. Somit wird jeder Zugriff implizit durch den PEP überprüft.

Für die Realisierung der impliziten Zugriffskontrolle wurde in SicAri der Sicherheitsmanager von Java durch einen neuen Sicherheitsmanager ersetzt. Javas standardmäßiger Sicherheitsmanager ist nur in der Lage, einfache Politik durchzusetzen, die den Ansprüchen der SicAri-Plattform nicht genügte. Mit dem Sicherheitsmanager wird auch der PDP und der PSP durch neue ersetzt. Der Sicherheitsmanager delegiert dann die Politikentscheidung zu dem neuen PDP, der anhand der neuen Politikspezifikation eine Entscheidung trifft.

Ablauf der Durchsetzung der Zugriffskontrolle:

1. Der Nutzer fragt eine sensitive Ressource an.
2. Eine SicAri-Komponente, z.B. eine Anwendung die für das Zustellen der Daten zuständig ist, versucht auf die sensitive Ressource zuzugreifen.
3. Der Sicherheitsproxy erhält die Anfrage und beauftragt den Sicherheitsmanager zu überprüfen, ob er den Zugriff erlauben soll.
4. Der Sicherheitsmanager besorgt des Nutzers Identität und Sicherheitskontext.
5. Der PEP formuliert und sendet eine Anfrage an den PDP, welcher die Genehmigung überprüft.
6. Der PDP überprüft die Anfrage und trifft basierend auf der Sicherheitspolitik eine Entscheidung und liefert `erlaubt` oder `verweigert` zurück.

2.6.2 Politikentscheidungskomponente

Da die Entscheidungslogik aus dem PEP herausgenommen wurde, werden verschiedene Auswertungsmodelle unterstützt wie z.B. eine einfache Zugriffskontrollliste, ein rollenbasiertes Modell oder ein erweitertes rollenbasiertes Modell, das z.B. Kontextinformationen betrachtet. Zusätzlich können auch verschiedene PEPs mit derselben PDP verwendet werden.

Das Ziel des PDP ist es, eine Entscheidung aufgrund ihr vorliegender Informationen zu treffen. Einerseits bezieht sie Information aus der Sicherheitspolitik, anderseits enthalten die Anfragen die noch benötigten Informationen.

Abhängig vom verwendeten Modell muss eine Anfrage mindestens folgende Angaben beinhalten:

user ID: eine eindeutige Identifikation des Nutzers innerhalb der SicAri-Platform.

resource ID: eine eindeutige Identifikation der Ressource, auf die der Nutzer zugreifen will.

operation ID: gibt die Art des Zugriffs an (z.B. lesend, schreibend)

In erweiterten Modellen werden unter Umständen zusätzliche Informationen benötigt, die in einem Sicherheitskontext gespeichert und an die `user ID` gebunden werden können:

activatedRoles: die aktivierte Rolle des Nutzers kann oder muss übergeben werden, wenn die Politik die Trennung der Verpflichtungen (*separation of duty*) in die Entscheidung einbezieht.

date and time information: Datum- und Zeitinformationen können gebraucht werden, falls Zeitbedingungen in der Politik spezifiziert werden sollen.

accessed objects: Informationen können in Form einer Liste von Objekten, auf die ein Nutzer zugegriffen hat, zum Schutz des Informationsflusses in die Politik einfließen.

Wenn der PDP eine Entscheidung getroffen hat, schickt er das Ergebnis, `true` für erlaubt und ansonsten `false`, zum Aufrufer zurück.

Der PDP wird nur durch den SicAri-Sicherheitsmanager angefragt, weil dieser aktuell der einzige PEP ist. Der architektonische Ansatz erlaubt aber das Einsetzen weiterer PEPs, die Gebrauch von der PDP machen.

2.7 Authentifikationsmanager

Der Authentifizierungsprozess stellt die Bindung zwischen der Identität des Subjektes oder Objektes und seinen Eigenschaften fest. Im SicAri-Kontext wird die Identität eines Nutzers mit geheimem Wissen wie z.B. Passwort beim Einloggen nachgewiesen. Nach einer erfolgreichen Authentifizierung wird dem authentifizierten Nutzer eine `session ID` zugewiesen, um zwischen aktiven und passiven Nutzern unterscheiden zu können.

Dieser Dienst arbeitet eng mit zwei weiteren Diensten zusammen. Dies ist zum einen der `Identitätsmanager`, der für Speicherung und Verwaltung der Nutzeridentitäten zuständig ist, und zum anderen `KeyMaster`, der die privaten Schlüssel der Nutzer aufbewahrt und für kryptographische Prozesse des Nutzers zuständig ist.

Der Authentifikationsmanager befindet sich sowohl auf den lokalen Plattformen als auch auf einer zentralen Plattform. Der zentrale Authentifikationsmanager basiert auf JAAS und bietet in der ersten Version dem Nutzer drei Authentifizierungsmöglichkeiten. Diese können mittels Name und Passwort, mittels eines Softtoken oder mittels einer Smartcard vollzogen werden. Auf der zentralen Plattform bietet der Authentifikationsmanager folgende Dienste:

- Verwaltung der Authentifikation der Nutzer.
- Lieferung verifizierter Zertifikate.
- Speicherung der `session IDs`.

Auf einer lokalen Plattform hat der Authentifikationsmanager folgende Aufgaben:

- Agieren als Client für entfernte Aufrufe zum zentralen Dienst.
- Speicherung der Ausweise der Nutzer, die sich lokal an dieser Plattform angemeldet haben.
- Aufrufen kryptographischer Funktionen im Namen der Nutzer.

Wie die Authentifikation im Detail vollzogen wird, welche Komponenten darin beteiligt sind und wie die Kommunikation zwischen diesen Komponenten aussieht, ist ein Teil meiner theoretischen Ausarbeitung und wird in Abschnitt 3 beschrieben.

2.8 Identitätsmanagement

Das Identitätsmanagement einer SicAri-Infrastruktur besteht zum einen aus einer zentralen Administrationsinstanz, die für die Verwaltung der Identitäten in einer Datenbank zuständig ist und zum anderen aus dem Identitätsmanager, der lokal auf jeder Plattform vorhanden ist und zur Laufzeit benutzt werden kann. Abbildung 2.9 zeigt die Komponenten und ihre Beziehungen zueinander. In dieser Abbildung ist eine optionale Komponente `KeyMaster` zu sehen, die für die Verwaltung privater Schlüssel der Nutzer zuständig ist, falls vorhanden. Im Folgenden werden diese Komponenten näher beschrieben.

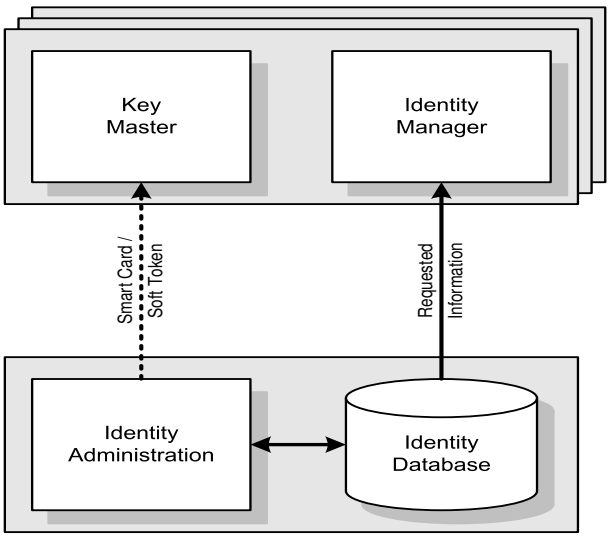

Abbildung 2.9: Komponenten des Identitätsmanagements

Administrationsinstanz

Die Aufgaben der Administrationsinstanz sind, neue Nutzer zu registrieren, ihre Eigenschaften bei Änderung anzupassen und gegebenenfalls die Registrierung zu löschen. Alle Eigenschaften der Nutzer werden in einer LDAP Datenbank gespeichert. Neben den Nutzern werden auch die Plattformadministratoren in derselben Datenbank untergebracht. Insgesamt wird zwischen drei Typen von Nutzern unterschieden, nämlich:

- Plattformadministrator: Jede SicAri-Plattform wird von einem ihr zugehörigen Plattformadministrator gestartet. Die privaten Schlüssel des Administrators werden während einer abgesicherten Kommunikation für eine eindeutige Identifikation der SicAri-Plattform verwendet.
- Service User: Um die Rechte der Basisdienste einzuschränken, werden die einzelnen Basisdienste der Plattform nicht direkt durch den Plattformadministrator, sondern durch den zugehörigen *service user* gestartet. So werden jedem *service user* bestimmte Rechte zugewiesen, die nur eine Teilmenge der Rechte eines Plattformadministrators enthalten. Beim Initialisierungsprozess darf der Plattformadministrator die Rolle jedes *service user* einnehmen.
- Nutzer (*regular user*): Normalerweise ist der Nutzer eine Person, die mit den SicAri-Services über eine lokale SicAri-Plattform interagiert. In Abhängigkeit der zugewiesenen Rechte, kann diese Person entweder nur bestimmte Services nutzen oder darf zusätzlich Basis- und Anwendungsdienste administrieren.

Identitätsmanager

Die Aufgabe des Identitätsmanager ist es, Informationen über eine Identität an andere Dienste oder Anwendungen zu vermitteln. Deswegen befindet sich dieser Basisdienst auf jeder Plattform und interagiert bei Anfrage ggf. mit der Identitätsdatenbank. Die einzelnen Teilaufgaben sind:

Suche nach Identitäten: sucht mittels Nutzerattributen nach den zugehörigen Identitäten.

Transfer der Identitätsrepräsentationen: bildet die drei globalen Nutzerattribute `distinguish name (dn)`, `user ID` und `e-mail` ineinander ab.

Anfrage der Identitätattributte: stellt zu einer gegebenen `user ID` eine Anfrage bezüglich der Nutzerattribute.

Jede Plattform bietet dem Nutzer die Verwaltung der geheimen Schlüssel und mit den Schlüsseln verbundene kryptographische Funktionen. Für die Verwaltung der Schlüssel ist der `KeyMaster` zuständig. Dieser Dienst ermöglicht es dem Nutzer auf seine privaten Schlüssel, die sich in einem Softtoken auf der Plattform befinden, zuzugreifen. Der Nutzer muss sein Passwort für das Öffnen des Softtoken eingeben. Auf diese Weise verlässt der Schlüssel nie die Plattform und wird vertraulich abgespeichert.

Eine andere Aufgabe des `KeyMaster` ist es, Zertifikate zu verwalten, welchen der Administrator vertraut. Diese Zertifikate werden benötigt, um eine Vertrauensbasis zu haben und neue Zertifikate über die Zertifikatsketten zu validieren.

Kapitel 3

Entwicklung und Implementierung

In den Kapiteln 1 und 2 wurden die Sicherheitsmechanismen und der aktuelle Stand der Plattform-Implementierung vorgestellt. In diesem Kapitel werde ich einige Sicherheitsmechanismen beschreiben, die in die SicAri-Plattform integriert und im Betrieb der Plattform benutzt werden sollen. Die Beschreibung dieser Mechanismen ist in zwei Teile aufgeteilt. Einmal handelt es sich um einen theoretischen Teil, in dem es um den sicheren Austausch von Nachrichten bei Authentifizierung eines Nutzers geht. Dabei werden verschiedene Authentifikationsprotokolle vorgestellt.

Der zweite Teil ist der praktische Teil, in dem es sich um das Ergebnis einer erfolgreichen Authentifizierung handelt, nämlich einerseits um die Erzeugung und Repräsentation eines Ausweises oder eines Sicherheitstokens für den Nutzer. Anderseits geht es auch um das Einfügen und Auslesen dieses Tokens in bzw. aus eine(r) Nachricht.

3.1 Authentifikation in SicAri

Eine SicAri-Plattform bietet dem Nutzer die Möglichkeit, sich lokal oder von einem entfernten Rechner anzumelden. Außerdem hat der Nutzer die Möglichkeit, sich von einem entfernten Rechner erst gegenüber einem zentralen Authentifikationsserver zu authentifizieren und von diesem einen Nachweis, ein Token, über die erfolgreiche Authentifikation zu bekommen. Mit diesem Token kann er dann die Dienste der SicAri-Plattformen nutzen.

Lokale Authentifikation

In der SicAri-Architektur werden einer Plattform generell zwei Authentifizierungsabläufe zur Wahl gestellt, wobei für volle Nutzung aller Dienste beide implementiert werden sollten. Die erste Möglichkeit ist, dass die lokale Plattform die Authentifizierung eines Nutzers durchführt und bei einer erfolgreichen Authentifizierung einen Ausweis, ein so genanntes Sicherheitstoken (im weiteren Verlauf des Kapitels einfach nur Token) ausstellt. In Abbildung 3.1 wird dies veranschaulicht.

Die zweite Möglichkeit besteht darin, den lokalen Authentifikationsmanager wie in Abschnitt 2.7 beschrieben als Proxy zwischen dem Nutzer und dem zentralen Authentifizierungsdienst zu verwenden und damit die Aufgabe der Authentifizierung an den zentralen Authentifizierungsdienst zu delegieren. Bei erfolgreicher Authentifizierung wird in diesem Fall für den Nutzer ein

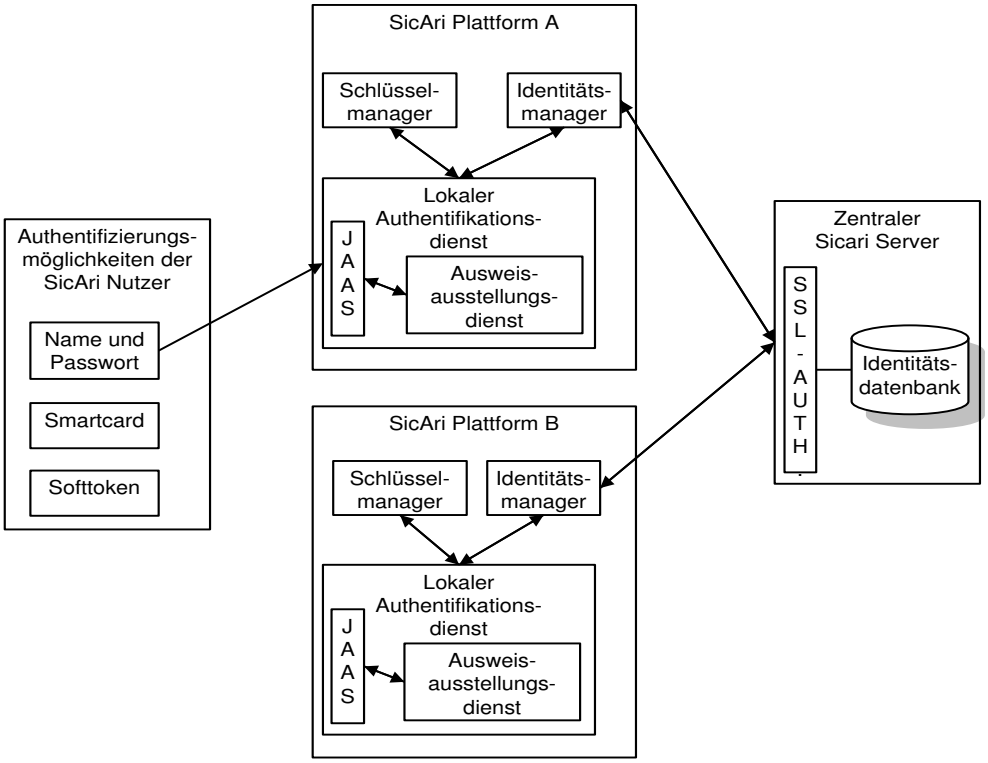

Abbildung 3.1: Authentifikation durch die SicAri-Plattform

globales Token von einer, von allen in der SicAri-Infrastruktur beteiligten Plattformen vertrauten Partei erstellt.

Die Authentifizierungs-Protokolle unterscheiden sich nur in der Bedeutung ihrer Ergebnisse, also in der des ausgestellten Token. Ein von einer lokalen Plattform ausgestelltes Token ermöglicht nur die Nutzung lokaler Dienste. Die Nutzung aller verfügbaren Dienste anderer Plattformen wird nicht erlaubt, weil das Sicherheitstoken die Basis zur Autorisierung bildet und in der Regel nicht von anderen Plattformen als vertrauenswürdig eingestuft wird.

Im Fall der Authentifizierung des Administrators oder allgemein bei einer Authentifizierung eines Nutzers, bei der keine Verbindung zum zentralen Authentifikationsservers besteht, ist dieses Vorgehen die einzige Möglichkeit und sollte nur in diesem speziellen Fall Anwendung finden.

Das globale Sicherheitstoken, das von einem zentralen Authentifizierungsdienst ausgestellt wird, wird von der Plattform, die bei der Authentifizierung als Proxy agiert, sicher verwaltet und bei Nutzung der Dienste anderer Plattformen in der Anfrage mitversendet. Auf diese Weise können die angefragten Plattformen ihre Zugriffskontrolle aufgrund eines vertrauenswürdigen Tokens und der Politikspezifikation durchführen. Im Normalfall soll als Ergebnis einer erfolgreichen Authentifizierung ein globales Token ausgestellt werden, weil dadurch das Single Sign-On innerhalb der gesamten SicAri-Infrastruktur ermöglicht wird.

Authentifikation vom entfernten Rechner

Wenn ein Nutzer sich von einem entfernten Rechner gegenüber einer SicAri-Plattform oder einem zentralen Authentifikationsserver erfolgreich authentifiziert, so muss er selbst das aus-

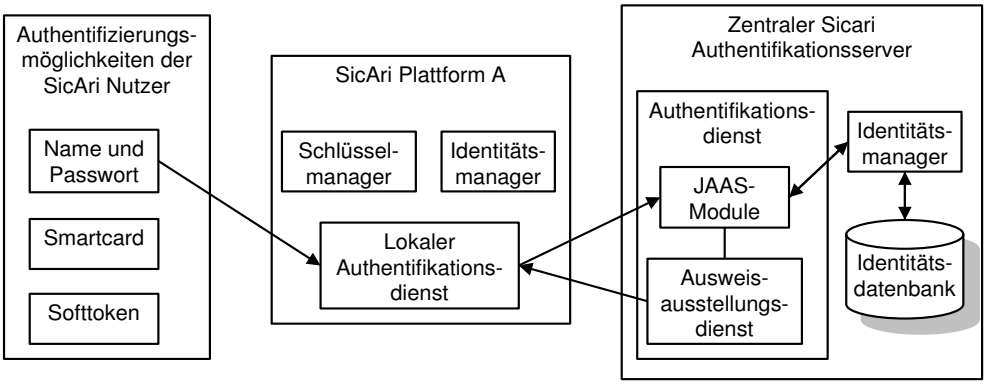

Abbildung 3.2: Authentifikation durch den zentralen Authentifikationsserver

gestellte Sicherheitstoken sicher aufbewahren und vor Diebstahl schützen. Das Sicherheitstoken wird als Nachweis einer erfolgreichen Authentifikation in jedem *request* an eine SicAri-Plattform mitgeschickt, um eine Neuauthentifikation zu vermeiden.

3.1.1 Mögliche Angriffe und Präventionen

Im vorherigen Abschnitt wurde die Bedeutung und die Wichtigkeit des Sicherheitstokens erklärt. Das Sicherheitstoken darf nicht in falsche Hände geraten, weil der Angreifer sonst einen Authentifikationsnachweis eines Nutzers besitzt und sich für diesen Nutzer ausgeben kann. Besonders in SSO-Szenarien kann das Token einen großen Wirkungsbereich haben und muss deswegen vor Angriffen geschützt werden.

Abhören

Das Token muss vertraulich behandelt und verschlüsselt übertragen werden, damit das einfache Abhören der Nachrichten und das Stehlen eines Tokens verhindert wird.

Fälschen

Das Token muss fälschungssicher sein. Fälschungen werden durch die Signatur verhindert. Die ausstellende Instanz erzeugt nur signierte Tokens und nur signierte Tokens werden für die Zugriffskontrolle verwendet. Ein Token kann jeder selbst erstellen, die Signatur einer vertrauenswürdigen Instanz nicht.

Sonstige Angriffe

Weitere Angriffe können gegen die Verschlüsselungs- oder Signaturalgorithmen gerichtet sein oder auf eine unsichere Aufbewahrung des Tokens und andere sensitive Daten abzielen. Auf diese Angriffe wird hier nicht näher eingegangen.

3.2 Authentifikationprotokolle

3.2.1 Globales Token als Ergebnis

Wie in Abbildung 3.2 zu sehen, werden für die Authentifikation eines Nutzers drei Parteien benötigt. Diese sind:

Der Nutzer hat drei Authentifizierungsmöglichkeiten. Er ist im Besitz einer Smartcard oder eines Softtokens oder er kennt ein Geheimnis in Form eines Passwortes, mit dem er sich gegenüber einer SicAri-Plattform authentifizieren kann.

Die SicAri-Plattform gibt dem Nutzer die Wahl einer Authentifizierungsmöglichkeit und agiert während der Authentifikation als Proxy zwischen dem Nutzer und dem zentralen Authentifikationsserver. Nach einer erfolgreichen Authentifikation speichert und verwaltet sie das Token, das vom zentralen Authentifikationsserver für den Nutzer erstellt wurde.

Der zentrale Authentifikationsserver besteht aus dem Authentifikationsdienst, dem Identitätsmanager und der Identitätsdatenbank. Der Authentifikationsdienst führt die Authentifikation mittels eines CR-Verfahren durch und erzeugt einen Token, falls das CR-Verfahren erfolgreich verläuft. Die benötigten Nutzerdaten für die Berechnung der *response* besorgt sich der Authentifikationsdienst aus der Identitätsdatenbank über den Identitätsmanager. Die benötigten Nutzerdaten, die in der Identitätsdatenbank unter dem Nutzernamen abgespeichert sind, ist der Hashwert des Passwortes und/oder das Nutzerzertifikat. Das Nutzerzertifikat enthält keine vertraulichen Daten und kann im Klartext abgelegt werden, muss aber vor Manipulation geschützt werden. Ein Angreifer darf nicht sein eigenes Zertifikat anstelle des Zertifikats vom Nutzer platzieren, weil er sich dann als dieser Nutzer ausgeben kann. Der Hashwert des Passwortes darf dagegen nicht im Klartext abgelegt werden und wird deswegen mit einem geheimen symmetrischen Schlüssel verschlüsselt.

In der Identitätsdatenbank werden Hashwerte von Passwörtern und nicht die Passwörter selbst verschlüsselt abgelegt. Dies hat den Vorteil, dass wenn der geheime Schlüssel oder das Verschlüsselungsverfahren, mit dem die Hashwerte der Passwörter verschlüsselt sind, durch einen Angriff gebrochen wird, dann kann der Angreifer nur auf die Hashwerte zugreifen. Würde er auf die Passwörter zugreifen können, könnte er überall einen Nutzer impersonifizieren, sich also für einen anderen Nutzer ausgeben, wo der Nutzer dieses Passwort verwendet hat.

Außerdem werden in der Identitätsdatenbank alle Namen der Plattformadministratoren und ihnen zugehörigen Zertifikate für die Authentifikation, das Signieren und das Verschlüsseln verwaltet. Somit können die Plattformadministratoren mittels CR-Verfahrens authentifiziert werden. Da der zentrale Authentifikationsdienst drei eigene Zertifikate für die Authentifikation, das Signieren und das Verschlüsseln besitzt, ist es immer möglich, zwischen den SicAri-Plattformen oder den SicAri-Plattformen und dem zentralen Authentifikationsservers eine auf Zertifikaten basierte SSL Verbindung aufzubauen, die Authentizität der Kommunikationspartner sowie Vertraulichkeit und Integrität der Nachrichten gewährleistet.

Als Vorraussetzung aller Authentifikationsmethoden und -abläufe gelten:

1. Die Nutzer, die sich authentifizieren möchten, sind registriert, d.h. dass in der Identitätsbank sind entweder Name und Passwort und/oder Name und das Zertifikat des Nutzers gespeichert.

2. Die Nachrichten, die zwischen der SicAri-Plattform und dem zentralen Authentifikationsserver ausgetauscht werden, sind signiert und verschlüsselt. Die beiden Parteien besitzen die dazu benötigten Schlüssel und sollten sie entweder in der Transportschicht (HTTPs) oder in der Anwendungsschicht (XMLenc und XMLdsig) einsetzen. Da die Kommunikation dieser Parteien über eine Punkt-zu-Punkt-Verbindung, also über keine Zwischenknoten, stattfindet, sollte die Auswahl der Schicht für die Sicherung der Authentifikationsnachrichten egal sein.

3.2.1.1 Authentifikation mittels Name und Passwort

Eine weit verbreitete, aber nicht sehr sichere Authentifikationsmöglichkeit, ist die Authentifikation mittels Name und Passwort. In diesem Abschnitt wird der Einsatz dieser Authentifikation in der SicAri-Umgebung vorgestellt. Der Ablauf der Authentifikation mittels Name und Passwort wird in Abbildung 3.3 dargestellt.

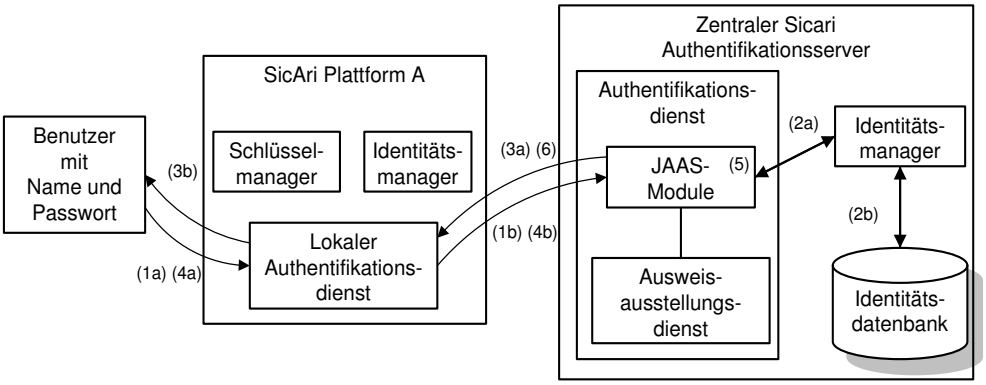

Abbildung 3.3: Authentifikation mittels Name und Passwort

Die einzelnen Schritte des lokalen Authentifikationablaufs:

1. Der Nutzer sitzt vor der SicAri-Plattform und wählt die Passwortauthentifikation aus. Dabei gibt er seinen Namen ein.

2a. Der Authentifikationsdienst schickt eine Anfrage mit dem Nutzernamen an den Identitätsmanager, damit der ihm den zugehörigen Hashwert des Passwortes liefert.

2b. Der Identitätsmanager leitet die Anfrage an die Identitätsdatenbank weiter und falls der Nutzername existiert, liefert er dem Authentifikationsdienst den zugehörigen Hashwert des Passwortes. Ansonsten wird eine entsprechende Fehlermeldung erzeugt, die an den Nutzer weitergeleitet wird.

3. Der Authentifikationsdienst erzeugt die *challenge*, eine Zufallszahl *nonce* und schickt diese an die SicAri-Plattform. Die *nonce* kann z.B. durch $nonce = H(aktuelleZeit|Name)$ gebildet werden.

4. Die SicAri-Plattform fragt den Nutzer nach seinen Passwort, berechnet für den Nutzer die *response* $C = H(H(Pwd)|nonce)$ und schickt sie an den Authentifikationsdienst des zentralen Authentifikationsserver.

5. Der Authentifikationsdienst berechnet $C' = H(H(Pwd)|nonce)$ und vergleicht $C = C'$. Wenn C und C' gleich sind, dann erzeugt der Authentifikationsdienst einen Token.

6. Wurde das CR-Verfahren erfolgreich abgeschlossen, dann schickt der Authentifikationsserver einen Token an die Plattform, sonst wird eine entsprechende Fehlermeldung erzeugt und dem Nutzer angezeigt.

Authentifikation eines Nutzers vom entfernten Rechner

Wenn der Nutzer sich von einem entfernten Rechner authentifiziert, muss er, um Spoofing Angriffe zu verhindern, sicherstellen, dass er wirklich von einem zentralen Authentifikationsserver

authentifiziert wird und nicht mit einem Angreifer interagiert, der versucht, aus den CR-Paaren das Passwort abzuleiten oder gezielt eine CR-Paar-Tabelle anzulegen, um dann zu versuchen sich als Nutzer auszugeben.

In dem bisher vorgestellten Protokoll wird nur der Nutzer authentifiziert, was zu solch einem Spoofing-Angriff führen kann, wenn ein Angreifer sich in die Kommunikation zwischen dem Nutzer und der SicAri-Plattform, die in diesem Szenario als Proxy agiert, einklinken und sich als die SicAri-Plattform ausgeben kann. Um eine gegenseitige Authentifikation zwischen dem Nutzer und dem zentralen Authentifikationsserver zu erreichen und damit einen möglichen Spoofing Angriff zu verhindern, kann der Nutzer alle Nachrichten mit dem öffentlichen Schlüssel des zentralen Authentifikationsservers verschlüsseln.

Eine Alternative wäre auch, im ersten Schritt in der Authentifikationsanfrage des Nutzers zusätzlich eine eigene *nonce* N zu schicken, die in die *challenge* einfließen soll. Diese würde wie folgt aussehen:

$$Authentifikationsanfrage = \{Name, N\}$$

Der Authentifikationsdienst erzeugt dann im dritten Schritt eine Nachricht, die die *response* auf die *challenge* des Nutzers und eine *challenge nonce* S an den Nutzer enthält.

$$Nachricht = \{H(H(Pwd)|N), S\}$$

wobei $response = H(H(Pwd)|N)$

Somit haben beide Parteien gegenseitig nachgewiesen, dass sie das vereinbarte Geheimnis in Form des Passwortes kennen.

Das oben beschriebene Protokoll muss noch um einen Schritt erweitert werden. In diesem letzten Schritt schickt die SicAri-Plattform dem Nutzer das vom zentralen Authentifikationsdienst ausgestellte Token zu. Mit dem Token erhält der Nutzer einen Nachweis über eine erfolgreiche Authentifikation und solange das Token gültig ist, muss der Nutzer sich nicht neu authentifizieren.

Dieses Protokoll kann auch direkt, ohne die SicAri-Plattform als Proxy, zwischen dem Nutzer und dem zentralen Authentifikationsserver stattfinden. Im ersten Schritt wendet sich der Nutzer direkt an den zentralen Authentifikationsserver, authentifiziert sich ihm gegenüber und erhält das Token, falls die Authentifizierung erfolgreich verlaufen ist. Dabei sollte auch eine gegenseitige Authentifikation benutzt werden.

Sicherheit des Protokolls

Wenn der Nutzer sich lokal an einer SicAri-Plattform einloggt, kann die Authentifikation ruhig mittels Name und Passwort verwendet werden, da die Nachrichten, die über das unsichere Internet gehen, nur zwischen der SicAri-Plattform und dem zentralen Authentifikationsserver ausgetauscht werden. Nach der gestellten Vorraussetzung (Punkt 2) werden diese Nachrichten signiert und verschlüsselt, so dass ein Abhören der Nachrichten oder ein Stehlen vom Token auf dem Transportweg verhindert wird.

Sicherheit der Authentifikation des Nutzers vom entfernten Rechner

Auf dem Transportweg zwischen dem Nutzer und der SicAri-Plattform werden die Nachrichten in dem oben beschriebenem Protokoll nicht verschlüsselt, wodurch einem Angreifer das Abhören der Nachrichten und das Stehlen des Tokens leicht gemacht wird.

Obwohl das Passwort nie im Klartext über das Internet gesendet wird, sollten alle Schritte über einen gesicherten Kanal z.B. über HTTPs erfolgen, damit die Nachrichten vertraulich übertragen werden und somit einem Angreifer das Abhören von Nachrichten erschwert wird. Besonders der letzte Schritt, in dem das Token an den Nutzer versendet wird, muss vor Angriffen wie einem einfachen Abhören geschützt werden. Durch die Verwendung von HTTPs wird zusätzlich erreicht, dass die SicAri-Plattform, die als Proxy agiert, sich gegenüber dem Nutzer authentifiziert und damit einen Man in the Middle Angriff verhindert.

Nach einer erfolgreichen Authentifikation, beim Zugriff auf einen Dienst einer SicAri-Plattform, wird das Token als Nachweis einer erfolgreichen Authentifikation an die SicAri-Plattform geschickt und kann von der SicAri-Plattform zur Zugriffskontrolle benutzt werden. Das Verschicken des Tokens in diesem Schritt sollte nicht im Klartext geschehen, weil einem Angreifer wiederum das Abhören und Stehlen des Tokens leicht gemacht wird. Das Token sollte mit dem öffentlichen Schlüssel des Plattformadministrators entweder auf der Transport- oder Anwendungsebene verschlüsselt werden. Die Zugriffanfrage an einen Dienst könnte die URL des Dienstes, das Token und die IP Adresse des Nutzers enthalten und wie folgt aussehen:

$$Zugriffsanfrage = \{Dienst - URL,\ \ Token,\ \ Nutzer - IPaddr.\}^{K^{public}_{Plattform}}$$

Wird diese Anfrage von einem Angreifer abgefangen, kann er diese im Gültigkeitsbereich des Tokens wiedereinspiegeln. Wobei er einige Probleme noch zu bewältigen hätte wie z.B. seine IP Adresse ändern, damit sie mit der IP in der Zugriffsanfrage übereinstimmt. Außerdem muss der Angreifer noch irgendwie die Antwort des SicAri-Dienstes abfangen, die wiederum an die IP, die in der Zugriffsanfrage steht, geschickt wird.

Zusammenfassung

Der wesentliche Nachteil dieses Protokolls besteht darin, dass die Nachrichten, die an den Nutzer geschickt werden, nicht verschlüsselt werden können, weil der Nutzer keinen privaten Schlüssel besitzt. Somit ist es einem Angreifer möglich, die Nachrichten abzuhören und den Inhalt zur eigenen Zwecken zu missbrauchen.

3.2.1.2 Authentifikation mittels der Smartcard

Die Authentifikation mittels einer Smartcard ist viel sicherer als die mittels Name und Passwort, weil die privaten Schlüssel für Signaturerzeugung und Verschlüsselung, die auf der Smartcard gespeichert sind, eingesetzt werden können. Der Ablauf einer Authentifikation mittels der Smartcard ist in Abbildung 3.4 dargestellt.

Die einzelnen Schritte des Authentifikationablaufs eines Nutzers vom entfernten Rechner:

1. Der Nutzer erstellt eine Verbindung mit der SicAri-Plattform und wählt die Smartcardauthentifikation aus. Dabei schickt er eine Authentifikationsanfrage mit seinem Namen über die SicAri-Plattform an den Authentifikationsdienst des zentralen Authentifikationsserver.

2a. Der Authentifikationsdienst schickt eine Anfrage mit dem Nutzernamen an den Identitätsmanager, damit der ihm die zugehörigen Zertifikate liefert.

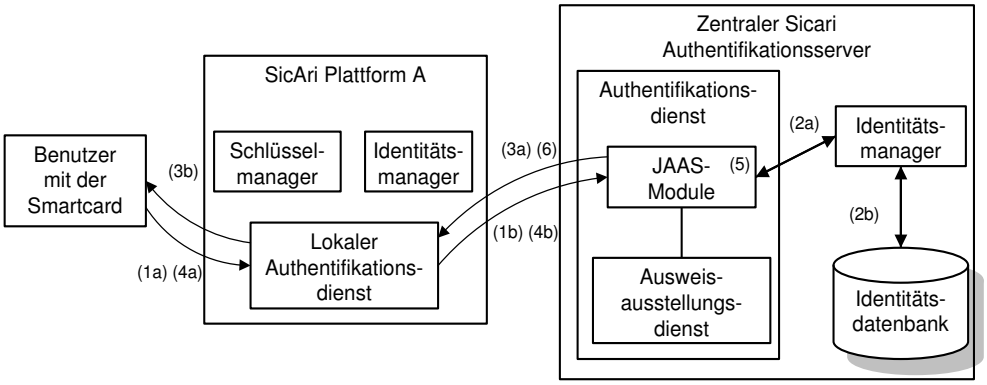

Abbildung 3.4: Authentifikation mittels der Smartcard

2b. Der Identitätsmanager leitet die Anfrage an die Identitätsdatenbank weiter und falls der Nutzername existiert, liefert er dem Authentifikationsdienst die zugehörigen Zertifikate. Ansonsten wird eine entsprechende Fehlermeldung erzeugt, die an den Nutzer weitergeleitet wird.

3. Der Authentifikationsdienst erzeugt die *challenge*, eine Zufallszahl *nonce*, verschlüsselt diese mit dem öffentlichen Schlüssel aus dem Nutzerzertifikat und schickt diese über die SicAri-Plattform an den Nutzer. Die *nonce* kann z.B. durch $nonce = H(aktuelleZeit|Name)$ gebildet werden.

4. Der Nutzer berechnet die *response* $C = H(H(Pwd)|nonce)$ und schickt sie über die SicAri-Plattform an den Authentifikationsdienst des zentralen Authentifikationsserver.

5. Der Authentifikationsdienst berechnet $C' = H(H(Pwd)|nonce)$ und vergleicht $C = C'$. Wenn C und C' gleich sind, dann erzeugt der Authentifikationsdienst einen Token.

6. Wurde das CR-Verfahren erfolgreich abgeschlossen, dann schickt der Authentifikationsserver das Token und das Zertifikat des Nutzers an die SicAri-Plattform, sonst wird eine entsprechende Fehlermeldung erzeugt und an den Nutzer weitergeleitet.

7. Die SicAri-Plattform verschlüsselt das Token mit dem öffentlichem Schlüssel aus dem Zertifikat des Nutzers und schickt dieses dem Nutzer zu, damit er einen Nachweis einer erfolgreichen Authentifikation erhält und den anstatt einer Neuauthentifizierung benutzen kann.

Um eine gegenseitige Authentifikation zu erreichen, kann das Protokoll entsprechend dem im vorherigen Abschnitt beschriebenen Protokoll, angepasst werden.

Sicherheit des Protokolls

Bei der Authentifikation mittels der Smartcard verlässt der private Schlüssel nie die Smartcard und die Aktionen wie Signaturerzeugung oder Verschlüsselung finden in einer sicheren Umgebung auf der Smartcard selbst statt. Somit wird verhindert, dass ein Angreifer an geheime Informationen kommt.

Gelingt es einem Angreifer, die Kommunikation zwischen dem Nutzer und der SicAri-Plattform abzuhören, kann er mit den Nachrichten nichts anfangen, weil sie verschlüsselt sind. Daher hängt die Sicherheit des Protokolls von der Stärke des verwendeten Verschlüsselungsalgorithmus ab.

3.2.1.3 Authentifikation mittels des Softtoken

Ein Softtoken oder ein Schlüsselspeicher ist eine kleine Mini-Datenbank in Form einer Datei, in der Schlüssel und zugehörige X.509-Zeritifikate gespeichert werden. Diese Datei ist mit einem symmetrischen Schlüssel, der aus dem Passwort des Nutzers abgeleitet wird, verschlüsselt und kann nur durch die Eingabe des Passwortes entschlüsselt werden. Ähnlich einer Smartcard erlaubt auch das Softtoken nur dem Nutzer den Zugriff, der sich ihm gegenüber erfolgreich authentifiziert hat. Ein bekannter Schlüsselspeicher ist *Java KeyStore (JKS)*.

Für den Nutzer von einem entfernten Rechner verläuft die Authentifikation mit einem Softtoken gleich der Authentifikation mit einer Smartcard.

Wenn der Nutzer sich lokal in einer SicAri-Plattform einloggen will, muss er das Softtoken auf die SicAri-Plattform kopieren und der SicAri-Plattform mittels Eingabe des Passwortes den Zugang zu dem privaten Schlüssel gewähren. Die SicAri-Plattform agiert dann im Namen des Nutzers und authentifiziert ihn gegenüber dem zentralen Authentifikationsdienst. Bei erfolgreicher Authentifikation erhält die SicAri-Plattform das Token und speichert dieses sicher ab.

3.2.2 Lokales Token als Ergebnis

Bei der lokalen Authentifikation sind dieselben Parteien beteiligt, nur statt einem zentralen Authentifikationsserver existiert ein zentraler SicAri-Server mit einer Identitätsdatenbank, die der Identitätsdatenbank des zentralen Authentifikationsservers gleicht. Dieser zentrale SicAri-Server akzeptiert Anfragen der SicAri-Plattformen nur über eine SSL-Verbindung und verhindert damit den Man in the Middle Angriff sowie andere unbefugte Zugriffe.

Sonst verläuft die Authentifikation wie die mit dem globalen Token, nur dass die SicAri-Plattform nicht als Proxy agiert, sondern selbst die Authentifikation durchführt.

3.3 Delegation

In Abschnitt 1.9 wurde beschrieben, warum Delegation in Webservice-orientierten Architekturen wichtig ist. In diesem Abschnitt wird darauf eingegangen, wie Delegation innerhalb der SicAri-Infrastruktur verwendet wird.

Innerhalb der SicAri-Infrastruktur wird die Zugriffskontrolle in Abhängigkeit vom Nutzer und der zuletzt in der Delegationskette anfragenden SicAri-Plattform gemacht. Ein Nutzer kann selbst kein Token für die SicAri-Plattform erstellen, damit diese in seinem Namen agieren kann. In der SicAri-Architektur kann aber das Token eines Nutzers von einer Plattform zur nächsten verschickt werden, d.h. wenn der Nutzer sich lokal an einer ihm vertauenswürdigen SicAri-Plattform angemeldet hat, erlaubt er dieser Plattform, sein Token zu verwalten und dieses Token zu nutzen, um in seinem Namen zu agieren.

Die SicAri-Plattformen ihrerseits verwalten Zertifikate vertrauenswürdiger SicAri-Plattformen. Das hat zur Folge, dass SicAri-Plattformen nur mit vertrauenswürdigen SicAri-Plattformen vertraulich über eine SSL-Verbindung kommunizieren und müssen diese nicht explizit in die Zugriffskontrolle einbeziehen.

Sollte ein Webservice geheime Informationen verwalten, die auf dem Transportweg zum Nutzer von Dritten nicht eingesehen werden dürfen, dann ist es empfehlenswert, dass dieser Webservice nur mit Nutzern kommuniziert, die im Besitz von privaten Schlüsseln sind. Dann können die geheimen Informationen mit XMLenc auf der Anwendungsebene abgesichert werden.

3.4 Das Sicherheitstoken

Das Sicherheitstoken oder einfach nur Token genannt, bildet die Grundlage der Autorisierung eines Nutzers und trägt eine wesentliche Rolle zur Realisierung des Single Sign-on. In Abschnitt 1.7.1 wurde schon ein ähnlicher Token in Form eines Kerberostickets vorgestellt. Dieser Abschnitt beantwortet folgende Fragen: was steht in einem Token, wer erstellt das Token, wie sieht ein Token aus und wie wird das Token verschickt.

Was steht in einem Token und wieso?

Ein Token enthält Informationen über einen Nutzer, die den Nutzer eindeutig identifizieren. Zusätzlich enthält das Token Zeitangaben, die seine Gültigkeit aus Sicherheitsgründen einschränken. In SicAri-Architektur enthält das Token folgende Einträge:

Pflichteinträge	Beschreibung
Version	Die aktuelle Versionsnummer des Tokens
User ID	Der eindeutige Identifikator des Nutzers (*user distinguish name*)
Instant time	Der Zeitpunkt der Erstellung des Tokens
NotBefore time	Der Zeitpunkt, ab dem das Token gültig ist
NotOnorAfter time	Der Zeitpunkt, ab dem das Token ungültig ist
Optionale Einträge	**Beschreibung**
Authentication Method	Gibt an, wie sich der Nutzer authentifiziert hat
Role	Ist die Rolle, in der der Nutzer aktiv ist

Tabelle 3.1: Einträge eines Sicherheitstokens

Es können beliebig viele optionale Einträge oder Attribute in das Token eingetragen werden.

Wie sieht ein Token aus?

Das Token hat in der SicAri-Architektur drei Repräsentationen. Zwei davon sind XML-Dokumente und das dritte ist in einem ASN.1-Format dargestellt. Alle diese Tokens enthalten eine Signatur des Erstellers (*issuer*), das die Herkunft und den Inhalt des Tokens sichert und vor unerlaubter Modifikation schützt. Nicht-signierte Token dürfen nicht für die Zugriffskontrolle verwendet werden, weil sonst sich jeder ein Token mit beliebigem Inhalt erstellen und damit die Zugriffskontrolle umgehen könnte. Nun zu den einzelnen Repräsentationen:

1. Das `SAMLSecurityToken`: Das `SAMLSecurityToken` ist repräsentiert als eine von OASIS standardisierte und von `opensaml.org` [1] implementierte SAML Attribute Assertion (im weiteren Verlauf des Kapitels einfach nur Assertion). Es ist ein XML-Dokument und kann leicht einem anderen XML-Dokument, wie z.B. einer SOAP-Nachricht, hinzugefügt werden. Eine signierte Assertion ist in Abbildung 3.5 dargestellt.

```
<Assertion xmlns="...:SAML:1.0:assertion" xmlns:saml="...:SAML:1.0:assertion"
  xmlns:samlp="...:SAML:1.0:protocol" AssertionID="123"
  IssueInstant="2006-04-25T08:33:04.250Z" Issuer="Authentication Authority"
  MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2006-04-25T08:33:04.171Z"
    NotOnOrAfter="2007-04-25T08:33:04.171Z"></Conditions>
  <AttributeStatement>
    <Subject><NameIdentifier Format="e-mail" NameQualifier =
        "https://igd.example.org/sicari">user@example.org</NameIdetifier>
    </Subject>
    <Attribute AttributeName="authenMethod" AttributeNamespace="namespace">
      <AttributeValue>password</AttributeValue> </Attribute>
  </AttributeStatement>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://.../xml-exc-c14n#">
      </ds:CanonicalizationMethod>
      <ds:SignatureMethod Algorithm="http://.../xmldsig#dsa-sha1">
      </ds:SignatureMethod>
      <ds:Reference URI="#123">
        <ds:Transforms>
          <ds:Transform Algorithm="http://.../xmldsig#enveloped-signature">
          </ds:Transform> <ds:Transform Algorithm="http://.../xml-exc-c14n#">
            <ec:InclusiveNamespaces xmlns:ec="http://.../xml-exc-c14n#"
              PrefixList="code ds kind rw saml samlp typens#default">
            </ec:InclusiveNamespaces></ds:Transform>
        </ds:Transforms> <ds:DigestMethod
          Algorithm="http://.../xmldsig#sha1"></ds:DigestMethod>
        <ds:DigestValue>3TAo/GpDmjdAUK/PlQ2QsKhtku8=</ds:DigestValue>
       </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>Ap8seoNpKBYxZcfNCtQzPOVrg==</ds:SignatureValue>
    <ds:KeyInfo> <ds:X509Data> <ds:X509Certificate>
        MIIDLTCCAuqgAwIBAgIEdxJRTjALBgcqhkjOOAQDBQAwZTELMAkGA1
    </ds:X509Certificate> </ds:X509Data> </ds:KeyInfo>
  </ds:Signature>
</Assertion>
```

Abbildung 3.5: Die signierte `SAML Attribte Assertion`

Diese Assertion besitzt den Vorteil, dass sie sich leicht in die SOAP-Nachrichten und damit auch in das WS-Framework integrieren lässt. Da diese Assertion, dem OASIS Standard entspricht, wird auch eine Verwendung außerhalb der SicAri-Architektur ermöglicht. Weiterhin ist die Assertion mit Platzhaltern (wie z.B. für die Signatur) ausgestattet und bietet damit Flexibilität und Erweiterbarkeit.

2. XML-Token: Das XML-Token ist ein selbst erstelltes XML-Dokument, das die Pflichteinträge beinhaltet und Platz für beliebig viele optionale Einträge bereitstellt. Die Signatur dieses

[1] www.opensaml.org

Dokumentes wird als XML-Signatur realisiert. Dieses Token bietet den Vorteil, dass es auf seine Verwendung in der SicAri-Architektur optimiert ist und damit kleiner als die Assertion ist. Die XML-Repräsentation dieses Tokens ist in Abbildung 3.6 dargestellt.

```
<XMLToken
  attributesCount="2"
  authenMethod="password"
  bezahlt="ja"
  expireTime="1147526292093"
  instantTime="1147526292093"
  notBeforeTime="1147526292093"
  role="student"
  tokenID="123"
  version="1.0"
  soapenv:actor="..."
  soapenv:mustUnderstand="0"
/>
```

Abbildung 3.6: Das `XMLSecurityToken`

Alle Einträge außer dem `attributesCount` wurden schon erklärt. Der Eintrag `attributesCount` gibt an, wie viele optionale Attribute dem Token hinzugefügt wurden. Da es beliebig viele optionale Attribute sein können, ist dieser Eintrag für das Auslesen der Attribute notwendig.

Die Signatur dieses Token ist nicht in dem Token selbst enthalten, sondern befindet sich separat (*detached*) in der SOAP-Nachricht mit einer Referenz auf das Token.

3. ASN.1-Token: Die *Abstract Syntax Notation One (ASN.1)* ist eine Beschreibungssprache zur Definition von Datenstrukturen sowie Festlegungen zur Umsetzung von Datenstrukturen und Elementen in ein netzeinheitliches Format. Sie ist ein gemeinsamer Standard der *International Telecommunication Union - Telecommunication Standardization Sector (ITU-T)* und der ISO.

Der Standard dient der abstrakten Beschreibung von Datentypen, ohne auf die rechnerinterne Darstellung einzugehen. Die Notation ist in den ITU-T-Standards X.680ff definiert. Die zugehörigen Standards X.690ff definieren verschiedene *encoding rules* wie die ASN.1-Datenwerte auf Bit-Ebene kodiert werden. Die Standards befinden sich auf der Homepage der ITU [2]. Mit Hilfe von ASN.1 und einer gemeinsamen *encoding rule* können Systeme mit unterschiedlichen internen Datendarstellungen Nachrichten austauschen.

Das ASN.1-Token besteht aus einer ASN.1-Struktur, die die Pflichteinträge sowie auch die optionalen Einträge als eine Sequenz in einem `PlainSecurityToken` kapselt.

Außerdem enthält das ASN.1-Token die Signatur und Informationen über den Signaturerzeuger oder über die -erzeugern, falls mehrere Signaturen vorhanden sind. Die Struktur der `SignedData`, die diese Informationen enthält, ist in Abbildung 3.8 dargestellt.

Dieses Token wird ähnlich einem X509-Zertifikat als Binärdaten erst mittels *Distinguished Encoding Rules (DER)* kodiert und dann in der `Base64`-kodierten Darstellung in eine SOAP-Nachricht eingefügt. Dieses Token besitzt den Vorteil, dass es auch in einer nicht auf Webservice-basierten Welt verwendet werden kann. Das Token ist in Abbildung 3.9 in gekürzter `Base64`-Kodierung in einem `SOAPHeader` dargestellt.

[2] http://www.itu.int/ITU-T/studygroups/com10/languages/

```
PlainSecurityToken ::= SEQUENCE
{
  version       ::= ASN1IA5String,
  tokenID       ::= ASN1IA5String,
  authMethod    ::= ASN1IA5String,
  instantTime   ::= ASN1UTCTime,
  notBeforeTime ::= ASN1UTCTime,
  expiryTime    ::= ASN1UTCTime,
  attributes    ::= ASN1SequenceOf
}
```

Abbildung 3.7: Das `PlainSecurityToken`

```
SignedData ::= SEQUENCE
{
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo ContentInfo,
  certificates
    [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
  crls
    [1] IMPLICIT CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos
}
```

Abbildung 3.8: Die Darstellung von `SignedData`

Wer erstellt ein Token?

Das Token wird von einem Token-Ausstellungsdienst erstellt, der sich entweder lokal auf der Plattform oder als globaler Dienst auf einem zentralen Server befindet. Der Ausstellungsdienst arbeitet eng mit dem Authentifikationsmanager zusammen oder ist ein Teil von ihm, weil erstens das Token erst nach einer erfolgreichen Authentifikation erzeugt wird und zweitens liefert der Authentifikationsmanager die Identitätseigenschaften des Nutzers, die in dem Token enthalten sind und dem Nutzer eindeutig zugeordnet werden können.

Wie wird ein Token einer Nachricht hinzugefügt bzw. aus der Nachricht wieder ausgelesen?

Wie in Abschnitt 2.5 beschrieben wurde, stützt sich SicAri auf die SOAP Implementierung und WS-Framework von Apache Axis. Dieses Framework basiert auf dem Konzept von Handlern oder Handler-Ketten, wobei jeder Handler für die Bearbeitung eines bestimmten Teils der Nachricht zuständig ist.

Um auf der Clientseite ein Token einer SOAP-Nachricht hinzuzufügen und auf der Serverseite dieses Token zu extrahieren und zu überprüfen, muss man die Handler-Kette um jeweils einen Handler pro Seite, der die zugehörige Aufgabe erfüllt, erweitern. Damit für die Programmierer und Nutzer die Bearbeitung und Verwendung des Tokens transparent bleibt, werden die neuen Handler in die globale Verarbeitungskette eingefügt. In Abbildung 3.10 wird dies noch ein mal veranschaulicht.

```
<soapenv:Header>
<BinaryToken
  ASN1SecurityToken="MIIEpAIBATELMAkGBSsOAwIaBQAwAwYBA
  KCCAzEwggMtMIIC6qADAgECAgR3ElFOMAsGByqGSM44BAMFADBlMQs
  wCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwE==
  " soapenv:actor="
  " soapenv:mustUnderstand="0"/>
</soapenv:Header>
```

Abbildung 3.9: Die Base64-Darstellung des `ASN1SecurityToken`

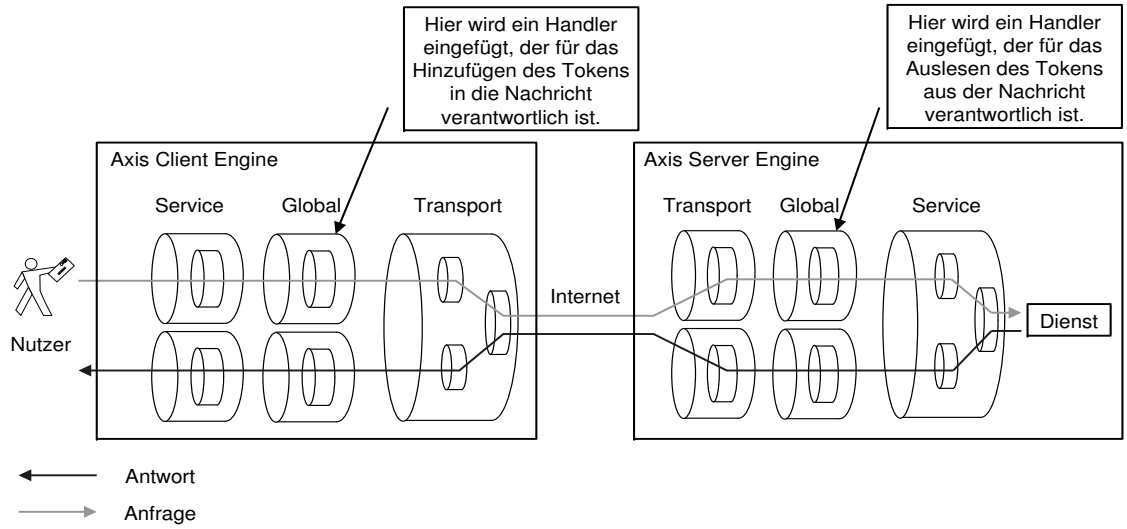

Abbildung 3.10: Die neu eingefügten Handler

Clientseitiger Handler (`GenerateTokenHandler`) und Serverseitiger Handler (`ProcessTokenHandler`)

Die Aufgabe des Handlers auf der Clientseite ist es nicht, nur das Token der Nachricht hinzuzufügen, sondern auch das Token auf der Anwendungsebene mit XMLenc zu verschlüsseln, falls das Token nicht in verschlüsselter Form vorliegt. Außerdem übernimmt der Handler auch das Signieren des Tokens, wenn wie z.B. das XML-Token bis zu diesem Zeitpunkt noch nicht signiert wurde. Die Erzeugung der Signatur erfolgt vor dem Verschlüsseln. Auf diese Weise werden die Schutzziele Authentizität, Integrität und Vertraulichkeit erreicht, auch wenn die Nachricht über mehrere Zwischenknoten transportiert wird.

Auf der Serverseite hat der Handler die Aufgabe, das Token zu entschlüsseln, des Tokens Gültigkeit und die Gültigkeit der Signatur zu überprüfen und anschließend das Token dem lokalen Authentifikationsmanger zur sicheren Verwaltung zu überreichen.

Der `GenerateTokenHandler` erbt dabei von der Klasse `WSDoAllSender` und der `ProcessTokenHandler` erbt von der Klasse `WSDoAllReceiver` (siehe Abbildung 3.11. Die einzige Methode, die dabei vererbt wird, ist die `invoke()`-Methode, über die der `MessageContext` an den Handler übergeben wird. Die `invoke()`-Methode wird für jeden Handler in der konfigurierten Handler-Kette beim Durchlaufen aufgerufen. Das Konfigurieren einer Handler-Kette geschieht in einer `.wsdd`-Datei *Web Service Deployment Descriptor (WSDD)*.

In dieser Datei werden auch die Webservices konfiguriert, worauf hier nicht näher eingegangen wird. SicAri-Plattformen besitzen eine oder zwei solcher Dateien mit dem Namen `client.wsdd` und `server.wsdd` (siehe Anhang B.1 bzw. B.2). SicAri-Plattformen, welche nur die Webservices anbieten, haben nur die server.wsdd. SicAri-Plattformen, die die Webservices anbieten und nutzen, haben beide.

3.5 Integration in SicAri

3.5.1 Integration der Handler

Um das beschriebene Token in die SicAri-Infrastruktur integrieren und nutzen zu können, müssen unter anderem die in Abschnitt 3.2 beschriebenen Protokolle realisiert werden. Da zur Zeit meiner Implementierung in der SicAri-Infrastruktur nicht alle Komponenten fertig gestellt waren, die für die Integration und Nutzung des Tokens benötigt werden, besitzen die Tokens in meinen Tests einen Standard-Inhalt (vgl. Beispiele aus Abschnitt 3.4) und werden nicht im Authentifikationsmanager, sondern im jeweiligen Handler generiert. Pro Tokenart (SAML, XML und ASN1) wurde für die Tests jeweils ein `GenerateTokenHandler` und ein `ProcessTokenHandler` implementiert (siehe Abbildung 3.11. In der aktuellen Version der Implementierung beginnt der Lebenszyklus des Tokens auf der Clientseite im jeweiligen `GenerateTokenHandler` und endet auf der Serverseite im jeweiligen `ProcessTokenHandler`. Das Token wird nicht an den Authentifikationsmanager weitergereicht, was aber in der späteren Entwicklung der SicAri-Plattform vorgesehen ist (siehe die Protokolle in Abschnitt 3.2). Außerdem müssen die Handler noch von Hand in die `.wsdd`-Konfigurationsdateien eingetragen werden und sind dort nach dem Start der Plattform statisch vorhanden. Dieses wird sich aber schon in naher Zukunft ändern und es wird ermöglicht, die Handler einer Plattform während des Betriebs dynamisch hinzuzufügen und auch dynamisch wieder zu entfernen. Die Handler-Klassen sind in Abbildung 3.11 dargestellt.

Wie in Abbildung 3.11 zu sehen, erben alle Handler die Methode `invoke()` von der Klasse `WSDoAllReceiver` bzw. `WSDoAllSender`, die ihrerseits das `org.apache.axis.Handler`-*interface* implementieren.

Im Folgenden werden die einzelnen Handler beschrieben. Der `GenerateASN1TokenHandler` und `ProcessASN1TokenHandler` werden detailliert behandelt. Bei den anderen Handlern werden aufgrund der Ähnlichkeit der Handler nur die Unterschiede dargestellt.

Bei der Beschreibung einzelner Ablaufschritte wird angenommen, dass keine Fehler auftauchen. Beim Auftreten eines Fehlers wird eine zugehörige Fehlermeldung angezeigt.

`GenerateASN1TokenHandler`: Dieser Handler fügt das `ASN1ScurityToken` in eine SOAP-Nachricht ein. Dabei werden folgende Schritte vollzogen:

1. Es wird ein Standard-ASN1SecurityToken erzeugt.
2. Der private Signaturschlüssel wird mittels der Klasse `KeyMaster` aus dem Schlüsselspeicher geladen. Das Token wird mit dem privaten Signaturschlüssel der agierenden SicAri-Plattform signiert. Die Signatur und das zum privaten Signaturschlüssel zugehörige Zertifikat wird im `SignedInfo` des Tokens gespeichert.

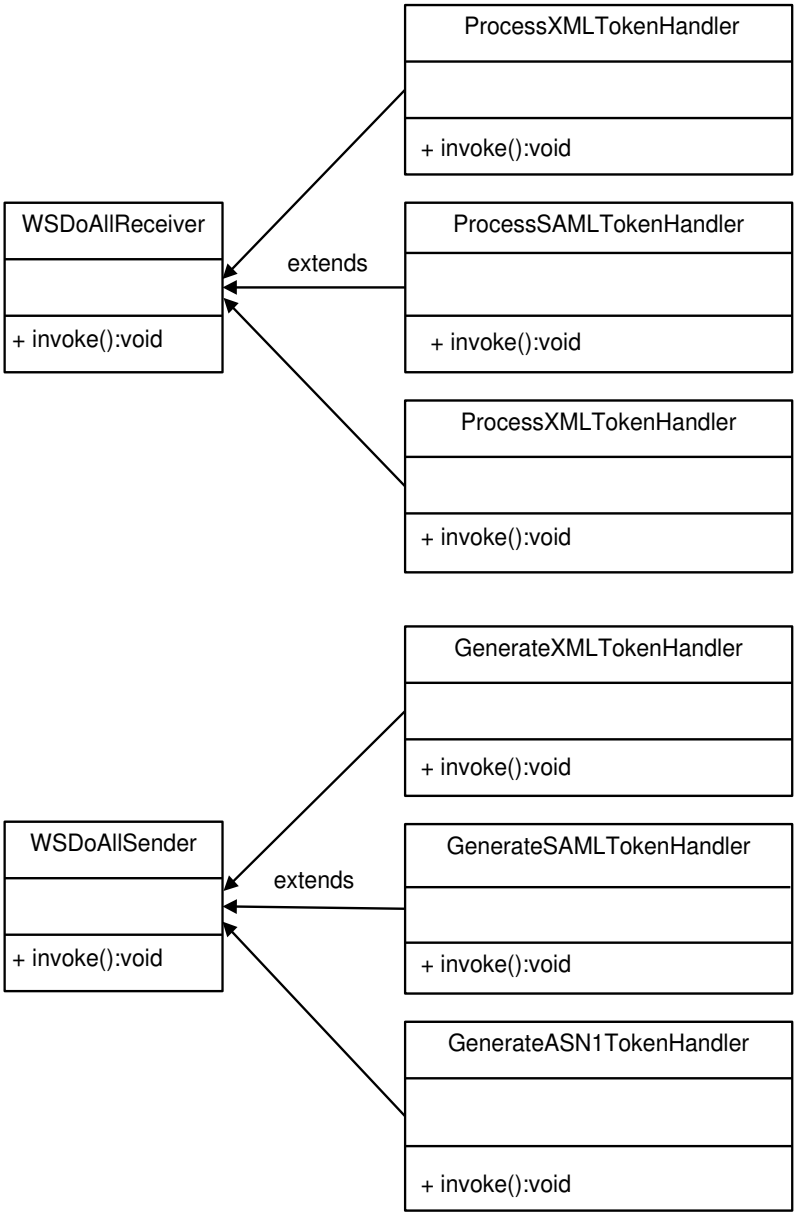

Abbildung 3.11: Die Handlerklassen aus dem `package de.sicari.authentication`

3. Das Token wird erst mittels DER in das binäre Format transformiert und dann mit `Base64` kodiert.

4. Dann wird ein XML-Dokument mit einem Element `BinaryToken` erzeugt. Diesem Element wird ein Attribut namens `ASN1SecurityToken` mit der `Base64`-Kodierung des Tokens als `value` hinzugefügt.

5. Das erzeugte XML-Dokument wird schließlich in den `SOAPHeader` der SOAP-Nachricht eingefügt.

`ProcessASN1TokenHandler`: Auf der Empfängerseite hat der `ProcessASN1TokenHandler` die Aufgabe, das Token zu überprüfen und das Token aus dem `SOAPHeader` der SOAP-Nachricht zu entfernen. Der Ablauf sieht wie folgt aus:

1. Es wird das Element mit dem Namen `BinaryToken` aus dem `SOAPHeader` der SOAP-Nachricht ausgelesen.

2. Die `Base64`-Kodierung des Tokens, die unter `value` des Attributs `ASN1SecurityToken`, wird dekodiert und man erhält die DER-Kodierung des Tokens.

3. Aus der DER-Kodierung des Tokens wird das Token dekodiert.

4. Das im `SignedInfo` des Tokens gespeicherte Zertifikat wird erst auf seine Gültigkeit überprüft.

5. Das Zertifikat wird validiert, d.h. es wird die Vertrauenswürdigkeit des Ausstellers überprüft.

6. Schließlich wird die Signatur verifiziert.

7. Das Token wird aus der Nachricht entfernt.

Wenn der `ASN1SecurityToken` verschlüsselt übertragen werden soll, wird in der Nachricht der Attributname des `BinaryToken`-Elements (siehe Schritt 4 im `GenerateASN1TokenHandler`) in `ASN1EncryptedSecurityToken` geändert und der oben beschriebene Ablauf angepasst. Die ersten zwei Schritte bleiben gleich, weil erst ein `ASN1SecurityToken` erzeugt und signiert wird. Um das `ASN1EncryptedSecurityToken` zu initialisieren müssen folgende Parameter übergeben werden:

- das zu verschlüsselnde ASN1SecurityToken,
- ein symmetrischer Schlüssel der für die Verschlüsselung des Tokens verwendet werden soll,
- Informationen zum symmetrischen Schlüssel wie AlgorithmParameters und der Name des verwendeten Algorithmus, sowie
- das Zertifikat des Empfängers aus dem der asymmetrische, öffentliche Verschlüsselungsschlüssel ausgelesen werden kann.

Diese Parameter werden in Schritt drei vorbereitet. Beim Testen wurde für die symmetrische Verschlüsselung der Algorithmus 3DES verwendet. Der symmetrische Schlüssel wird mit den zugehörigen Informationen wie Name und `AlgorithmParameters` im Handler generiert. Das Zertifikat des Empfängers, also einer anderen vertrauenswürdigen SicAri-Plattform, wird vom `KeyMaster` geliefert.

Da in den ersten beiden Schritten ein signiertes `ASN1SecurityToken` erzeugt wurde, im dritten Schritt die benötigten Parameter generiert wurden, kann im vierten Schritt das `ASN1-EncryptedSecurityToken` initialisiert werden.

Die weiteren Schritte gleichen dem Ablauf des `GenerateASN1TokenHandler`, mit einem Unterschied, dass der Attributname `ASN1SecurityToken` des `BinaryToken`-Elements in `ASN1EncryptedSecurityToken` geändert wird. Anhand des Attributnamen kann der `ProcessASN1TokenHandler` die richtige Methode zur Bearbeitung des Tokens aufrufen.

`GenerateSAMLTokenHandler`:

Der Ablauf dieses Handlers ist dem `GenerateASN1TokenHandler` ähnlich. In den ersten zwei Schritten wird auch ein Standard-Token, also die Assertion mit den Standard-Einträgen, erzeugt und signiert. Die Signatur wird jedoch als eine XMLdsig in der Assertion (*enveloped*) mit dem zugehörigen Zertifikat abgespeichert. Die Schritte 3 und 4 entfallen, weil die Assertion ein XML-Dokument ist und ohne weitere Kodierung in die SOAP-Nachricht eingefügt werden kann.

Falls eine Verschlüsselung der Assertion erwünscht ist, wird nach dem Einfügen der Assertion in die SOAP-Nachricht das `Assertion`-Element mit XMLenc verschlüsselt, unter dem die Assertion sich in der SOAP-Nachtticht befindet.

`ProcessSAMLTokenHandler`:

Die Assertion befindet sich in der Nachricht unter dem Assertion-Element und kann aus diesem Element erzeugt bzw. ausgelesen werden. Ist die Assertion verschlüsselt, muss sie erst entschlüsselt werden. Als nächstes folgen die Schritte zur Überprüfung des Zertifikats und der Signatur, bevor die Assertion aus der SOAP-Nachricht entfernt wird.

`GenerateXMLTokenHandler`:

Das `XMLSecurityToken` wird nach seiner Erzeugung direkt unter dem `XMLSecurity-Token`-Element in die SOAP-Nachricht eingefügt und dann wird es mit XMLdsig signiert und kann mit XMLenc verschlüsselt werden. Die Signatur ist nicht in dem `XMLSecurityToken`-Element eingeschlossen (*detached*) und kann, aber muss nicht mit dem `XMLSecurityToken` zusammen verschlüsselt werden.

`ProcessXMLTokenHandler`:

Auf der Empfängerseite wird das `XMLSecurityToken`, falls es verschlüsselt war, entschlüsselt und dann das Zertifikat und die Signatur des Erzeugers überprüft.

Bei der Verwendung der XMLdsig und XMLenc habe ich die öffentliche Implementierung von *Web Services Security for Java (WSS4J 1.4.0)* [3] benutzt, in der unter anderem das Standard von OASIS WS-Security: SOAP Message Security 1.0 umgesetzt ist. Die verwendeten Dateien sind unter SicAri\lib\wss4j zu finden.

[3] http://ws.apache.org/wss4j/

3.5.2 Integration der Tokens

Während der Implementierung ist das `package de.sicari.authentication` entstanden, das in Abbildung 3.12 dargestellt ist (die Handler wurden schon in Abbildung 3.11 dargestellt).

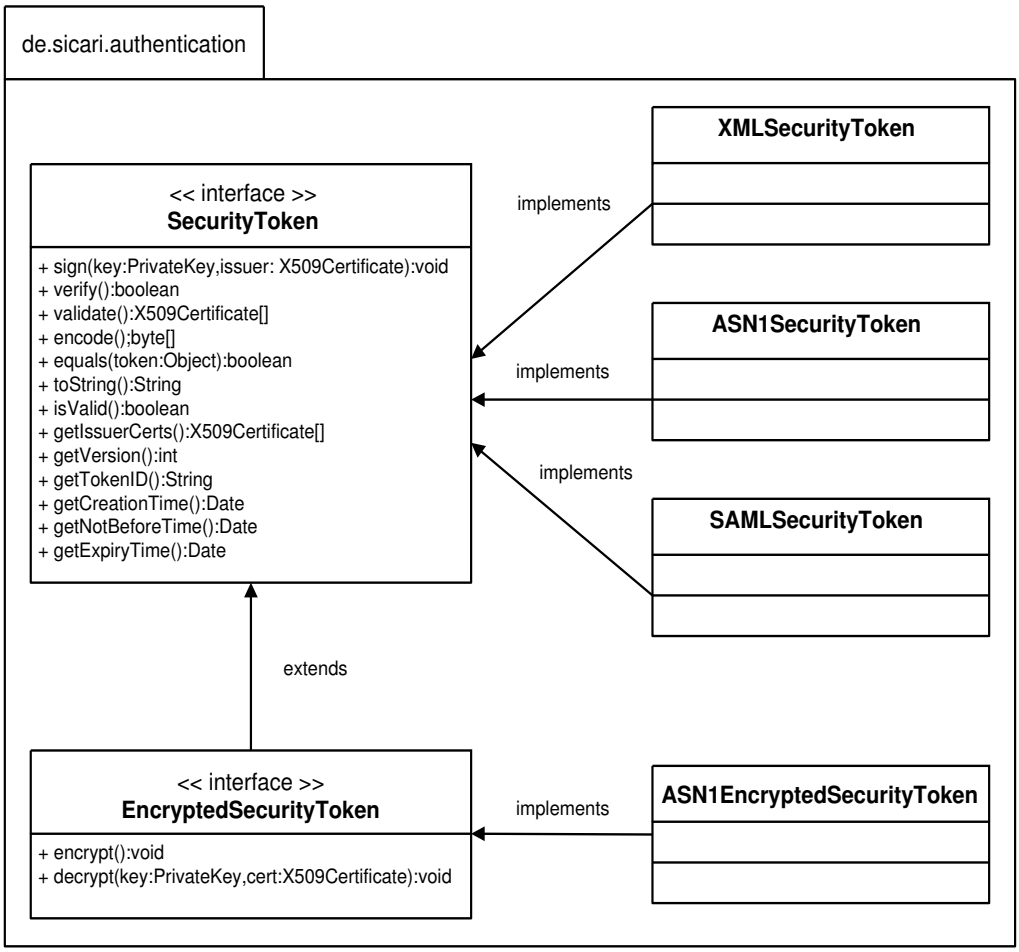

Abbildung 3.12: Das `package de.sicari.authentication`

Das `SecurityToken`-*interface* bildet die Basis der Funktionalitäten eines Tokens. Die Methoden dieses *interface* sind im Folgenden beschrieben:

- `getVersion()`: liefert die aktuelle Versionsnummer als `integer` des vorliegenden Tokens.
- `getTokenID()`: liefert die `TokenID` als ein `String` des vorliegenden Tokens.
- `getCreationTime()`: liefert den Zeitpunkt der Erstellung des vorliegenden Tokens, die `CreationTime` als ein `Date`.
- `getNotBeforeTime()`: liefert den Zeitpunkt, ab wann das vorliegende Token gültig ist. Die `NotBeforeTime` ist als ein `Date` repräsentiert.

- `getExpiryTime()`: liefert den Zeitpunkt, ab wann das vorliegende Token ungültig ist. Die `ExpiryTime` ist als ein `Date` repräsentiert.
- `isValid()`: liefert ein `boolean`, das besagt, ob das vorliegende Token zeitliche Gültigkeit besitzt, `true` falls ja, `false` bei nein.
- `getIssuerCerts()`: liefert `X509Certificate[]`, ein `Array` von X509-Zertifikaten, das erfolgreich validierte Zertifikate von Ausstellern beinhaltet, die das vorliegende Token signiert haben und deren Signatur erfolgreich verifiziert wurde.
- `sign(PrivateKey privateKey, X509Certificate issuer)`: diese Methode wird aufgerufen, um das vorliegende Token mit einem privaten Schlüssel zu signieren. Damit der Empfänger des Tokens die Gültigkeit und die Herkunft der Signatur überprüfen kann, wird das X509-Zertifikat des Ausstellers mit dem Token gespeichert.
- `verify()`: diese Methode ist für das Verifizieren der Signatur zuständig, die mit gültigen und vertrauenswürdigen Zertifikaten erzeugt wurde, d.h. dass die Methoden `validate()` und `isValid()` aufgerufen wurden. Sie liefert ein `boolean` zurück, `true` wenn die Verifikation erfolgreich verlaufen ist, `false` sonst.
- `validate()`: diese Methode ist für die Validierung des Aussteller-Zertifikats zuständig, d.h. es wird überprüft, ob das Zertifikat vertrauenswürdig ist, wobei eine lokale Verwaltung vertrauenswürdiger Zertifikate durch die SicAri-Plattform vorausgesetzt wird. Sie liefert `X509Certificate[]`, ein `Array` von X509-Zertifikaten, das erfolgreich validierte Zertifikate von Ausstellern beinhaltet, die das vorliegende Token signiert haben.
- `encode()`: diese Methode transferiert das Token in eine binäre `Base64`-Darstellung.
- `toString()`: liefert eine für den Menschen lesbare Darstellung des Tokens.
- `equals(Object securityToken)`: vergleicht zwei Token auf die Übereinstimmung ihrer Inhalte. Liefert `true`, wenn zwei Tokens identisch sind, sonst `false`.

Das `SecurityToken`-*interface* wird von den drei Token `XMLSecurityToken`, `SAMLSecurityToken` und `ASN1SecurityToken` implementiert. Das `EncryptedSecurityToken`-*interface* erbt vom `SecurityToken`-*interface* und besitzt zusätzlich zwei weitere Methoden, nämlich:

- `encrypt()`: diese Methode verschlüsselt das `SecurityToken` mit initialisierten Parametern. Sie wird nach der Erzeugung der Signatur und vor der Transformation des Tokens in eine binäre Darstellung gebracht.
- `decrypt(PrivateKey privateKey, X509Certificate recipient)`: diese Methode ist für die Entschlüsselung des Tokens zuständig. Durch das `X509Certificate recipient` weiß der Empfänger, an wen das Token gerichtet ist und kann mit seinem privaten Schlüssel das Token entschlüsseln.

Dieses `EncryptedSecurityToken`-*interface* wird nur vom `ASN1EncryptedSecurityToken` implementiert, weil `XMLSecurityToken` und `SAMLSecurityToken` mit XMLenc verschlüsselt werden.

Das Zusammenspiel der verschiedenen `packages` ist in Abbildung 3.13 dargestellt.

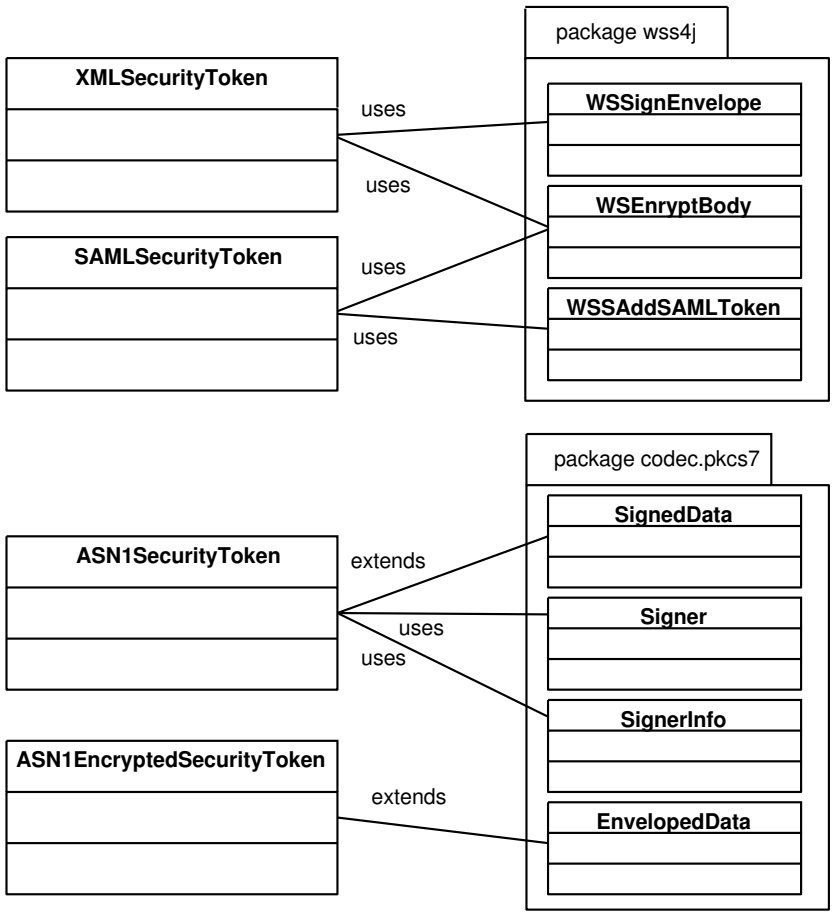

Abbildung 3.13: Das Zusammenspiel der verschiedenen `packages`

3.6 Test

Zielsetzung: Der Test dient der Bestimmung des Nachrichtenlänge- und Zeit-*Overhead*, das durch das Hinzufügen des Sicherheitstoken (und jeweils zugehöriger Informationen wie Referenzen und Signaturen) in eine SOAP-Nachricht entstehen.

Testumgebung:

Die Tests wurden auf einem Computer mit einer Internetverbindung mit folgenden Eigenschaften durchgeführt:

- Computer: Intel Pentium 4 CPU 2.80GHz, 768MB RAM
- Internetverbindung: 2048 kbits/sec downstream, 192 kbits/sec upstream

Testablauf:

Der Testablauf wurde drei Mal, also einmal für jede Art von Token und der zugehörigen Handler durchgeführt. Es wurden signierte, unverschlüsselte Token verwendet. Der Testablauf ist im Folgenden beschrieben:

1. Start einer SicAri-Plattform 1 unter dem `Port 8081`. Diese SicAri-Plattform fungierte als Service-Provider und diente dazu, den Zeitverbrauch des jeweiligen `ProcessTokenHandler` und den Zeitpunkt der Ankunft einer Anfragenachricht zu messen. Der Zeitverbrauch des `ProcessTokenHandler` wurde aus der Differenz zweier Zeitpunkte `startProcessHandler` und `endProcessHandler` gebildet (siehe Abbildung 3.14). Als Zeitpunkt der Ankunft einer Anfragenachricht wurde die gemessene Zeit am Zeitpunkt `startProcessHandler` genommen.
2. Die SicAri-Plattform 1 publizierte einen einfachen Hello-Service, der in einem UDDI-Verzeichnis veröffentlicht wurde. Der Hello-Service ermöglicht dem Nutzer, Anfragen auf Java-Elementartypen wie z.B. `getByte()`, `getString()` zu stellen und liefert eine Instanz des angefragten Elementartyps zurück.
3. Start einer SicAri-Plattform 2 unter dem `Port 8082`. Diese SicAri-Plattform fungierte als Service-Nutzer und diente dazu, den Zeitverbrauch des jeweiligen `GenerateTokenHandler` und den Zeitpunkt des Abschickens einer Anfragenachricht zu messen. Der Zeitverbrauch des `GenerateTokenHandler` wurde auch aus der Differenz zweier Zeitpunkte `startGenerateHandler` und `endGenerateHandler` gebildet (siehe Abbildung 3.14). Als Zeitpunkt der Ankunft einer Anfragenachricht wurde die gemessene Zeit am Zeitpunkt `endGenerateHandler` genommen.
4. Von der SicAri-Plattform 2 wurde nun der Hello-Service aufgerufen und die gewünschten Messdaten gesammelt.

Um die Länge der Nachricht und des eingefügten `SOAPHeader` zu bestimmen, wurden einige Nachrichten in eine Log-Datei geschrieben und später ausgewertet. Die Messzeitpunkte sind in Abbildung 3.14 veranschaulicht.

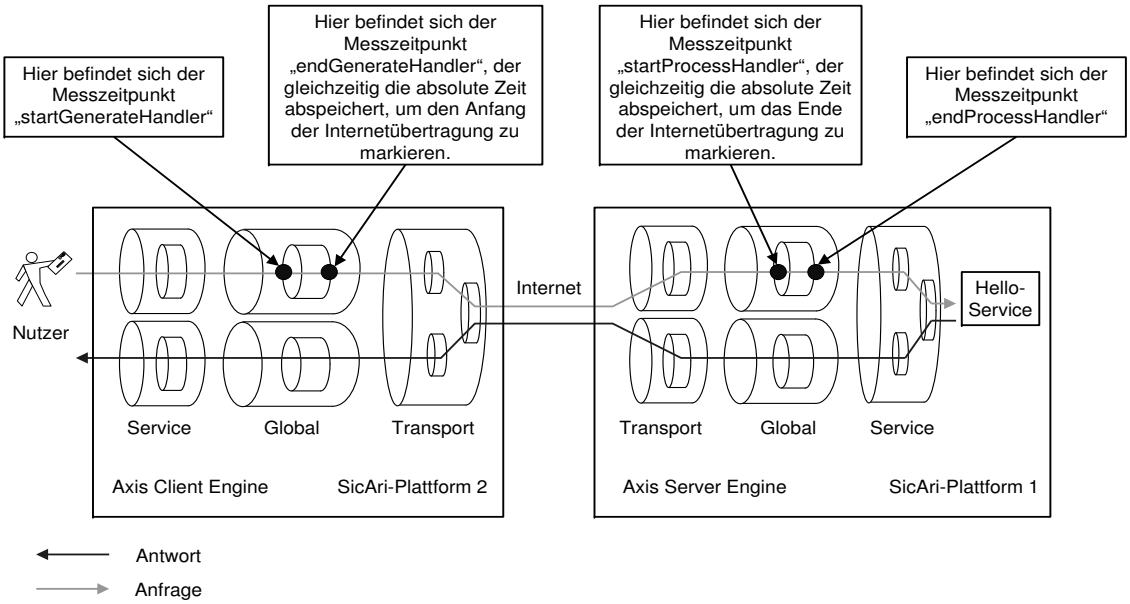

Abbildung 3.14: Die Messpunkte während der Tests

Testergebnisse

Bei der Messung wurden Serviceanfragen nahezu gleicher Länge ausgetauscht (max. Differenz bestand in 12 Zeichen), deswegen sind die Längen der SOAP-Nachricht und einzelner `SOAPHeader` anhand einer `getByte()`-Anfrage über die Anzahl der Zeichen gemessen worden. Die Ergebnisse sind in Tabelle 3.2 dargestellt.

Länge von	XMLSecurityToken	SAMLSecurityToken	ASN1SecurityToken
Nachricht	392	392	392
Nachricht mit Header	1958	3499	2096
Header	1566	3107	1704

Tabelle 3.2: Längen der SOAP-Nachrichten und `SOAPHeader` gemessen über [Anzahl der Zeichen].

Bei der Messung des Zeitverbrauchs der Handler wurden 210 Messwerte und bei der Messung der Dauer einer Nachricht über das Internet $2*84$ Messwerte ausgewertet. Die Ergebnisse der Messung sind in Tabelle 3.3 dargestellt. Die Zeitangaben sind in Millisekunden.

Zeitverbrauch von	XML	SAML	ASN1
GenerateTokenHandler	25	20	71
ProcessTokenHandler	52	60	64
Internetübertragung	40	42	26

Tabelle 3.3: Die Dauer einzelner Abläufe in [msec].

Auswertung des Tests

Der Test sollte einen Überblick über das entstandene *Overhead* der Nachrichtenlänge und des Zeitverbrauchs geben. Die Ergebnisse zeigen, dass beim Verschicken kleiner Nachrichten wie sie im Test verwendet wurden, das *Overhead* der Nachrichtenlänge als auch des Zeitverbrauchs enorm sind. Der Einsatz von Sicherheitstoken sollte nicht für jeden Webservice als Standard verwendet werden. Es ist genau zu überlegen, ob ein Webservice eine Zugriffskontrolle benötigt und ob sie durch Sicherheitstoken realisiert werden soll. Wenn diese beiden Fragen mit ja beantwortet werden, dann muss noch die Wahl der passenden Tokenart getroffen werden.

Kapitel 4

Ausblick

4.1 Ausblick

Für die vollständige Umsetzung der Authentifikation müssen die Authentifikationsprotokolle, die in Abschnitt 3.2 beschrieben sind, realisiert werden. Durch die Authentifikationsprotokolle wird einem Nutzer ermöglicht, sich lokal an einer SicAri-Plattform oder von einem entfernten Rechner anzumelden. Dabei muss der Nutzer sich nur einmal erfolgreich authentifizieren und er erhält ein Sicherheitstoken, wie es in Abschnitt 3.4 beschrieben ist.

Für die Nutzung der *Web Services* mit hohen Sicherheitsanforderungen muss die Authentifikationsmethode auf jeden Fall in die Zugriffskontrolle und damit auch in das Sicherheitstoken einfließen. Den Nutzern, die keine privaten Schlüssel besitzen, sollte der Zugriff auf sensitive *Web Services* verweigert werden, weil die Nachrichten an diese Nutzer im Klartext verschickt werden müssen.

Nach einer erfolgreichen Authentifikation sollte die Kommunikation zwischen dem Nutzer und dem *Web Service* vertraulich stattfinden. Das bisherige zustandslose WS-Security kennt keine Zusammenhänge zwischen den ausgetauschten Nachrichten. Das führt dazu, dass beispielsweise bei mehreren verschlüsselten Nachrichten in Folge an den gleichen Adressaten jedes Mal ein neuer temporärer symmetrischer Schlüssel mit dem öffentlichen Schlüssel des Empfängers verschlüsselt wird. Aus Effizienzgründen sollte einmal am Anfang der Kommunikation ein symmetrischer Schlüssel vereinbart oder von einer Partei vorgeschlagen werden. Mit diesem Schlüssel kann auf der Nachrichtenebene ein Sicherheitskontext erzeugt werden, der die ganze Nachrichtensequenz absichert.

Es sind neue Standards von OASIS in Entwicklung, die dieses Problem angehen und ihren Einsatz im SicAri-Projekt finden sollen. Bis Mitte 2007 sollen drei neue OASIS-Standards die Konzepte von WS-Security erweitern: WS-Trust, WS-SecureConversation und WS-SecurityPolicy [41]. WS-Trust beschreibt einen universellen Token-Dienst, der eine stärkere Entkopplung zwischen *Web Service* Client und Provider erreichen soll. WS-SecureConversation führt einen Sicherheitskontext auf Nachrichtenebene ein und erweitert damit bewährte Konzepte der Transportschicht. WS-SecurityPolicy liefert einen umfangreichen Wortschatz zur Beschreibung von Sicherheitsanforderungen eines *Web Service*.

Anhang A

Akronyme

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
AS	Authentifikationsserver
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation Number 1
CA	Certificate Authority
CGI	Common Gateway Interface
CID	Card Identification
CR	Challenge-Response
CVH	Card Holder Verification
DES	Data Encryption Standard
DH	Diffie-Hellman
DLL	Dynamic Link Libraries
DoS	Denial of Service
DTD	Document Type Definition
HTTP	HyperText Transport Protocol
HTTPs	HTTP secure
ID	Identität
IDL	Interface Definition Language

IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPsec	Internet Protocol Security
ISO	International Organization for Standardization
JRMP	Java Remote Method Protocol
JVM	Java Virtual Machine
KDC	Key Distribution Center
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
MD	Message Digest
NIS	Network Information Service
NFS	Network File System
Nonce	Number used once
NTFS	New Technology File System
OASIS	Organization for the Advancement of Structured Information Standards
OSF	Open Software Foundation
OSI	Open Systems Interconnection
PAM	Pluggable Authentication Modules
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PSP	Policy Specification Point
PUID	Personal User ID
PUK	PIN Unblocking Key
RBAC	Role-based Access Control
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSA	Rivest Shamir Adleman
SAML	Security Assertion Markup Language PostScript

SASL	Simple Authentication and Security Layer
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOA	Service-orientierte Architektur
SOAP	Simple Object Access Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single-Sign-On
TCP	Transmission Control Protocol
TGS	Ticket-Granting Server
TGT	Ticket-Granting-Ticket
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
URI	Uniform Ressource Identification
URL	Uniform Resource Locator
WS	Web Service
WSDD	Web Service Deployment Descriptor
WSDL	Web Service Description Language
W3C	World Wide Web Consortium
XACML	Extensible Access Control Markup Language
XKMS	XML Key Management Specification
XML	Extensible Markup Language
XMLdsig	XML Signature
XMLenc	XML Encryption
X-KISS	Key Information Service Specification
X-KRSS	Key Registration Service Specification
X-TASS	Trust Assertion Service

Anhang B

WSDD-Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultClientConfig"
            xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <globalConfiguration>
   <requestFlow>
     <handler type=
       "java:de.sicari.authentication.asn1.GenerateASN1TokenHandler"/>
   </requestFlow>
 </globalConfiguration>
 <transport name="http"
   pivot="java:org.apache.axis.transport.http.HTTPSender">
 </transport>
 <transport name="local"
   pivot="java:org.apache.axis.transport.local.LocalSender">
 </transport>
 <transport name="java"
   pivot="java:org.apache.axis.transport.java.JavaSender">
 </transport>
</deployment>
```

Abbildung B.1: Die Datei `client.wsdd`

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
 xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <globalConfiguration>
  <parameter name="sendMultiRefs" value="true"/>
  <parameter name="disablePrettyXML" value="true"/>
  <parameter name="adminPassword" value="admin"/>
  <parameter name="attachments.Directory"
    value="U:\diplom\sicari\lib\axis\WEB-INF\attachments"/>
  <parameter name="axis.sendMinimizedElements" value="true"/>
  <parameter name="enableNamespacePrefixOptimization" value="true"/>
  <parameter name="sendXMLDeclaration" value="true"/>
  <parameter name="sendXsiTypes" value="true"/>
  <parameter name="attachments.implementation"
    value="org.apache.axis.attachments.AttachmentsImpl"/>
  <requestFlow>
   <handler type=
     "java:de.sicari.authentication.asn1.ProcessASN1TokenHandler"/>
   <handler type="java:org.apache.axis.handlers.JWSHandler">
    <parameter name="scope" value="session"/>
   </handler>
   <handler type="java:org.apache.axis.handlers.JWSHandler">
    <parameter name="scope" value="request"/>
    <parameter name="extension" value=".jwr"/>
   </handler>
  </requestFlow>
 </globalConfiguration>
 <handler name="LocalResponder"
   type="java:org.apache.axis.transport.local.LocalResponder"/>
 <handler name="URLMapper"
   type="java:org.apache.axis.handlers.http.URLMapper"/>
 <handler name="Authenticate"
   type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"/>
 <service name="AdminService" provider="java:MSG">
  <parameter name="allowedMethods" value="AdminService"/>
  <parameter name="enableRemoteAdmin" value="false"/>
  <parameter name="className" value="org.apache.axis.utils.Admin"/>
  <namespace>http://xml.apache.org/axis/wsdd/</namespace>
 </service>
 <transport name="http">
  <requestFlow>
   <handler type="URLMapper"/>
   <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
  </requestFlow>
  <parameter name="qs.list"
    value="org.apache.axis.transport.http.QSListHandler"/>
  <parameter name="qs.method"
    value="org.apache.axis.transport.http.QSMethodHandler"/>
  <parameter name="qs.wsdl"
    value="org.apache.axis.transport.http.QSWSDLHandler"/>
 </transport>
 <transport name="local">
  <responseFlow>
   <handler type="LocalResponder"/>
  </responseFlow>
 </transport>
</deployment>
```

Abbildung B.2: Die Datei `server.wsdd`

Literaturverzeichnis

[1] Basic Raference Model. Technical report, International Organization for Standardization, October 1994.

[2] D. Eastlake 3rd and J. Reagle. (Extensible Markup Language) XML-Signature Syntax and Processing (RFC 3275). Technical report, Network Working Group, 2002.

[3] David Elliott Bell. Looking Back at the Bell-La Padula Model. Technical report, Dezember 2005.

[4] J. Boyer. Canonical XML Version 1.0 (RFC 3076). Technical report, Network Working Group Request for Comments, März 2001.

[5] Johannes Buchmann. *Einführung in die Kryptographie*. Springer, 3 edition, 2004. ISBN 3-540-40508-9.

[6] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. A Formal Analysis of Some Properties of Kerberos 5 Using MSR. Technical report, University of Pennsylvania Philadelphia, 2002.

[7] Scott Cantor. SAML 2.0 Single Sign-On withConstrained Delegation. Technical report, OASIS, Oktober 2005.

[8] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 1.2. Technical report, OASIS, 2003.

[9] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. Technical report.

[10] T. Dierks and C. Allen. The TLS Protocol Version 1.0(RFC 2246). Technical report, Network Working Group Request for Comments, Januar 1999.

[11] Wolfgang Dostal, Mario Jeckle, Ingo Melzer, and Barbara Zengler. *Service-orientierte Architekturen mit Web Services*. Spektrum Verlag, 1 edition, 2005. ISBN 3-8274-1457-1.

[12] Claudia Eckert. *IT-Sicherheit*. Oldenburg, 3 edition, 2004. ISBN 3-486-20000-3.

[13] David Endler. The Evolution of Cross-Site Scripting Attacks, Mai 2002.

[14] R. Fielding, T. Berners-Lee, J. Gettys, J. Mogul, and et al. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616). Technical report, Network Working Group Request for Comments, Juni 1999.

[15] J. Franks, P. Hallam-Baker, and et al. HTTP Authentication: Basic and Digest Access Authentication (RFC 2617). Technical report, Network Working Group Request for Comments, Juni 1999.

[16] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. Technical report, Transport Layer Security Working Group, November 1996.

[17] Jeremiah Grossman. Cross-Site Tracing (XST), Januar 2003.

[18] Markus Hillenbrand, Joachim Götze, Jochen Müller, and Paul Müller. A Single Sign-On Framework for Web-Services-based Distributed Applications. Technical report, University of Kaiserslautern, 2005.

[19] Jeff Hodges, Robert Aarts, and Paul Madsen. Liberty ID-WSF Authentication Service andSingle Sign-On Service Specification. Technical report, Liberty Alliance Project, November 2005.

[20] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 3280). Technical report, Network Working Group Request for Comments, April 2002.

[21] John Hughes and Scott Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005.

[22] John Hughes and Scott Cantor et al. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005.

[23] John Hughes and Scott Cantor et al. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005.

[24] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML Encryption Syntax and Processing, 2002.

[25] J.Kohl and C.Neuman. The Kerberos Network Authentication Service (V5) (RFC 1510). Technical report, Network Working Group Request for Comments, September 1993.

[26] S. Josefsson. The Base16, Base32, and Base64 Data Encodings (RFC 3548). Technical report, Network Working Group Request for Comments, Juli 2003.

[27] Len LaPadula. Secure Computer Systems Mathematical Foundations. Technical report, März 1976.

[28] Len LaPadula. Secure Computer Systems Exposition And Multics Interpretation. Technical report, November 1996.

[29] Nilo Mitra. SOAP Version 1.2, 2003.

[30] A.G. Morgan. Pluggable Authentication Modules (PAM), 2001.

[31] Bob Morgan. Delegation/intermediaries use case model. Technical report, OASIS, November 2003.

[32] Robert Morris and Ken Thompson. Password Security: A Case History. *CACM*, 22(11):594–597, 1979.

[33] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS, 2005.

[34] J. Myers. Simple Authentication and Security Layer (SASL) (RFC 2222). Technical report, Network Working Group, October 1997.

[35] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security: SOAP Message Security 1.1. Technical report, OASIS, 2006.

[36] B. Clifford Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *International Conference on Distributed Computing Systems*, pages 283–291, 1993.

[37] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment, 2003.

[38] no name. The Why and How of Delegations in Distributed Security Systems.

[39] Rolf Oppliger. Microsoft .NET Passport: A Security Analysis. Technical report, IEEE, Juli 2003.

[40] Oracle Corporation. *Single Sign-On Using Cookies for Web Applications*.

[41] Martin Raepple. Dreiklang. WS-Security: Neue Standards für mehr Sicherheit. *iX 5/2006, S.126 Web Services*, (5), Mai 2006.

[42] Till Rausch. Service Orientierte Architektur Übersicht und Einordnung.

[43] E. Rescorla. Diffie-Hellman Key Agreement Method (RFC 2631). Technical report, Network Working Group Request for Comments, Juni 1999.

[44] V. Samar and R. Schemers. Unified Login with Pluggable Authentication Modules (PAM). Technical report, Open Software Foundation, Oktober 1995.

[45] R. S. Sandhu. A Lattice Interpretation of the Chinese Wall Policy. In *Proc. 15th NIST-NCSC National Computer Security Conference*, pages 329–339, 1992.

[46] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST Model for Role-based Access Control: Towards a Unified Standard. pages 47–64.

[47] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *(IEEE) Computer*, 29(2):38–47, 1996.

[48] R. Srinivasan. Remote Procedure Call Protocol Specification Version 2 (RFC 1831). Technical report, Network Working Group Request for Comments, August 1995.

[49] Andrew Tanenbaum and Marten van Steen. *Verteilete Systeme*. Pearson Studium, 1 edition, 2003. ISBN 3-8271-7057-4.

[50] Steve Vinoski. CORBA integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[51] Dapeng Wang, Thomas Bayer, Thilo Frotscher, and Marc Teufel. *Java Web Services mit Apache Axis*. Software und Support Verlag, 1 edition, 2004. ISBN 3-935042-57-4.

[52] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol (RFC 4252). Technical report, Network Working Group Request for Comments, Januar 2006.