EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Wilhelm-
Schickard-
Institut für Informatik

Diplomarbeit

Securing CICS Web Services

Zur Erlangung des akademischen Grades eines

Dipl. - Informatiker (Uni)

Universität Tübingen

Fachbereich Technische Informatik

Aufgabensteller: Frau Arnold (IBM)

Betreuer: Herr Prof. Dr.-Ing. Wilhelm G. Spruth

Abgabetermin: 30.09.2009

Eingereicht von:

Usama Abu Al Aaish
Brettacherstrasse 10
70437 Stuttgart
Matrikelnummer: 2446675

Erklärung:

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, den 30.09.2009

……..

Abu Al Aaish, Usama

Danksagungen

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben. An erster Stelle möchte ich mich bei meiner Familie bedanken, die mich während meines ganzen Studiums mehr als nur unterstützt hat. Ganz besonders möchte ich mich bei Herrn Prof. Dr.-Ing. Wilhelm G. Spruth bedanken, der zum einen meine Diplomarbeit betreut hat und zum anderen für seine stete Förderung im Bezug auf Weiterbildungsmöglichkeiten im Bereich Mainframe. Weiterhin möchte ich mich bei Frau Isabel Arnold, Frau Martina Schmidt und Herrn Uwe Denneler aus dem IBM Labor Böblingen für Ihre technische Unterstützung bedanken.

Vor allem jedoch möchte ich bei meiner Freundin Charlotte Coy bedanken, die mir während dieser gesamten Zeit stets liebevoll zur Seite stand.

Vielen Dank.

Abstract

Die vorlegende Arbeit enthält einen Überblick über die Sicherheitseinrichtungen, die im Zusammenhang mit den CICS Web Services existieren. Die Diplomarbeit stellt mehrere Verfahren vor, mit denen ein bereits implementierter Web Service auf CICS gesichert werden kann.

Ausgehend von einem CICS, das mit keinerlei Sicherheit konfiguriert wurde, wird erläutert, wie SSL in CICS nutzbar gemacht werden kann. Es wird beschrieben, welche CICS Ressourcen für Web Services benötigt werden, und wie diese Ressourcen abgeändert werden müssen, damit Sicherheitseinrichtungen auf CICS Web Services angewandt werden können. Es wird darauf eingegangen, welche Maßnahmen auf der Clientseite durchzuführen sind, um zunächst beim Aufruf eines CICS Web Services eine SSL Verbindung aufzubauen, und wie der Client und der Server konfiguriert werden müssen, damit Sicherheit auch auf der Nachrichtenebene benutzt werden kann.

In diesen Zusammenhang wird erläutert, wie auf dem Client in der Administrationskonsole des WebSphere Application Server v6.1 die erforderlichen Sicherheitselemente (wie. z.B. Zertifikate) generiert und implementiert werden, und wie diese in die Clientengenerierung mit einfließen. Auf die notwendigen Befehle zum Generieren von Sicherheitselementen mittels RACF seitens des Servers wird eingegangen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1 – Gliederung der Arbeit

In der vorlegenden Arbeit werden die verschiedenen Möglichkeiten behandelt, wie CICS Web Services gesichert werden können. Es werden sowohl transportbasierte, als auch nachrichtenbasierte Sicherheitsmechanismen betrachtet. Darüber hinaus werden für Sicherheitsarchitekten notwendigen Informationen zur Verfügung gestellt. Diese Informationen können zum einen bei der Planung und zum anderen bei der Entscheidung der richtigen Wahl der Sicherheitsmaßnahmen, hilfreich sein. Für Systemprogrammierer werden Anleitungen ausgearbeitet. In diesen werden Konfigurationen anhand verschiedener Szenarien schrittweise auf übersichtlicher Art und Weise erläutert.

Kapitel 2 gibt eine kurze Einführung in CICS Web Services. Anschließend wird ein Überblick über die externen Standards gegeben, die von CICS unterstützt werden. Da die Web Service Description Language (WSDL) und das Simple Object Access Protocol (SOAP) fundamental für das Benutzen von Web Services sind, werden diese ebenfalls angesprochen.

In Kapitel 3 werden die Sicherheitsmechanismen vorgestellt, die für das Sichern von CICS Web Service nötig sind. In einem Szenario wird gezeigt, wie ein Web Service Client ungesichert auf einen Web Service Provider (der unter CICS läuft) zugreift. Hierbei wird verdeutlicht, wie ein unberechtigter Dritter in das Geschehen eingreifen kann. Um dem entgegenzuwirken, werden die entsprechenden Sicherheitsmöglichkeiten ausgearbeitet. Da der Service Provider bestimmte CICS Ressourcen benutzt und diese für das Sichern von Web Service eine wichtige Rolle spielen, wird auf dieses Thema eingegangen.

In Kapitel 4 wird die *Sicherheit auf Nachrichtenebene* behandelt. Zunächst wird ein Überblick über die von CICS unterstützten Sicherheitsspezifikationen (*WS-Security*) gegeben werden. Im weiteren Verlauf des Kapitels wird dann auf die Konfiguration des von CICS bereitgestellten Security Handlers, eingegangen. Eine Gegenüberstellung der bisher vorgestellten Sicherheitsmechanismen schließt dieses Kapitel ab.

In Kapitel 5 wird die *Sicherheit auf Transportebene* behandelt. Zuerst wird die Verwendung von Basisauthentifizierung erläutert, um einen Web Service Client zu authentifizieren. Anschließend wird gezeigt wie SSL/TLS in CICS nutzbar gemacht werden kann. Damit kann zum einen der Web Service Client authentifiziert und zum anderen Geheimhaltung und Integrität bewahrt werden. Es werden dabei nicht nur die Konfigurationen aufgezählt, die an den Ressourcen vorgenommen werden müssen, sondern darüber hinaus auch konkret die

Einstellungen, die von Sicherheitsarchitekten am System selbst vorgenommen werden (wie z.B. Systeminitialisierung konfigurieren, Bibliotheken importieren…).

In Kapitel 6 werden zwei Lösungen für das Ausgangsproblem (Aufgabenstellung) vorgestellt. Nach Veranschaulichung der Ausgangssituation (Aufgabenstellung), wird in zwei (Lösungs-) Modellen die Realisierung der Sicherheitsmechanismen, die in den vorangegangen Abschnitten behandelt wurden, dargestellt.

Im ersten Modell wird auf die *Sicherheit auf der Transportschicht* eingegangen. Im zweiten Modell wird auf die *Sicherheit auf Nachrichtenebene* eingegangen. Für diese Realisierung kamen der *WebSphere Application Server Toolkit* und *WebSphere Application Server v6.1* zum Einsatz. Die Konfigurationen der CICS Region und der WebSphere Umgebungen werden dabei schrittweise in einer Anleitung festgehalten. Diese Anleitungen sollen Sicherheitsarchitekten beim Planen helfen und Systemprogrammieren beim Implementieren unterstützen.

In Kapitel 7 rundet eine Zusammenfassung und Ausblick diese Arbeit ab.

Kapitel 2 - Einführung in CICS Web Services

Was das World-Wide-Web für die Interaktion zwischen Anwendungen und Benutzern ermöglicht hat, ermöglichen Web Services für die „Anwendung zu Anwendung“ Interaktionen. Mit Web Services können Anwendungen schneller, effizienter und günstiger integriert werden, als jemals zuvor. In diesem Kapitel wird daher kurz darauf eingegangen was Web Services sind und welche Eigenschaften sie haben. Desweiteren wird kurz auf die Architektur eingegangen und inwieweit die externen Standards von CICS unterstützt werden. In diesem Zusammenhang wird die Bedeutung der Web Service Description Language (WSDL) und die des Protokolls SOAP (Simple Object Access Protokoll) angesprochen. Die einzelnen Abschnitte sind bewusst kurz gehalten, weil eine ausführliche Einführung in Web Services den Rahmen der Diplomarbeit sprengen würde. Jedoch wird für weitere Informationen auf entsprechende Quellen verwiesen.

2.1. Was ist eigentlich ein Web Service?

Ein Web Service ist ein Software System, dass für eine kompatible Maschine-zu-Maschine Interaktion über ein Netzwerk konzipiert wurde. Es hat ein Interface, dass in einem Maschinen bearbeitbarem Format (speziell die Web Service Definition Language, WSDL) dargestellt wird ist.

Web Services führen eine spezielle Aufgabe oder eine Reihe von Aufgaben aus. Ein Web Service wird in einem Standard beschrieben, offiziell in XML Notation, die „Service Description“ oder Service Beschreibung genannt wird. Diese Beschreibung bietet alle nötigen Details, einschließlich des Nachrichtenformats (das die Operationen detailliert), dem Transportprotokoll und der Lokation, um mit dem Service zu interagieren. [IBM01]

2.2. Eigenschaften von Web Services

Web Services führen Geschäftsfunktionen aus, die von einfacher Anfrage–Antwort Interaktion, bis zu vollständigen Geschäftsprozessen reichen können. Diese Services können sowohl neue Applikationen, als auch bereits existierende Geschäftsfunktionen, sein. Im zweiten Fall werden diese Funktionen so verpackt, dass ein Aufruf über das Netzwerk ermöglicht wird. Services können von anderen Services abhängen, um Ihre Ziele zu erreichen.

Web Services haben die Eigenschaft, dass auf der Clientseite keine zusätzliche Software benötigt wird. Alles was erfordert wird, ist lediglich eine Programmiersprache mit XML und ein Client, der HTTP unterstützt. Auf der Serverseite benötigt man nur einen HTTP Server und einen SOAP Server.

Durch das Verwenden von der Web Services Description Language (WSDL) werden einem alle Informationen zur Verfügung gestellt, die benötigt werden, um einen Web Service als Provider zu implementieren oder um ein Web Service als Requester aufzurufen.
Web Services können über das Web veröffentlicht, aufgefunden und aufgerufen werden. Diese Technologie benutzt bestehende (leichtgewichtige) Internet Standards wie HTTP. Es setzt die vorhandene Infrastruktur wirksam ein.

Einfache Web Services können zu komplexeren vereinigt werden, in dem entweder Technologien angewendet werden können, die den Arbeitsablauf regeln oder in dem Lower-Layer Web Services einer Web Service Implementierung aufgerufen werden. Web Services können verkettet werden, um Higher-Level Geschäftsfunktionen auszuführen. Dies verkürzt die Entwicklungszeit und führt zur bestmöglichen Implementierung.

Client und Server können in verschiedenen Umgebungen implementiert werden. Theoretisch kann jede beliebige Sprache in Betracht gezogen werden, um einen Web Service Client und Server zu implementieren.

XML und HTTP sind die wichtigsten technischen Grundlagen für Web Services. Ein großer Anteil der Web Service Technologie wurde entwickelt, in dem Open-Source Projekte verwendet wurden. Daher sind die Unabhängigkeit von Herstellern und die Kompatibilität sehr realistische Ziele.

Gewöhnliche Anwendungsentwicklung hängt von einer sehr engen Verbindung zu beiden Enden ab. Web Services benötigen eine einfachere Koordination, was eine flexiblere Neukonfiguration der betreffenden Services, die es zu integrieren gilt, ermöglicht.
Web Services benötigen kein graphisches Interface. Sie operieren auf Codeebene. Service Nutzer müssen zwar die Interfaces des Web Service kennen, jedoch nicht die Einzelheiten der Implementierung des Services. [IBM02]

2.3. Übersicht – Web Service Begriffe

Dieser Abschnitt gibt einen Überblick über die Begriffe, die in Zusammenhang mit Web Services stehen. [IBM03]

Extensible Markup Language (XML)

Ein Standard für die Auszeichnungen (Markup) von Dokumenten, welches eine generische Syntax verwendet, um Daten mit einfachen Tags zu bezeichnen, die auch von Menschen lesbar sind. Der Standard ist vom World Wide Web Consortium (W3C) gebilligt worden.

Initial SOAP Sender

Der SOAP Sender, der die SOAP Nachricht am Beginn des SOAP Nachrichtenpfades erzeugt.

Service Provider

Eine Ansammlung von Software, die einen Web Service bereitstellt.

Service Provider Application

Eine Anwendung, die in einem Service Provider angewendet wird. Normalerweise stellt eine Service Provider Application die Business Logik (Geschäftslogik) eines Service Providers bereit.

Service Requester

Eine Ansammlung von Software, die Verantwortlich für die Anfrage an einen Web Service eines Service Providers ist.

Service Requester Application

Eine Anwendung, die im Service Requester benutzt wird. Üblicherweise stellt eine Service Requester Application die Business Logik (Geschäftslogik) eines Service Requester bereit.

SOAP - Simple Object Access Protocol

SOAP ist ein Netzwerkprotokoll mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. 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.

SOAP intermediary[1]

Ein SOAP Knoten ist sowohl ein SOAP Receiver (Empfänger) als auch ein SOAP Sender. Es empfängt also die Nachricht, verarbeitet Teile der Nachricht (genauer die SOAP Header Blöcke), modifiziert eventuell Teile der Nachricht und sendet die Nachricht (an den „Ultimate SOAP Receiver“) weiter.

SOAP message path

Es ist der Pfad den eine SOAP Nachricht auf dem Weg zu ihrem endgültigen Bestimmungsort bestreitet. Dieser beinhaltet den SOAP Sender, null oder mehr SOAP Intermediaries und einen „Ultimate SOAP Receiver“.

SOAP node

Verarbeitungslogik die auf einer SOAP Nachricht angewendet wird.

SOAP receiver

Ein SOAP Knoten, der eine SOAP Nachricht annimmt.

S**OAP sender**

Einer SOAP Knoten, der SOAP Nachrichten versendet.

Ultimate SOAP receiver

Der endgültige SOAP Empfänger. Dieser ist für die Verarbeitung des Inhalts im SOAP Body und jeder der SOAP Header Blöcke verantwortlich.

UDDI - Universal Description, Discovery and Integration

“Universal Description, Discovery and Integration” ist eine Spezifikation für verteilte Webbasierte Informationsregistrierungen von Web Services. Es ist also ein Verzeichnisdienst, der die Informationen über ein Web Service beinhaltet. UDDI kann auch als öffentliche zugängliche Implementierung der Spezifikationen angesehen werden, die es ermöglichen Web Services anderen zur Verfügung zu stellen, so dass diese Web Service auch aufgefunden werden können, wenn danach gesucht wird. Diese Spezifikation wurde von OASIS veröffentlicht.

Web Services Atomic Transaction

Eine Spezifikation, die die Definition eines Typs der “atomaren Transaktionen” ermöglicht. (Dementsprechend wird die komplette Transaktion bis zum Ende durchgeführt, oder überhaupt nicht.)

[1] „**intermediary**“ engl. für „Vermittler“

Web service binding file

Eine Datei, die mit der WEBSERVICE Ressource zusammenhängt. Sie beinhaltet die Informationen, die CICS nutzt um Daten zwischen einkommenden und ausgehenden Nachrichten abzubilden. Diese Datei enthält auch die Datenstrukturen der Anwendung.

Web service description

Ein XML Dokument mit dem ein Service Provider die Spezifikationen, die für den Aufruf des Web Service nötig sind, dem Server Requester mitteilt. Die Web Service Description ist in einer Web Service Description Sprache (WSDL) geschrieben.

Web Service Description Language (WSDL)

WSDL ist eine Metasprache mit deren Hilfe die angebotenen Funktionen, Daten, Datentypen und Austauschprotokolle eines Web Service beschrieben werden können. Es werden im Wesentlichen die Operationen definiert, die von außen zugänglich sind, sowie die Parameter und Rückgabewerte dieser Operationen.

Web Services Security

Eine Reihe an Verbesserungen, die es SOAP Nachrichten ermöglichen Integrität und Geheimhaltung zu gewährleisten.

WS-Atomic Transaction

Web Services Atomic Transaction

WS-I Basic Profile

Besteht aus Klarstellungen, Verfeinerungen, Interpretationen und Verstärkungen von Web Services Basisspezifikationen, um Interoperabilität begünstigen.

WSDL

Web Service Description Language

WSS

Web Services Security

XML

Extensible Markup Language.

XML namespace

Eine Ansammlung an Namen, die durch eine URI referenziert und die in XML Dokumente als Element Typen und Attributnamen benutzt werden.

XML Schema

Ein XML Dokument, das die Datenstruktur beschreibt und den Inhalt andere XML Dokumente einschränkt.

XML schema definition language

Ein XML Syntax für das Schreiben von XML Schemas, dass von W3C (Word Wide Web Consortium) empfohlen wird.

2.4. Die Web Service Architektur

In diesem Abschnitt wird kurz auf die Web Service Architektur eingegangen. Für ausführliche Information siehe [IBM03].

2.4.1. Find-And-Bind

Die Web Service Architektur basiert auf Interaktionen dreier Komponenten, deren Funktionen im Folgenden kurz erläutert werden:

- Dem Service Provider,
- dem Service Requester
- und der Service Registry.

Service Provider

Es ist das Bündel an Software, welches den Web Service anbietet. Es beinhaltet:

- das Anwendungsprogramm
- die Middleware
- die Plattform auf dem sie läuft.

Service Requester

Es ist das Bündel an Software, welches für die Anfrage des Web Services an einen Service Provider verantwortlich ist. Es beinhaltet:

- das Anwendungsprogramm
- die Middleware
- die Plattform auf dem sie läuft.

Service Registry

Ist ein Ort an dem die Service Providers die Beschreibung ihrer angebotenen Web Services zur Verfügung stellen, damit sie ein Service Requester finden kann. Sie ist eine *optionale* Komponente der Web Service Architektur, weil es sehr viele Situationen gibt, in denen Web Service Provider und Web Service Requester ohne diese Registry kommunizieren können. Zum Beispiel könnte die Organisation, die den Web Service anbietet, die Beschreibung direkt an die User verteilen, indem es ihnen diese Beschreibung per Email zukommen lässt, die User könnten diese Beschreibung von einem FTP Server downloaden oder einfach die Beschreibung mittels einer CD-ROM verteilen.

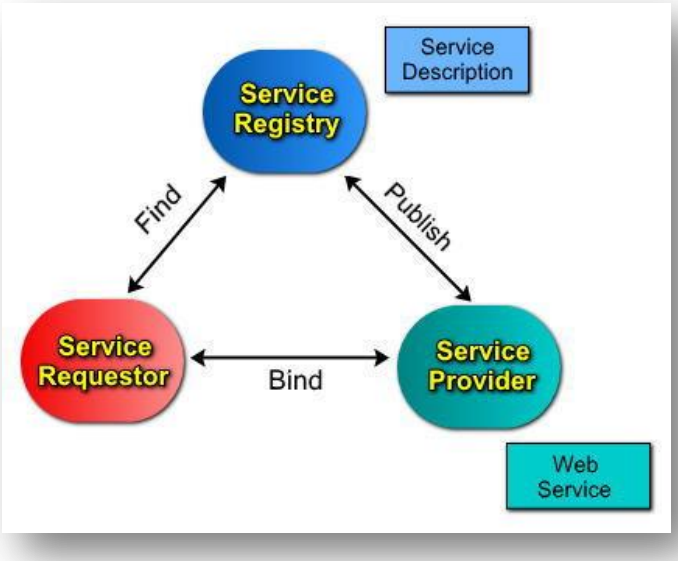

Abbildung 2.1: Web Service Architektur

CICS bietet direkten Support bei der Implementierung eines Web Service Providers oder eines Web Service Requester. Für den Einsatz einer Service Registry wird jedoch zusätzliche

Software benötigt. Weil jedoch die Web Service Architektur plattformunabhängig ist, könnte die Registry auch einfach auf einer anderen Plattform installiert werden.

Die Interaktionen zwischen den drei in Abbildung 1.1 dargestellten Komponenten bestehen aus:

„Bind“: Der Service Requester benutzt die Web Service Description (Service Beschreibung), um an den Service Provider anzubinden und mit der Web Service Implementierung zu kommunizieren.

„Publish“ (veröffentlichen): Wird eine Service Registry verwendet, dann kann der Service Provider seine Service Description in der Service Registry veröffentlichen, damit sie die Service Requester finden zu kann.

„Find“: Wenn eine Service Registry benutzt wird, dann findet ein Service Requester die Service Description in der Registry.

2.5 Die Unterstützung von Externen Standards in CICS V3.2

Die Web Service Technologie umfasst eine Menge von Spezifikationen. Die in der folgenden Abbildung gelb hervorgehobenen Spezifikationen ergeben die *Web Service Base Technology*. Da diese Spezifikationen eine Reihe von externen Standards definieren, wird daher im folgenden Abschnitt auf die externen Standards eingegangen, die von CICS TS 3.2 unterstützt werden und die wesentliche Idee, die dahinter steckt erläutert.

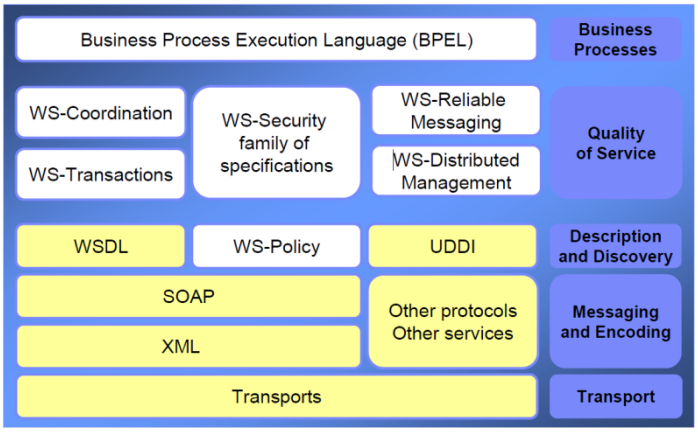

Abbildung 2.2: Web services base technologies

2.5.1 XML – Extensible Markup Language

2.5.1.1 XML Extensible Markup Language (XML) Version 1.0

XML ist eine Teilmenge von SGML, dessen Ziel es ist generische SGML zu ermöglichen, im Web wie HTML anzubieten, zu empfangen und zu verarbeiten. XML wurde für eine einfachere Implementierung entwickelt und ist sowohl mit SGML als auch HTML kompatibel sein. [XML01]

2.5.1.2 XML Encryption Syntax and Processing

Beschreibt den Prozess wie Daten verschlüsselt werden und das Ergebnis in XML dargestellt wird. Diese Daten schließen ein XML Dokument, ein XML Element oder ein XML Content mit ein. Das Ergebnis des Verschlüsseln der Daten ist ein verschlüsseltes XML „Encryption Element“ das entweder den [Geheim]Code enthält oder diesen referenziert. [XML02]

2.5.1.3 XML-Signatur Syntax and Processing

Beschreibt die Regeln und den Syntax für die „XML Digital Signatures“. XML Digital Signatures bieten Datenintegrität, Nachrichtenauthentifizierung und Unterschriftenauthentifizierung für Daten jeden Typs an, ob sie nun in der XML Signatur enthalten oder sonstwo sind. [XML03]

2.5.1.4 XML-binary Optimized Packaging (XOP)

XOP definiert, wie XML Infosets am effizientesten serialisiert werden können, die eine bestimmte Art von Inhalt haben. XOP wird als eine MTOM Spezifizierung implementiert, welche die Optimierung von SOAP Nachrichten definiert. Weil diese beiden Spezifikationen so eng miteinander in Beziehung stehen, werden sie daher als MTOM/XOP verlinkt. [XML04]

2.5.2 Profile

Die Web Service Interoperability Organistaion (WS-I) ist eine Gruppe von Industriepartnern, die das Ziel verfolgen Webstandards zu vereinheitlichen. Wie der Name schon sagt, ist ein Hauptaugenmerk die Plattformunabhängigkeit. Das WS-I ist dabei weder eine WS-Spezifikation noch eine Organisation, die Standards herausgibt, sondern sie erstellt

sogenannte „Profile“ für existierende Spezifikationen. Die von CICS unterstützen Profile sind folgende:

2.5.2.1 WS-I Basic Profile Version 1.0

Bezieht sich auf die Serialisierung von „Envelope“ und deren Darstellung in Nachrichten. [WS-I01]

2.5.2.2. WS-I Basic Profile Version 1.0

wurde jetzt in zwei einzelne Profile aufgeteilt, die veröffentlicht wurden:

- WS-I Simple SOAP Binding Profile Version 1.0 und
- WS-I Basic Profile Version 1.1

2.5.2.3 WS-I Simple SOAP Binding Profile Version 1.0 (SSBP 1.0)

- Ist ein Satz an Spezifikationen zusammen mit Ergänzungen der Spezifikationen, der die Kompatibilität begünstigt.
- Das SSBP 1.0 leitet sich von „WS-I Basic Profile 1.0 Requirement“ ab [WS-I02]

2.5.2.4 WS-I Basic Profile Version 1.1

Ist ein Satz an Spezifizierungen zusammen mit Ergänzungen der Spezifikationen, der die Kompatibilität zwischen Implementierungen verschiedener Web Services begünstigt.

- Es leitet sich von „WS-I Basic Profile 1.0 Requirements“ ab. Dabei wurden die veröffentlichte Errate berücksichtigt und die Anforderungen für die Serialisierung des „Envelope“ und dessen Darstellung in Nachrichten, separiert.
  Diese Anforderungen sind jetzt Teil des Simple SOAP Binding Profile Version 1.0. [WS-I03]

2.6. WSDL – Web Service Description Language

2.6.1 Web Service Description

Eine Web Service Description ist ein Dokument, in dem der Service Provider die Spezifikationen des Web Services darlegt, damit ein Service Requester Kenntnis darüber hat, wie dieser aufzurufen ist. Die Web Service Description wird in XML beschrieben, die so auch unter Web Service Description Language bekannt ist.

Die Service Description beschreibt den Web Service in einer Art und Weise, so dass die Menge an Informationen und der Programmieraufwand, um eine Kommunikation zwischen Service Provider und Service Requester sicherzustellen, geringfügig bleiben.

Zum Beispiel brauchen weder der Service Requester noch der Service Provider zu wissen, welche Plattform der andere Kommunikationspartner benutzt oder welches die eigentliche Programmiersprache ist, in der die Applikation geschrieben ist, die der Service Requester im Service Provider aufrufen will.

Hinweis: Im Folgenden wird auf die WSDL 1.1 Spezifikation eingegangen.

2.6.2 Web Services Description Language Version 1.1

WSDL ist ein XML Format zum Beschreiben von Netzwerkservices, als eine Anzahl von auf Nachrichten operierenden „Endpoints“, die sowohl Dokumenten orientierte, als auch Prozeduren orientierte Informationen beinhalten. Die Operationen und Nachrichten werden abstrakt beschrieben und werden dann an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden, um einen Endpoint zu definieren.

Endpoints, die eindeutig in Verbindung stehen werden zu abstrakten Endpoints zusammengefasst (Services). Die Flexibilität von WSDL zeichnet sich dadurch aus, dass es die Beschreibung von Endpoints und den Nachrichten, unabhängig von dem Nachrichtenformat oder dem Netzwerkprotokoll, das für die Kommunikation eingesetzt wird, ermöglicht. Die WSDL 1.1 Spezifikation definiert nur bindings, die Veranschaulichen wie WSDL in Verbindung mit SOAP 1.1, HTTP GET und POST, und MIMI beschrieben werden.

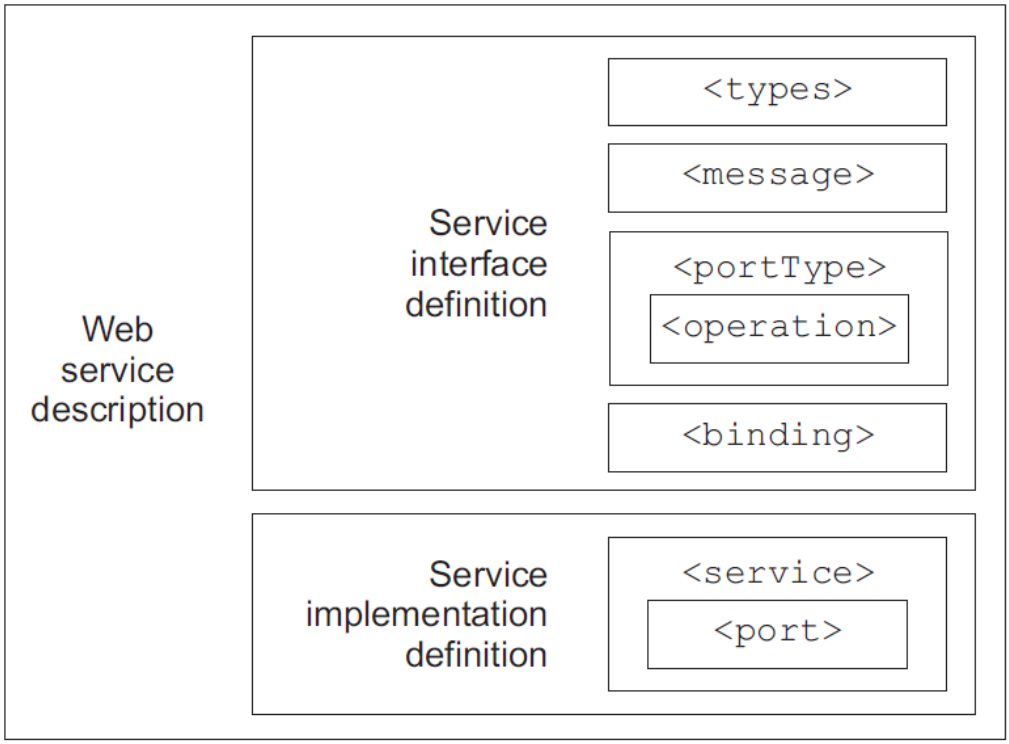

Abbildung 2.3: Web Service Description Struktur

Die WSDL-Struktur ermöglicht es einer Service Description in zwei Teile eingeteilt zu werden (siehe Abbildung 1.3):

- Eine abstrakte Service Interface Description beschreibt die Schnittstellen des Web Service auf eine Art und Weise, die das Schreiben von Anwendungen ermöglicht, die dann den Service implementieren und aufrufen.
- Eine eindeutige Beschreibung der Service Implementierung, die den Ort (oder Endpunkt) des Web Service Providers im Netzwerk oder andere implementierungsspezifischen Einzelheiten so beschreibt, dass es einem Service Requester ermöglicht, sich mit dem Service Provider zu verbinden.

Ein WSDL 1.1 Dokument, das ein Netzwerkservice definiert, besteht aus folgenden Hauptelementen:

<types>

Ein Container für Daten Typen Definitionen, um einige Typsysteme (wie das XML Schema) zu benutzen. Es definiert Datentypen, die in der Nachricht benutzt werden. Das <types>-Element wird nicht benötigt, wenn alle Nachrichten aus einfachen Datentypen bestehen.

<message>

Beschreibt welche XML Datentypen benutzt werden, um die Input und die Output Parameter einer Operation zu beschreiben.

<portType>

Beschreibt einen Satz von Operationen, die von einem oder mehreren Endpunkten unterstützt werden. Innerhalb eines <portType>-Elementes wird jede Operation durch das <operation>-Element beschrieben.

<operation>

Beschreibt welche XML Nachrichten in Einkommenden und Ausgehenden Datenströmen auftreten können. Eine Operation kann mit der Methodensignatur in einer Programmiersprache verglichen werden.

<binding>

Beschreibt das Protokoll, das Datenformat, die Sicherheit und andere Attribute eines <portType>-Elementes.

<port>

Beschreibt die Netzwerkadresse eines Endpunkts und bringt es mit einem <binding>-Element in Verbindung.

<service>

Definiert den Web Service als eine Ansammlung von sich in Beziehung stehenden Endpunkten.
Ein <service>-Element enthält ein oder mehrere <port>-Elemente.

Die Möglichkeit, die Web Service Beschreibung aufzuteilen, ermöglicht auch eine Aufteilung der Verantwortung, bei der Erstellung einer kompletten Beschreibung.

- Der „*Body*“ im Standard beinhaltet eine Service Interface Definition, die folgende Elemente enthält:

  <types>

<message>
<portType>
<binding>

- Ein Service Provider, der eine Implementierung des Services anbieten will, benötigt eine Service Implementierung, die folgende Elemente enthält:

<port>
<service>

Für weitere Informationen über WSDL 1.1 finden sie hier: [W3C01].

2.6.3 Service Veröffentlichung

Eine Service Beschreibung kann durch eine Vielzahl verschiedener Mechanismen veröffentlicht werden. Jeder Mechanismus hat unterschiedliche Ausprägungen und kann in verschiedenen Situationen angewandt werden. Bei Bedarf kann dann eine Service Beschreibung auf mehreren Wegen veröffentlicht werden. Obwohl CICS nicht direkt Support für Service Veröffentlichungen anbietet, kann jede der gleich beschriebenen Möglichkeiten in CICS benutzt werden.

Direkte Veröffentlichung

Dies ist der einfachste Mechanismus, um eine Servicebeschreibung zu veröffentlichen. Der Service Provider sendet die Servicebeschreibung direkt zu dem Service Requester. (Per Email, FTP-Site, CD-ROM,…)

Advertisement and Discovery of Services (ADS)

DISCO

Dieses proprietäre Protokoll bietet einen dynamischen Veröffentlichungsmechanismus. Der Service Requester benutzt einen einfachen HTTP GET Mechanismus, um die Web Service Beschreibung einer Lokation im Netzwerk zu erhalten, die von dem Service Provider spezifiziert und mittels einer URL identifiziert wurde. [IBM04]

Universal Description, Discovery and Integration (UDDI)

Eine Beschreibung für werteilte web-basierte Informationsregistrierungen von Web Services. UDDI kann auch als öffentlich zugängliche Implementierung der Spezifikationen angesehen werden, die es ermöglichen Web Services anderen in der Art zur Verfügung zu stellen, so dass diese Web Services auch aufgefunden werden, wenn danach gesucht wird. [WIKI01]

2.6.4 WSDL 1.1 Binding Extension for SOAP 1.2

„WSDL 1.1 Binding Extension for SOAP 1.2“ ist eine Beschreibung, die die “Binding” - Erweiterungen definiert. Diese werde für das Anzeigen von Web Service Nachrichten, die an das SOAP 1.2 Protokoll gebunden sind, benötigt. Das Ziel dieser Beschreibung ist es Funktionalitäten anzubieten, die mit den „binding“ für SOAP 1.1 vergleichbar sind. [W3C02]

2.6.5 WSDL 2.0 Unterstützung

Die WSDL 2.0 Spezifikation ist eine W3C Candidate Empfehlung. Wenn Veränderungen an dieser Spezifikation vorgenommen werden, was bei einer W3C Empfehlung des Öfteren vorkommt, dann kann dies eine Verschiebung oder eine Veränderung der Implementierung eines WSDL 2.0 Support im CICS Transaction Server v3.2 nach sich ziehen. Der CICS Support für WSDL 2.0 ist Einschränkungen unterzogen. Hier ist eine Liste der vorgeschriebenen Anforderungen:

- Nur die Muster *in-only*, *in-out*, *robust in-only*, und *in-optional-out* können für den Austausch von Nachrichten in WSDL benutzt werden
- Für jeden Service ist nur ein Endpoint erlaubt.
- Es muss mindestens eine Operation geben.
- Endpoints können nur mit einer URI beschrieben werden.
- Es muss eine SOAP Binding geben.
- Als Typ muss das XML Schema benutzt werden.

Informationen im SOAP Header werden von DFHWS2LS ignoriert. Jedoch besteht die Möglichkeit einen eigenen Message Handler einer Pipeline zu erstellen, um damit die

benötigten Informationen im SOAP Header für einkommende und ausgehende Nachrichten zu verarbeiten. [Weitere Informationen zu DFHWS2LS siehe [IBM01] Seite 34 ff]

Das Anhängen von SOAP Header an die von CICS erhaltenen SOAP Nachrichten wird *nicht* von dem Web Service Assistenten (Tool) unterstützt. Jedoch besteht die Möglichkeit, dass dies manuell in der Pipeline konfiguriert wird.

2.6.6 WSDL Message Exchange Patterns

Unterstützte Message Exchange Patterns:

- In-Only
  - CICS als Service Provider
    - CICS wird eine Nachricht empfangen, jedoch keine Antwort senden.
  - CICS als Service Requester
    - Die CICS Anwendung sendet eine Nachricht und erwartet keine Antwort.
- In-Out
  - CICS als Service Provider
    - CICS wird eine Nachricht empfangen und antwortet mit einer normalen Antwort oder einer Fehlermeldung.
    - Die CICS Anwendung sendet eine Nachricht und erwartet eine normale Antwort oder eine Fehlermeldung.
- Robust in-only
  - CICS als Service Provider
    - CICS wird eine Nachricht empfangen und sendet nur eine Antwort im Falle eines Fehlers.
  - CICS als Service Requester
    - Die CICS Anwendung sendet eine Nachricht und erwartet nur im Falle eines Fehlers eine Meldung.
      - Neue „Timeout“ Spezifikation in der PIPELINE Definition
- In-Optional-Out
  - CICS als Service Provider

- CICS empfängt eine Nachricht und kann mit :
  - einer normalen Antwort,
  - einer Fehlermeldung oder
  - überhaupt nicht

  antworten.

- CICS als Service Requester
  - CICS Anwendung sendet eine Nachricht und erwartet:
    - eine normale Antwort,
    - eine Fehlermeldung oder
    - überhaupt nichts.

Quelle: [W3C03]

2.7. SOAP – Simple Object Access Protocol

SOAP ist ein Protokoll für den Austausch von Informationen in einer Verteilten (dezentralisierten) Umgebung. SOAP Nachrichten sind wie XML Dokumente codiert und können durch eine große Anzahl von darunterliegenden Protokollen genutzt werden.

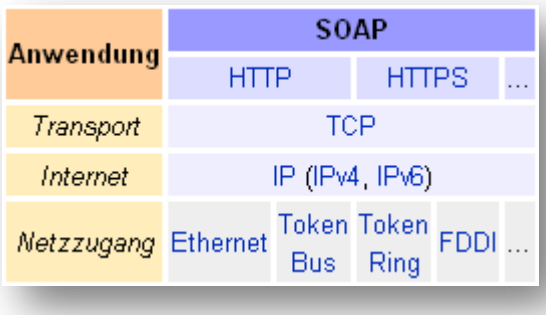

Abbildung 2.4: SOAP im TCP/IP Protokollstapel

Formell ist es ein Akronym für Simple Object Access Protocol. SOAP wurde vom World Wide Web Consortium (W3C) entwickelt.

Die SOAP Spezifikationen beschreiben ein verteiltes Verarbeitungsmodell in dem SOAP Nachrichten zwischen SOAP Knoten durchgereicht werden. Die Nachrichten entstehen beim SOAP Sender und werden zu den SOAP Empfänger geschickt. Zwischen dem Sender und

dem Empfänger könnte die Nachricht von einem oder mehreren SOAP Intermediaries verarbeitet werden. Eine SOAP Nachricht ist eine *One-Way* Übertragung (also eine Übertragung, die nur in *eine* Richtung erfolgt) zwischen SOAP Knoten, von einem SOAP Sender zu einem SOAP Empfänger. Jedoch können Nachrichten kombiniert werden, um eine komplexere Interaktion zu konstruieren, wie beispielsweise *Request and Response* und *Peer-To-Peer* Konversation.

Für weitere Informationen zu SOAP wird auf folgende Dokumente verwiesen:

- **Simple Object Access Protocol (SOAP) 1.1 (W3C note):**
  [W3C04]
- **SOAP Version 1.2 Part 0: Elementarbuch (W3C Empfehlung)**:
  [W3C05]
- **SOAP Version 1.2 Part 1: Messaging Framework (W3C Empfehlung)**:
  [W3C06]
- **SOAP Version 1.2 Part 2: Zusätze (W3C Empfehlung):**
  [W3C07]

2.7.1 Struktur der SOAP Nachricht

Eine SOAP Nachricht ist ein in XML codiertes Dokument, das aus einem <Envelope>-Element besteht. Dieses kann optional ein <Header>-Element haben, muss ein <Body>-Element besitzen. Das <Fault>-Element, das sich im Body befindet, wird für Fehlermeldungen benutzt.

SOAP envelope

Der SOAP <Envelope> ist das Root Element in jeder SOAP Nachricht und besteht aus zwei Kindelemente, einem optionalem <Header> und einem verbindlichem <Body>.

SOAP header

SOAP <Header> ist ein optionales Subelement des SOAP-Envelope und wird für das Verarbeiten von anwendungsrelevanten Informationen, die von auf dem Nachrichtenpfad gelegenen SOAP Knoten durchgeführt werden, verwendet.

SOAP body

SOAP <Body> ist ein obligatorisches Subelement des SOAP Envelope. Es enthält Informationen, die für den endgültigen Empfänger bestimmt sind.

SOAP fault

SOAP <Fault> ist ein Subelement des SOAP Body welches für Fehlermeldungen benutzt wird.

Mit Ausnahme des <Fault>-Elementes, welches sich im <Body>-Element einer SOAP Nachricht befindet, werden XML Elemente, die sich innerhalb der <Header> und <Body>-Elemente befinden, von den Anwendungen definiert. Diese Anwendungen machen davon Gebrauch, obwohl die SOAP Spezifikationen bezüglich Ihrer Strukturen einige Einschränkungen auferlegen.

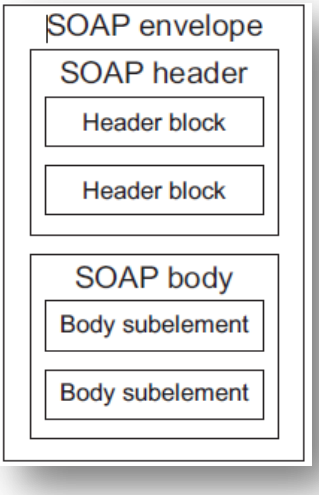

Abbildung 2.5: Die Struktur einer SOAP Nachricht

Einen Verweise für weitere Informationen und sowie eine Einführung zu SOAP befindet sich im Literaturverzeichnis unter [W3C08].

Kapitel 3 - CICS Sicherheit

In diesem Kapitel werden die Sicherheitsaspekte vorgestellt, um CICS Web Services zu sichern. Zuerst wird die konventionelle Sicherheit in CICS erläutert und anschließend wird anhand eines Szenarios die Notwendigkeit bestimmter Sicherheitseinrichtungen demonstriert. Im weiteren Verlauf wird dargestellt, wie diese Sicherheitseinrichtungen realisieren werden können, welche CICS Ressourcen dafür benötigt werden und wie diese Ressourcen zu konfigurieren sind, um Sicherheit gewährleisten zu können.

3.1 Konventionelle Sicherheit in CICS

In einer CICS-Umgebung will man normalerweise die Anwendungsprogramme, sowie die Ressourcen, auf den die Anwendungsprogramme zugreifen, vor unbefugten Zugriff schützen. Dies kann dadurch erreicht werden, dass die Zugriffe in eine CICS Umgebung, sowie in andere CICS Komponenten kontrolliert werden.
Sollen einem CICS Anwender nur die Rechte zugewiesen werden für die er autorisiert sein soll, können individuelle Sicherheitsmechanismen implementiert werden.

Transaction Security (Transaktionssicherheit)
Dadurch wird sichergestellt, dass ein Benutzer, der eine Transaktion durchführen will auch berechtigt wird diese durchzuführen.

Resource Security (Ressourcensicherheit)
Hiermit wird sichergestellt, dass ein Benutzer, der eine Ressource, wie beispielsweise Dateien oder die „Transient Data Warteschlange“ benutzen will auch berechtigt ist dies zu tun. Transient Data werden benutzt, um Daten zwischen CICS und seiner Umgebung auszutauschen.

Command Security (Befehlsausführungssicherheit)
Dies stellt sicher, dass Benutzer, die die CICS eigenen Systembefehle zum Programmieren benutzen wollen auch berechtigt sind diese auszuführen.

Surrogate Security (Stellvertretersicherheit)

Einem Benutzer, der im Auftrag eines anderen Benutzer etwas ausführen soll (surrogate user), wird mittels dieser Implementierung dazu berechtigt.

Die Anfrage auf eine Transaktion und die Anfrage der Transaktion auf eine eventuell benötigte Ressource, verläuft bei aktivierter CICS Security nur in Verbindung mit einer User ID. Bei einer Anfrage wird CICS einen externen Sicherheitsmanager, wie RACF, aufrufen, um zu überprüfen, ob diese User ID auch berechtigt ist, diese Transaktion komplett durchzuführen. Wenn der Benutzer nicht die entsprechenden Rechte besitzt, wird dem Benutzer die Transaktion verweigert.

Der Benutzer kann eine Person sein, welche mittels eines Terminals, einer Workstation oder einem Webbrowser mit CICS interagiert oder, wie im Falle einer Web Service Lösung, eine Applikation, die auf einem Client läuft wurde.

3.1.1 CICS User ID

Verbindet sich ein User mittels eines Terminal mit CICS, so wird beim Start der Sitzung nach einer User ID und nach dem zugehörigen Passwort gefragt. Diese User ID bleibt bis zum Beenden (also dem LOGOFF) der Sitzung bestehen. Die Transaktionen, die der Benutzer vom Terminal ausführt, und die Anfragen, die durch diese Transaktion gemacht werden, stehen mit dieser User ID in Verbindung.

Für Verbindungen, die von Benutzern über das Web gemacht worden sind, gibt es andere Wege, die einen Benutzer gegenüber einer CICS Transaktion identifizieren:

- Ein HTTP Client kann nach einer Information fragen, die den Benutzer eindeutig identifizieren kann. Es wird also nach einer User ID und einem Passwort gefragt. Diese Information wird *Basic Authentication Information* genannt. Die Transaktionen, die vom Web aus angefragt werden, sowie weitere, die von diesen Transaktionen angefragt werden, laufen unter dieser User ID.

- Eine Client Anwendung, die mit CICS mittels Secure Sockets Layer (SSL) (einer kryptografisch verschlüsselten Verbindung) kommuniziert, stellt ein eigenes sogenanntes „Client Certificate“ zur Verfügung. Dieses kann dafür benutzt werden, um den Benutzer der die Verbindung aufgebaut hat (folglich die Anwendung, die die Verbindung hergestellt hat) eindeutig identifiziert. Der „Security Manager“ bildet nun

dieses „Client Certificate“ auf eine User ID ab. Die Transaktion, sowie weitere, die von dieser Transaktion aus angefragt werden, laufen unter dieser User ID.

- Zusätzlich zu diesen Möglichkeiten der Authentifizierung, die sich auf der Transportebene abspielen, können Web Service Clients die Authentifizierungsdaten in Form von Security Token übergeben, die in der SOAP Nachricht eingebettet werden. CICS bietet direkte Unterstützung von Username Token und X.509 Zertifikaten und ist auch mit einem Security Token Service (STS), wie dem Tivoli Federated Identity Manager, kompatibel, um fortgeschrittene Authentifizierungen von Web Services anzubieten.

3.1.2 Spezielle CICS User IDs

Zusätzlich zu den anderen User IDs, die die Benutzer identifizieren, gibt es noch zwei weitere spezifische User IDs, die CICS benutzt. Die Region User ID und die Default User ID:

Region User ID: Die CICS Region User IDs werden benutzt, um die Autorisierung von Anfragen zu überprüfen, die auf das CICS System zugreifen wollen. Beispielsweise Anfragen auf die CICS Data Sets und andere Server.

Default User ID: Wenn sich ein Benutzer nicht anmelden sollte, dann wird CICS diesem Benutzer eine Default User ID (Standard Benutzer) zugewiesen. Daher sollte dem Default User ein Minimum an Rechten zugewiesen werden.

3.2 Sicherheitsüberlegungen – Motivation

In dem folgenden Szenario wurde für die Kommunikation die *Web Service -Technologie* benutzt. Dieses kleine Szenario wird zeigen, welche Sicherheitsmechanismen es gibt und wie diese in Verbindung mit Web Services eingesetzt werden sollten:

Annahmen:

- Man sitzt vor seinem Computer (Client) und will beispielsweise mittels Online Banking eine Überweisung durchführen.
- Die Client-Applikation (Service Requester) verbindet sich hierzu mit der Bank (Service Provider), um die entsprechende Anwendung aufzurufen.

- Es wird angenommen, dass keinerlei Sicherheitsvorkehrungen für die Web Service Kommunikation vorgenommen wurden.

Analyse:

In diesem Szenario hätte ein unberechtigter Dritter (Angreifer) folgende Möglichkeiten in das Geschehen einzugreifen:

- Manipulation: Der Angreifer könnte sich für uns ausgeben und dem Service Provider eine SOAP Nachricht schicken, um so an geheime Informationen zu gelangen oder sogar Geld von unserem Konto abzuheben.
  Diese Art der Manipulation wird auch als „Spoofing“ bezeichnet.

Dieses Problem könnte die Bank dadurch beheben, dass eine *Authentifizierung* verlangt wird.

- Verfälschung*:* Der Angreifer könnte die SOAP Nachricht zwischen uns (*Web Service Requester*) und dem Rechenzentrum der Bank (*Web Service Provider*) abfangen und modifizieren. Er könnte beispielsweise eine andere Kontonummer angeben, auf dem das Geld überwiesen werden soll.
  Die Bank benötigt eine Möglichkeit, die ein Verfälschen verhindert und eine Möglichkeit zur Sicherstellung der Identität. Der Vorgang der Verfälschung wird auch mit „Tampering“ bezeichnet.

Mittels einer *digitalen Signatur* könnten diese Probleme behoben werden.

- Abhören: Ein Angreifer könnte eine SOAP Nachricht abfangen und die Informationen lesen, die diese Nachricht beinhaltet, da diese in Klartext über das Netz geschickt wurde. Er könnte somit an geheime Bankkundendaten, Kontonummern und Kontostände gelangen. Diese Art der Attacke wird als „Eavesdropping“ bezeichnet.

Durch eine *Verschlüsselung* wäre auch dieses Problem behoben.

Dieses kleine Beispiel verdeutlicht nur ein paar Mechanismen, die benötigt werden. Daher folgt nun eine vollständige Auflistung von Mechanismen, wenn eine komplette Sicherheitslösung erstellt werden soll.

3.3 Zielsetzungen für die Sicherheit

Einerseits wäre da die Identität des Benutzers. Diese sollte dem System mitgeteilt werden. Je nachdem was für ein Sicherheitsmodel benutzt wird, in dem diese Identifikation des Benutzers abläuft, wird diese Identität „User ID“, „ID“, „UID“ oder „Principal“ genannt.

Als nächstes muss es eine Möglichkeit geben, dem System zu zeigen, dass man auch der ist, für den man sich ausgibt. Hierzu gibt es verschiedene Möglichkeiten wie z.B. die Angabe einer Information, die nur der Benutzer wissen kann, wie z.B. Benutzername und Password. Eine andere Möglichkeit sich zu authentifizieren ist die Benutzung von Token, die von einem vertrauenswürdigen Dritten zugeteilt wird, wie dem Kerberos ticket oder einem X.509 Zertifikat.
Solche Informationen, die dem Zugriffsberechtigen zugeteilt werden, nennt man *Accessor's Credentials*. Authentifizierungen werden benötigt, um überhaupt Zugriffsrechte zuweisen zu können. Wenn jetzt dieser Benutzer auf eine Ressource zugreifen will, dann benötigt er bestimmte Zugriffsrechte. Er muss also *autorisiert* sein, dies zu tun. Eine typische Implementierung solch einer Autorisierung ist dem Mechanismus, der die Zugriffrechte kontrolliert, einen *Securtiy Context* mitzugeben. Dieser Security Context beinhaltet die authentifizierte Identität.

Ein zusätzlicher Punkt der berücksichtiget werden muss ist die Bewahrung der Integrität. Es muss also sichergestellt werden, dass die Daten die über das Netz kommen, auch die Daten sind, die verschickt worden sind, ohne jegliche Veränderungen, seien diese nun unbeabsichtigt oder mutwillig. Wenn die Daten über das Netz versendet werden, dann sollte es auch die Möglichkeit geben, diese zu verschlüsseln.

Angenommen, es ist eine Verletzung der Sicherheit aufgetreten, dann wäre es doch von Vorteil, wenn nachvollzogen werden kann, wann und eventuell von wem diese induziert wurde. Daher sollten bei sicherheitsrelevanten Vorgängen, wie dem Einloggen oder Ausloggen, diese protokolliert werden.

In vielen Situationen ist es auch angebracht, einem Dritten gegenüber beweisen zu können, dass die Daten, die von jemand verschickt worden sind, von dem Empfänger auch empfangen wurden (und zwar die identischen Daten).

3.4 CICS Ressourcen für das Sichern von Web Service

In folgendem Abschnitt werden die Ressourcen aufgezählt, die von den Systemprogrammierern definiert und konfiguriert werden können, so dass sichere Web Service über HTTP aufgerufen werden können.

3.4.1 CICS als Service Provider

In Abbildung 3.1 wird der Ablauf demonstriert, wenn ein Service Requester eine SOAP Nachricht über HTTP zu einem Service Provider sendet, der sich in einer CICS Transaction Server Version 3 Umgebung befindet.

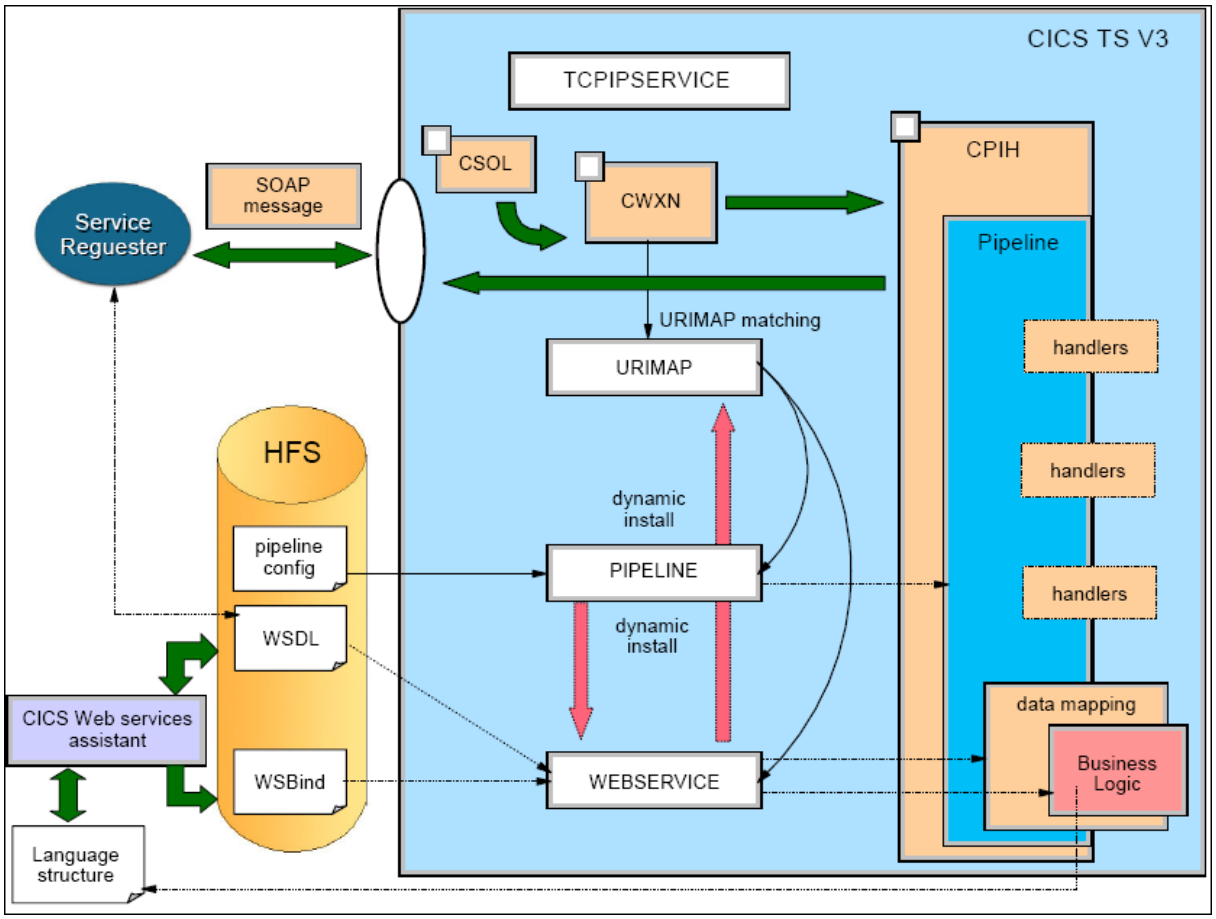

Abbildung 3.1: Web Service Laufzeitumgebung für Service Provider

Der TCPIPSERVICE ist eine sogenannte Ressourcen Definition. In diesem werden z.B. die Ports für eingehende HTTP Anfragen definiert.

Der **CSOL** (**C**ICS-supplied **So**cket **L**istener Transaction) zeigt den Port an, der im TCPIPSERVICE für eingehende HTTP-Anfragen definiert ist. Wenn eine SOAP Nachricht über diesen Port ankommt, ordnet CSOL dieser die Transaktion zu, die im TRANSACTION-Attribut des TCPIPSERVICE definiert ist. Üblicherweise ist dies die Transaktion namens „CWXN“, die von CICS bereitgestellt wird.

CWXN findet die URI in der HTTP Anfrage und durchsucht die URIMAP Ressourcen Definition nach dem Attribut USAGE, das auf PIPELINE gesetzt ist und dessen PATH Attribut auf die URI gesetzt ist, die in der HTTP Anfrage empfangen wurde. Wenn CWXN eine solche URIMAP findet, benutzt es die PIPELINE und WEBSERVICE Attribute der URIMAP Definition, um die Namen der PIPELINE und WEBSERICE Definitionen zu erhalten, welche benutzt werden, um die einkommende Anfrage zu verarbeiten.

CWXN verwendet ebenso das TRANSACTION Attribut, um den Namen der Transaktion zu ermitteln, die für die Verarbeitung der Pipeline zugeordnet wurde. Normalerweise ist es die Transaktion namens CPIH. CPIH startet die Pipeline Verarbeitung. Es benutzt die PIPELINE Definition, um den Name der Konfigurationsdatei zu finden. CPIH benutzt dann diese Konfigurationsdatei, um zu ermitteln, welches Message Handler Programm aufgerufen werden soll.
Ein Message Handler in der Pipeline entfernt den SOAP Envelope der einkommenden Anfrage und übergibt den SOAP Body der Data Mapper Funktion. Auch ein Security Message Handler kann der Pipeline hinzugefügt werden. Ein Security Handler wird benutzt, um den *Security Context* einer Anfrage zu ändern (z.B. die User ID zu ändern unter der die Zielanfrage läuft).

CICS benutzt sogenannte Channels und Container, um Daten zwischen den Message Handler einer Pipeline zu übergeben. Der DFHWS-WEBSERVICE Container wird benutzt, um den Namen der benötigten Webservice Definition zum Data Mapper zu übergeben. Der Data Mapper benutzt die WEBSERVICE Definition, um die Hauptspeicher Kontrollblöcke zu lokalisieren, die von ihm benötigt werden, um die einkommenden Service Anfragen (XML) auf einer COMMAREA oder einem Container abzubilden.
Der Data Mapper ist mit der zu erreichenden Service Provider Anwendung verknüpft. Es stellt dieser Anwendung den Input in einem Format bereit, welches es auch erwartet. Die Anwendung erkennt dadurch nicht, dass sie als Web Service ausgeführt wird. Das Programm führt seinen normalen Ablauf durch und gibt dann eine Ausgabe an die COMMAREA oder einem Container aus, die dann dem Data Mapper übergeben wird. Die ausgegebenen Daten

der CICS Anwendung können nicht einfach zu dem Pipeline Code zurückgeschickt werden. Der Data Mapper must zuerst die Ausgabe der COMMAREA oder des Containerformats standardkonform in eine SOAP Body konvertieren.

3.5 CICS Haupt - Ressourcen und zugehörigen Sicherheitsmechanismen

Um Sicherheitsmechanismen für CICS Web Service zu implementieren werden bestimmte CICS Ressourcen benötigt. Dieser Abschnitt verschafft einen Überblick über die Ressourcen, die dafür benötigt werden, für was diese Ressourcen gebraucht werden, was die Argumente dieser Ressourcen bedeuten und welche Werte für diese Argumente zulässig sind.

3.5.1 TCPIPSERVICE

Eine TCPIPSERVICE Definition wird benötigt, wenn HTTP oder HTTPS als Transport benutzt wird. Es beinhaltet Informationen über den Port, auf welchen die einkommende Anfragen empfangen werden und ob irgendwelche transportbasierte Sicherheitsmechanismen auf CICS angewendet werden. Tabelle 3.1 erläutert die für die Sicherheit relevanten Attribute einer TCPIPSERVICE Ressource.

> **Bemerkung:** Ein *TCPIPSERVICE* Definition wird benutzt, um Transportsicherheit zu konfigurieren.

Attribute	**Beschreibung**
AUTHENTICATE	Ermittelt, ob ein Authentifizierungs,- und Identifizierungsschema auf Transportebene benutzt wird, z.B. HTTP Basic Authentication.
CERTIFICATE	Bestimmt das Label eines X.509 Zertifikat, dass als Server Zertifikat während eines SSL Handshake benutzt wird.
CIPHERS	Bestimmt die Liste der Chiffre, die von dieser CICS Region für SSL Verschlüsselung unterstützt wird.
PORTNUMBER	Bestimmt die Nummer des Ports auf dem CICS auf eingehende HTTP und HTTPS Anfragen lauscht.
SSL	Bestimmt ob SSL für Verschlüsselung und Authentifizierung benutzt werden soll.

Tabelle 3.1: Sicherheitsattribute in der TCPIPSERVICE Ressource

3.5.2 URIMAP

Eine URIMAP Ressource Definition entspricht den URIs der Web Service Anfragen. URIMAP Definitionen für einkommende Web Service Anfragen haben eine USAGE Attribut dessen Wert auf PIPELINE gesetzt ist. Die URIMAP assoziiert eine URI für die Anfrage mit einer PIPELINE und WEBSERVICE Ressource, die die auszuführende Verarbeitung bestimmen.

Im Wesentlichen können URIMAP benutzt werden, um einerseits den Namen der Transaktion zu bestimmen, unter welcher die Pipeline alias Transaktion unter CICS läuft, und zum anderen die User ID, unter welcher die alias Transaktion läuft.

Tabelle 3.2 erläutert die relevanten Attribute einer URIMAP Ressource, wenn CICS als Service Provider agiert.

Attribute	**Beschreibung**
HOST	Beschreibt die Host Komponente der URI, auf welche sich die URIMAP Definition bezieht z.B. www.beispiel.com. Dieses Attribut kann benutzt werden, um Web Service Anfragen auf bestimmte Hostnamen zu beschränken.
PIPELINE	Beschreibt den Namen der PIPELINE Ressource Definition für den Web Service. Die PIPELINE Ressourcen Definition enthält Informationen über die Message Handler, inklusive Security Message Handler, welche auf die Service Anfrage des Clients reagieren.
SCHEME	Beschreibt die Scheme (Schema) Komponente der URI auf welche sich die URIMAP Definition bezieht, die entweder HTTP (ohne SSL) oder HTTPS (mit SSL) sein kann. Es kann benutzt werden, um Web Service Anfragen nur auf HTTPS zu beschränken.
TCPIPSERVICE	Beschreibt den Namen der TCPIPSERVICE Ressource Definition, die einen Inbound Port (d.h. einen Port für einkommende Daten) definiert, auf welchen sich die URIMAP Definition bezieht. Es kann benutzt werden, um einen Zugriff auf Web Services durch einen bestimmten TCPIPSERVICE und dem damit zusammenhängenden transportbasiertem Sicherheitsmechanismus zu beschränken.

TRANSACTION	Beschreibt den Aliasnamen der Pipeline Transaktion, die benutzt wird, um die Pipeline zu starten. Das ist ein sehr wichtiges Attribut, weil es direkt die Transaktionsidentifikatoren kontrolliert, die für Web Service Anfragen benutzt werden und die daher geschützt werden müssen, in dem Transaktionssicherheitsmechanismen verwendet werden.
USAGE	Muss PIPELINE spezifizieren, um anzuzeigen, dass diese URIMAP Definition auf einkommende *Web Service* Anfragen angewandt wird.
User ID	Spezifiziert die User ID, dem die Pipeline Alias Transaktion zugeteilt ist. Wichtig ist hierbei, dass eine User ID, die in der URIMAP Definition spezifiziert wird, von irgendeiner User ID, die man direkt von Client erhalten hat, überschrieben wird.
WEBSERVICE	Bezeichnet den Namen des WebService.

Tabelle 3.2: Sicherheitsattribute für die URIMAP Ressource, wenn CICS als Service Provider agiert

3.5.3 PIPELINE

Eine PIPELINE Ressourcen Definition bietet Informationen über die Message Handler, die auf die Service Anfragen (Request) und Service Antworten (Response) reagieren. Die Informationen über den Message Handler werden indirekt zur Verfügung gestellt; das CONFIGFILE Attribut der PIPELINE Definition spezifiziert den Namen einer HFS Datei, die als Pipeline Konfigurationsdatei bezeichnet wird, welche die XML Beschreibung der Message Handler und deren Konfiguration beinhaltet.
Zu den wichtigsten Attributen der PIPELINE Ressourcen Definition zählen folgende:

WSDIR:

Das WSDIR Attribut spezifiziert den Namen des Web Service Binding Verzeichnisses, auch bekannt unter dem *Pickup Directory*. Dieses Verzeichnis enthält die WSBIND-Dateien, die mit der PIPELINE in Verbindung stehen und die automatisch von dem CICS Scanning Mechanismus installiert werden. Wenn die PIPELINE Definition installiert wird, scannt CICS dieses Verzeichnis und installiert automatisch jedes Web Service Binding File, das es dort auffindet.
Wenn ein Wert für das WSDIR Attribut spezifiziert wird, ist es wichtig darauf zu achten, dass dieses Verzeichnis auch im HFS existiert und CICS zumindest Zugriff für

das Lesen dieses Verzeichnisses hat. Sollte dies nicht der Fall sein, dann wird der Versuch die PIPELINE Ressourcen zu installieren höchstwahrscheinlich fehlschlagen.

SHELF:

Das SHELF Attribut spezifiziert den Namen des HFS Verzeichnisses, in das CICS alle Informationen über installierte Web Services kopiert. CICS Regionen, in welche die PIPELINE Definition installiert werden soll, müssen sowohl Lese,- und Schreibrechte, als auch das Recht besitzen Unterordner erstellen zu können.

CONFIGFILE:

Dieses Attribut spezifiziert den Namen und den Pfad zu der PIPELINE Konfigurationsdatei.

3.5.3.1 PIPELINE Konfigurationsdatei

Wenn CICS eine Web Service Anfrage aufarbeitet, dann benutzt es eine Pipeline bestehend aus einem oder mehreren Message Handlern, die die Anfrage verarbeiten. Die Konfiguration der Pipeline hängt von den Anforderungen der Anwendung ab. Die Konfiguration einer Pipeline, die eine Web Service Anfrage verarbeiten soll, wird in einem XML Dokument spezifiziert, das als Pipeline Konfigurationsdatei bekannt ist. Um diese Datei zu bearbeiten, kann ein gewöhnlicher XML-Editor oder Texteditor benutzt werden.

Es gibt zwei Arten von Pipeline Konfigurationsdateien: Eine, die die Konfiguration einer Service Provider Pipeline beschreibt und eine, die der Service Requester Pipeline beschreibt. Jede wird in ihrem eigenen Schema beschrieben und besitzt ein verschiedenes Root Element. Das Root Element für eine Provider Pipeline ist <provider_pipeline> und das der Requester Pipeline <requester_pipeline>.

Abbildung 3.2 zeigt ein Beispiel für Konfigurationsdatei der (Provider) Pipeline:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Abbildung 3.2: basicsoap11provider.xml

Die mögliche Struktur, die eine Konfigurationsdatei für eine Service Provider Pipeline annehmen kann, wird in der folgenden Abbildung 2.3 dargestellt.

Anmerkung: Die Kindelemente von `<cics_soap_1.1_handler>` und `<cics_soap_1.2_handler>` Elementen wurden in dieser Abbildung 2.3 übersichtshalber nicht angezeigt.)

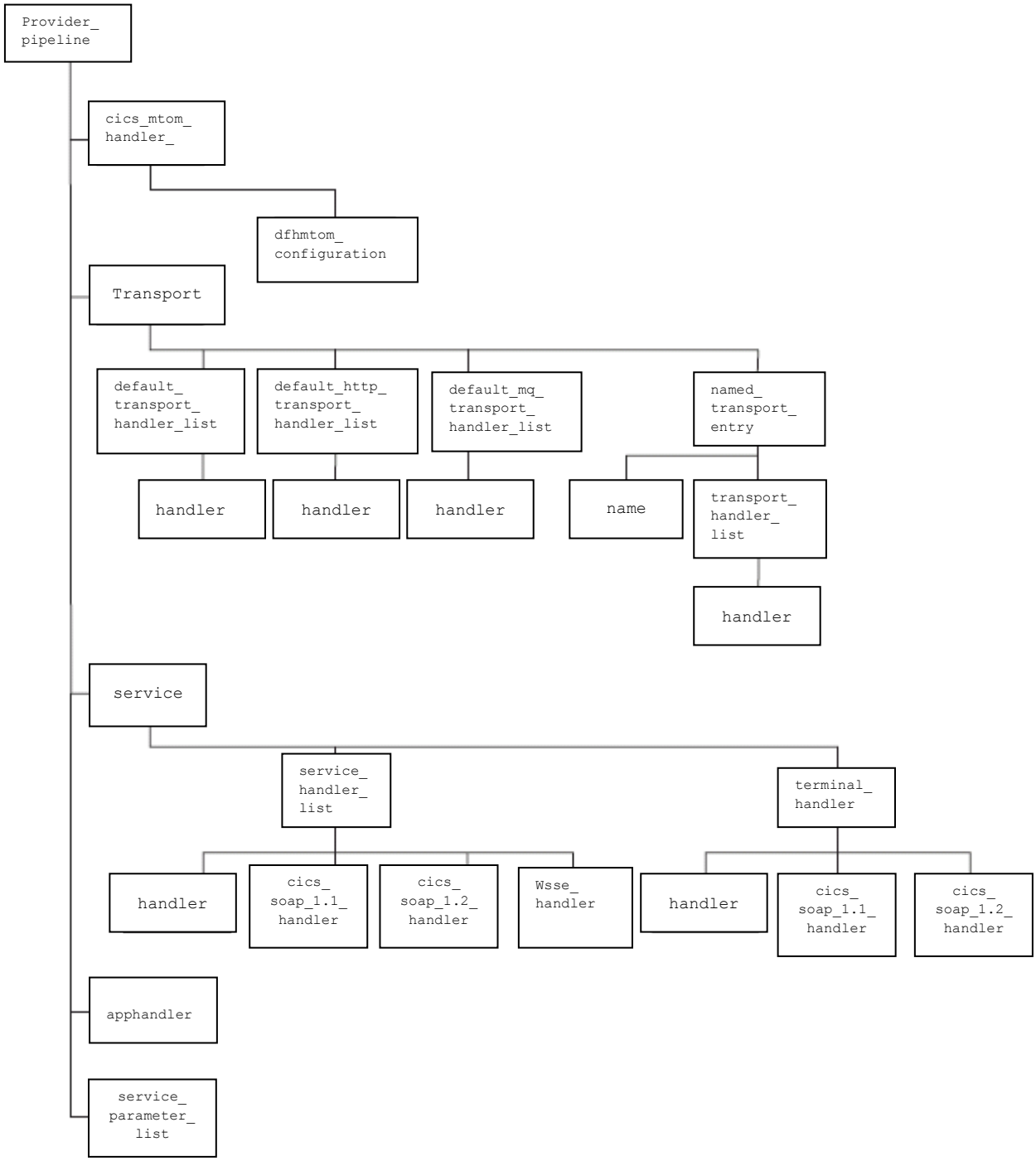

Abbildung 3.3: Struktur einer Pipeline Definition für einen Service Provider

Erläuterung:

`<service>` Element:

Direkt nach dem `<provider_pipeline>` Element folgt das `<service>` Element. Dieses Element beschreibt, welche Message Handler (einschließlich des

Terminal Handlers) benutzt werden. Ein Terminal Handler ist der letzte Handler einer Pipeline.

Weitere optionale Message Handler werden durch das Element `<handler>`, welches sich innerhalb des `<service_handler_list>` Elementes befindet, aufgelistet. Wenn in dem obigen Beispiel ein weiterer Message Handler aufgerufen wird, der einen Request verarbeiten soll, bevor das *Data Conversion Program* (DFHPITP) aufgerufen wird, würde das mit dem optionalen `<handler>` Element gemacht werden, welches sich innerhalb des `<service_handler_list>` - Elementes befindet.

Das DFHPITP ist ein von CICS bereitgestelltes Programm, dessen Aufgaben es ist die WSBIND Datei zu lokalisieren, die in der Pipelineressource angegeben ist, eine XML zu COMMAREA oder XML zu Container Kommunikation durchzuführen (welches in der WSBIND Datei spezifiziert wurde), das Anwendungsprogramm, dass in der WSBIND Datei spezifiziert wurde aufzurufen und anschließend eine COMMAREA (oder Container) zu XML Kommunikation durchzuführen.

Ein weiteres Kindelement des `<service_handler_list>` Elements, ist das `<wsse_handler>` Element, welches die Security Elemente innerhalb einer SOAP Nachricht verarbeitet.

`<transport>` (optionales) Element:

Beschreibt Message Handler die zur Laufzeit ausgewählt werden. Zum Beispiel kann für den HTTP Transport bestimmt werden, dass CICS nur dann den Message Handler aufruft, wenn die Anfrage über den Port erhalten wird, der in einer (bestimmten) TCPIPSERVICE Ressource definiert wurde. Was dabei berücksichtigt werden sollte ist, dass jegliche Handler, die im Service Teil der Pipeline Definition definiert werden, zusätzlich zu den Handler aufgerufen werden, die hier im Transport Part der Pipeline Definition beschrieben sind.

`<apphandler>` (optionales) Element:

Beschreibt den Namen der Anwendung auf die der Terminal Message Handler zeigt. Standardmäßig ist das die Zielanwendung (oder ein Wrapper Programm), welches den Service anbietet.

Message Handler können zur Laufzeit ein anderes Programm bestimmen, in dem sie den „DFHWS-APPHANDLER“ – Container verwenden, so dass der Code, der hier codiert wurde, nicht immer das Programm ist, auf den gelinkt wird.

Wenn beim Entwickeln von Web Services die Programme „DFHLS2WS“ oder „DFHWS2LS“ angewendet wurden, *muss* DFHPITP als das Zielprogramm angegeben werden. DFHPITP enthält den Namen der Zielanwendung (oder des Wrapper Programmes) aus der WSBIND-Datei.

Das `<apphandler>` Element wird verwendet, wenn der letzte Message Handler der Pipeline (der Terminal Handler) einer der CICS bereitgestellten SOAP Message Handler ist.

Wird dieses Element nicht benutzt, muss einer der Message Handler die DFHWS-APPHANDLER Container in Anspruch genommen werden, um den Namen des Programms zu beschreiben.

`<service_parameter_list>` (optionales) Element:

Dieses Element enthält Parameter, die CICS den Message Handler durch den „DFH-SERVICEPLIST“ Container zur Verfügung stellt.

`<terminal_handler>` Element:

Dieses Element definiert den Terminal Message Handler der Service Provider Pipeline. Es beinhaltet entweder

- das `<handler>` Element,
- das `<cics_soap_1.1_handler>` Element oder
- das `<cics_soap_1.2_handler>` Element.

Für weitere Einzelheiten zu diesen Elementen oder den Container wird auf die Seiten 69 ff. des [IBM03] verwiesen.

Um eine Pipeline so zu konfigurieren, dass sie SOAP Message Security unterstützt, muss ein *Security Handler* der Konfigurationsdatei der Pipeline hinzugefügt werden. Dabei kann der von CICS bereitgestellte Security Handler verwendet oder ein benutzerdefinierter Handler erstellt werden.

Ein Security Handler wird meistens dafür verwendet, den *Security Context* einer Anfrage, basierend auf dem Inhalt der SOAP Nachricht, zu ändern. Der von CICS bereitgestellte Security Handler kann für die Verarbeitung von XML Digitalen Signaturen und ebenso für XML Verschlüsslungen, eingesetzt werden.

Wenn ein Security Handler den Inhalt des DFHWS-USERID Container ändert wird die Zielanwendung in einem neuen Task ausgeführt, das unter der neuen Transaktion ID läuft.

3.5.3.2 Message Handler

Ein Message Handler ist ein Programm, welches dazu verwendet wird, um eine Web Service Anfrage beim Eintreffen und die Antwort beim Versenden, zu verarbeiten. Message Handler benutzen Channels und Container, um zum einen untereinander und zum anderen mit dem System zu kommunizieren.

Das Message Handler Interface ermöglicht es folgende Aufgaben in einem Message Handler durchzuführen:

- Den Inhalt einer einkommenden oder ausgehenden Nachricht zu überprüfen, ohne den Inhalt zu ändern.

  Zum Beispiel könnte ein Message Handler, der Verbindungsprotokollierungen durchführt, den für das Protokoll benötigten Teil der Nachricht kopieren. Die Nachricht, die zu dem nächsten Message Handler kommt bleibt dann unverändert.
- Den Inhalt einer XML Anfrage oder Antwort ändern.

  Ein Beispiel hierfür wäre es, wenn eine verschlüsselte Nachricht durch einen Message Handler entschlüsselt wird und diese dann an den nächsten Handler übergeben wird. (Beim Antworten passiert dann genau das Gegenteil. Ein Klartext wird dann von diesem verschlüsselt und weitergeleitet.)
- Ein Message Handler, der nicht ein Terminal Message Handler ist, übergibt eine XML Anfrage oder Antwort an den nächsten Message Handler in der Pipeline.
- In einem Terminal Message Handler wird eine Applikation aufgerufen und die Antwort generiert.
- In der Anfrage-Phase der Pipeline kann ein Übergang zur Antwort Phase der Pipeline, durch die Aufnahme der Anfrage und Generierung der Antwort, erzwungen werden.
- Um Fehler zu verarbeiten.

Alle Programme, die als Message Handler benutzt werden, werden über dasselbe Interface aufgerufen: Sie werden mittels eines *Channels* aufgerufen, der einige Container enthält. Diese Container können folgendermaßen unterteilt werden:

Control Containers

Diese sind für die Operationen der Pipeline unverzichtbar. Message Handler können die Control Container benutzen, um die Reihenfolge zu verändern, in der die Nachrichten verarbeitet werden.

Context Containers

In einigen Situationen benötigen Message Handler Informationen über den Zusammenhang, in dem sie aufgerufen wurden. CICS bietet diese Information in einer Reihe von Context Container an, die an die Programme übergeben werden. Einige der Context Container beinhalten Informationen, die im Message Handler verändert werden können. Zum Beispiel kann in einer Service Provider Pipeline die User ID und Transaction ID der Zielapplikation geändert werden, in dem der Inhalt des entsprechenden Context Container DFHWS-USERID und DFHWS-TRANID geändert wird. Die Auswirkung, einer durch den Message Handler veränderten Transaktions-ID ist die, dass die Zielapplikation und das Data Mapping in zwei verschiedenen Tasks laufen, die dabei die veränderte Transaktions-ID verwenden.

User Containers

Diese beinhalten Informationen, die ein Message Handler benötigt, um es an andere zu übergeben.

3.5.3.3 SOAP Message Handlers

Die SOAP Message Handlers sind von CICS bereitgestellte Message Handler, die in einer Pipeline untergebracht werden können, um SOAP 1.1 und SOAP 1.2 Nachrichten zu verarbeiten. Sie sind sowohl in der Service Requester, wie auch in der Service Provider Pipelines benutzbar.

Bei eintreffenden SOAP Nachrichten parsen die SOAP Message Handler die SOAP Nachrichten und entfernen das SOAP <Body> Element, um es in der Anwendung zu benutzen. Bei ausgehenden SOAP Nachrichten erstellt der SOAP Message Handler eine komplette SOAP Nachricht, in dem es das <Body> Element benutzt, dass die Anwendung anbietet.

Wenn SOAP Headers in den Nachrichten genutzt werden, können die SOAP Handler von Benutzer geschriebenen „Header Processing Programs“ einsetzen, was das Verarbeiten von einkommenden SOAP Header und das Hinzufügen dieser an ausgehenden Nachrichten, ermöglicht.

3.5.4 WEBSERVICE

Eine WEBSERVICE Ressourcen Definition definiert die Aspekte einer Laufzeitumgebung für CICS Anwendungsprogramme, die als Web Service verwendet werden. Drei Objekte definieren die Ausführungsumgebung, die es einem CICS Anwendungsprogramm ermöglicht, als Web Service Provider zu operieren:

- die Web Service Beschreibung (WSDL)
- das Web Service Binding File (WSBIND)
- die Pipeline .

Diese drei Objekte werden für CICS in den folgenden Attributen der WEBSERVICE Ressource Definition definiert:

- WSDLFILE
- WSBIND
- PIPELINE

Die WEBSERVICE Definition hat keine direkten Auswirkungen auf den Security Kontext, die für die Bearbeitung der Anfrage eingesetzt wird.

3.5.5 Setzen der User ID

Es ist möglich, dass für einzelne Web Service Anfragen, die mittels HTTP transportiert werden sollen, von den verschiedenen Methoden, die für das Setzen der User ID denkbar sind, einige gleichzeitig zum Einsatz kommen. In diesem Fall verwendet CICS die *folgende Präzedenzordnung*, um die User ID zu ermitteln, unter welcher dann das (angestrebte) Programm mit der Geschäftslogik läuft:

1. Eine in der Pipeline enthaltene User ID, welche SOAP Nachricht verarbeitet und die von einem Message Handler oder eine SOAP Header Verarbeitungsprogramm spezifiziert wird.
2. Eine User ID, die man von einem Webclient erhält, der Basic Authentication benutzt, oder einer User ID, die mit einem Client Zertifikate verknüpft ist.
3. Eine User ID, die in der URIMAP Definition für die Anfrage spezifiziert ist.
4. Die CICS Standard User ID, wenn keine andere ID bestimmt werden kann.

Sollten verschiedene Mechanismen zur Identifizierung für dieselbe Anfrage benutzt werden, dann hat die SOAP Message Security Priorität.

Kapitel 4 - SOAP Nachrichten Sicherheit

CICS unterstützt eine Reihe von Spezifikationen, die es ermöglichen SOAP Nachrichten zu sichern. In diesem Kapitel wird ein Überblick dieser Security Spezifikationen gegeben und erläutert wie der von CICS bereitgestellte Security Handler konfiguriert wird.

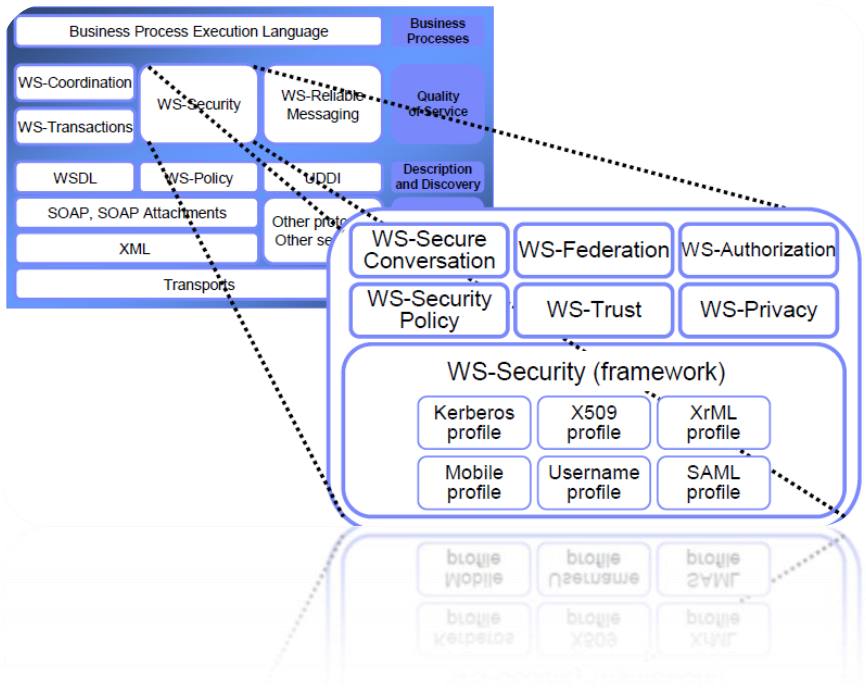

Abbildung 4.1: Web Service Security Komponenten

4.1. WS-Security

WS-Security stellt eine grundlegende Anzahl von Message Erweiterungen zur Verfügung, um sichere Web Services zu erstellen. In dem neue Elemente definiert werden, die im SOAP Header benutzt werden, wird dadurch Sicherheit auf Nachrichten-Ebene gewährleistet. Es spezifiziert die Benutzung von Security Token, Digitalen Signaturen und XML Verschlüsslungen, um SOAP Nachrichten zu schützen und zu authentifizieren.

Darüberhinaus spezifiziert es das Anwenden von Digitalen Signaturen, um die Unversehrtheit der XML Elemente einer SOAP Nachricht zu garantieren und spezifiziert die Verschlüsselung, um die Vertraulichkeit der XML Elemente einer SOAP Nachricht aufrechtzuerhalten. Diese Spezifikation erlaubt es den Body einer Nachricht oder irgendeinem XML Elements, innerhalb des Bodys oder des Headers, zu schützen.

Es ist dabei möglich, dass innerhalb einer SOAP Nachrichten verschiedene Stufen des Schutzes für verschiedene Elemente gesetzt werden.

Der Vorteil der Anwendung von WS-Security über SSL ist, dass „End-To-End“ Sicherheit auf Nachrichtenebene zur Verfügung gestellt werden kann. Das bedeutet, dass die Nachrichtensicherheit auch dann noch gewährleistet werden kann, wenn die Nachrichten verschiedene Dienste, die man „Intermediaries“ nennt, durchlaufen.
SSL Sicherheit wird als „Punkt-zu-Punkt“ Verbindung gesehen und die Daten können daher vor dem Erreichen des eigentlichen Empfängers entschlüsselt werden.

Wie man in folgender Abbildung 3.2 sieht, wird, wenn sich der Service Requester gegenüber dem Intermediate Gateway identifiziert, und das Intermediate Gateway sich gegenüber dem Service Provider identifiziert, der angestrebte Service unter der Identität des Intermediate Gateway laufen, anstatt unter der des Service Requesters.

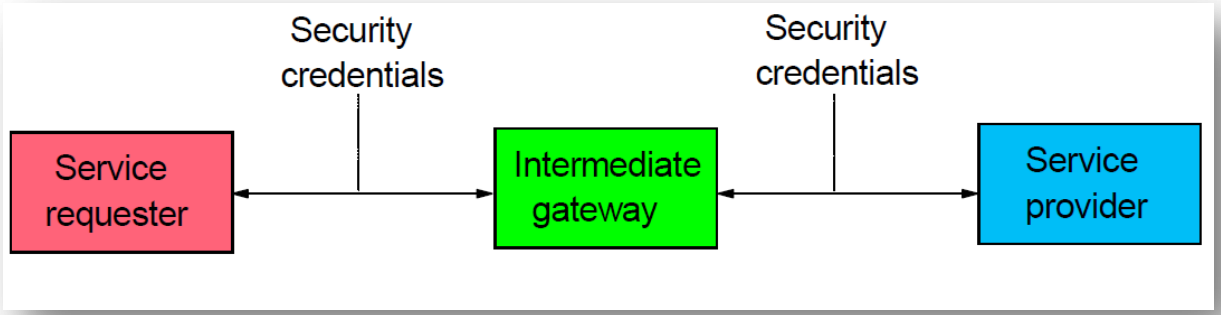

Abbildung 4.2: Sicherheit mit einem Intermediate Gateway auf Transportebene

WS-Security umgeht dieses Problem, in dem es der SOAP Nachricht ermöglicht, (sicherheitsrelevante) Informationen (so genannte „Security Credentials“) mitzugeben. Dadurch wird ermöglicht, dass die Informationen, die der Service Requester dem Service Provider vorweisen will (wie z.B. Password,…) auch über Intermediate Gateway hinweg, unverändert bleiben, wie folgende Abbildung 4.3 zeigt.

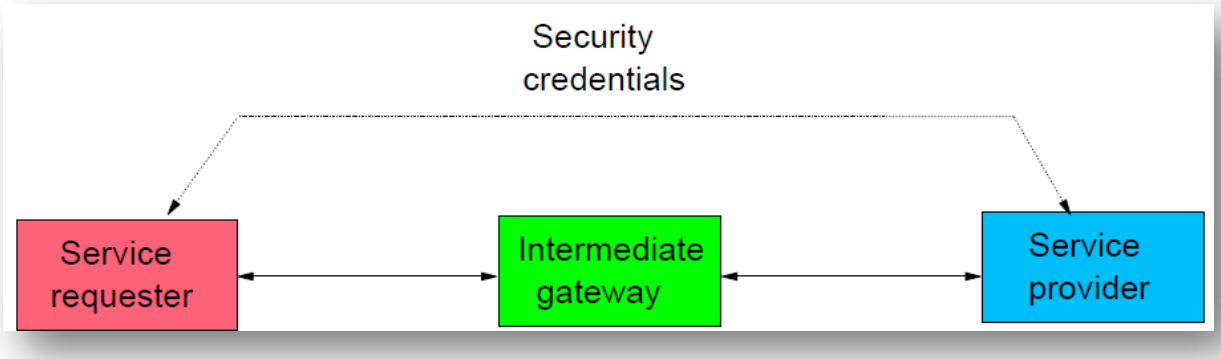

Abbildung 4.3: SOAP Nachrichten Sicherheit mit einem Intermediate Gateway

Die folgende Abbildung 3.4 zeigt wie eine SOAP Nachricht mit sicherheitsrelevanten Daten erweitert werden kann. Diese werden benutzt, um den Service Requester zu authentifizieren und um die Nachricht, auf ihrem Weg vom Requester zum Service Provider, zu schützen. Im Netzwerk, dass sich zwischen Requester und Provider befindet (Internet), könnte auch viele nicht vertrauenswürdigen Intermediate Knoten enthalten.

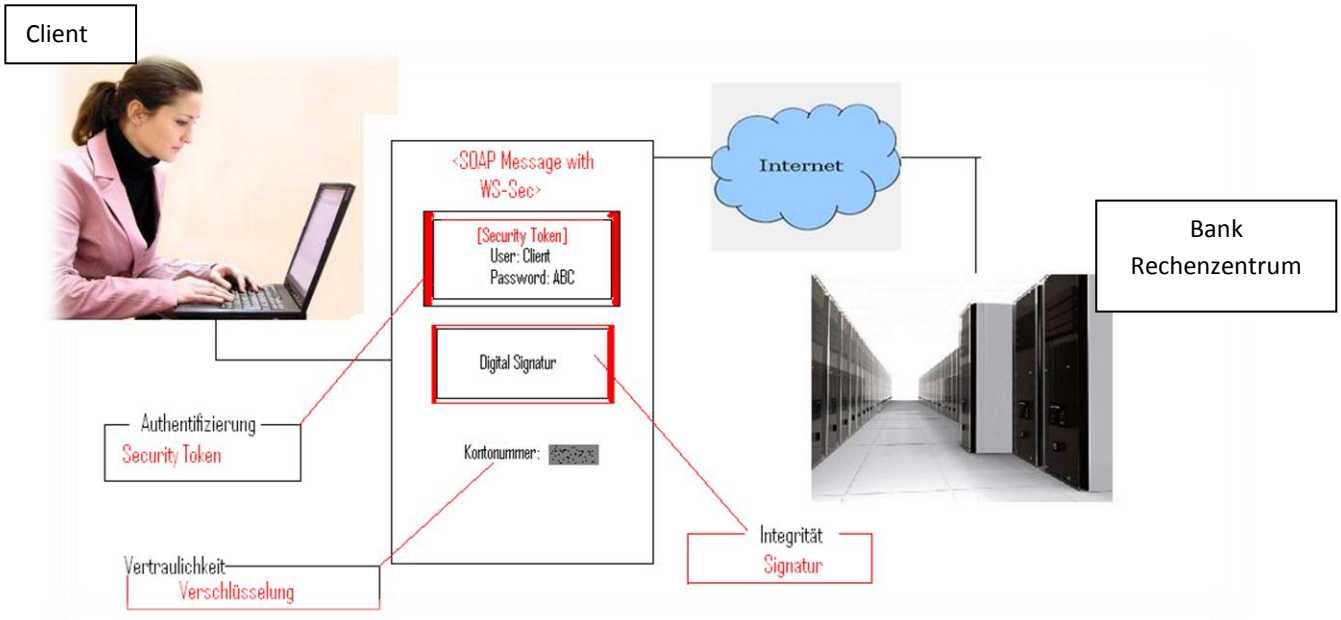

Abbildung 4.4: SOAP Nachricht mit Security Erweiterung

Die SOAP Nachricht in der Abbildung zeigt drei Aspekte der Sicherheit:

- Ein Security Token, das benutzt wird, um den Client zu Authentifizieren und Identifizieren

- Eine XML Digital Signatur, die sicherstellt, dass die Nachricht während ihrer Beförderung vom Requester zum Provider von niemandem verändert wird.
- Ein verschlüsseltes XML Element für die Kontonummer, das die Geheimhaltung sicherstellt.

4.2 Web Service-Security Model Framework

WS-Security spricht nur einen Teil der Security Services an. Durch das Benutzen von WS-Security kann man Authentifizierung, Integrität und Geheimhaltung auf *Nachrichtenebene* anwenden. Um auch andere Sicherheitsaspekte wie zum Beispiel das Protokollieren vom Ein und Ausloggen abzudecken, wird ein allgemeineres Sicherheitsmodell benötigt. Die Definitionen dieser Anforderungen für solche Sicherheitsaspekte werden im Web Service Security Model Framework erläutert.

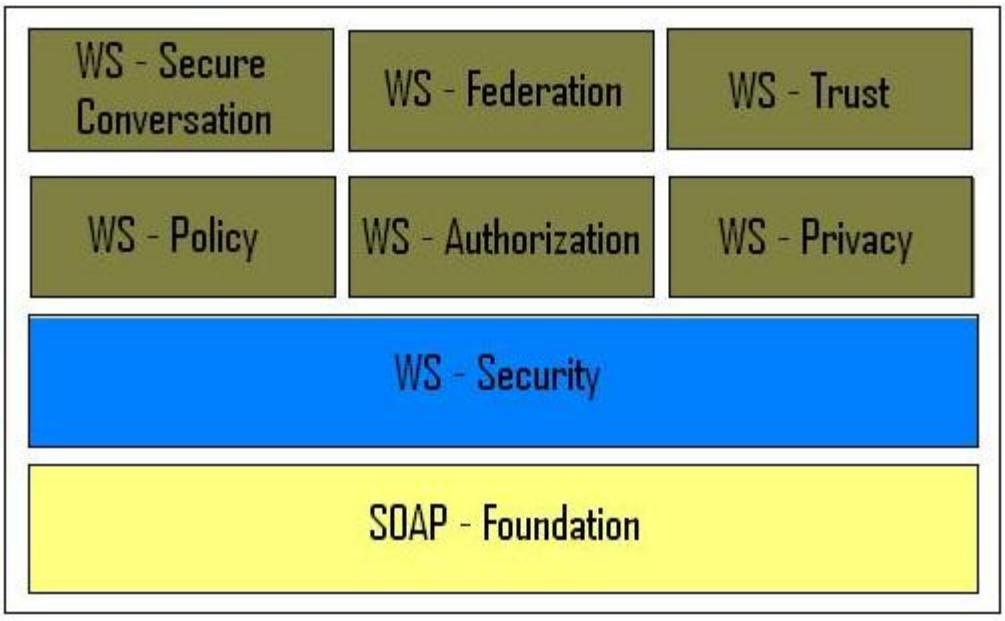

Abbildung 4.5: WS-Security Model Framework

- **WS-Secure Conversation:**

  ermöglicht auf Nachrichtenebene das, was SSL auf Transportebene leistet - einen Sicherheitskontext herzustellen, der den sicheren Austausch von Nachrichten gewährleistet.

- **WS-Federation:**

  unterstützt die Errichtung von Vertrauensverhältnissen bzw. einheitlichen, organisationsübergreifenden Sicherheitsrichtlinien um z.B. Kerberos- und X.509-Systeme zusammenführen zu können.

- **WS-Policy:**

  spezifiziert die Möglichkeiten, wie Web Services ihre Sicherheitsrichtlinien beschreiben und kommunizieren können. Untergeordnete Teilmengen davon sind WS-PolicyAssertions und WS-PolicyAttachment.

- **WS-Trust:**

  ermöglicht die Errichtung von übergeordneten Vertrauensverhältnissen für den sicheren Austausch von Nachrichten.

- **WS-Privacy:**

  wird innerhalb von P3P (Platform for Privacy Preferences) implementiert, und dient zur Kommunikation privater Richtlinien, die der Sender einer SOAP Nachricht erfüllen muss um einen Web Service aufzurufen.

- **WS-Authorization:**

  beschreibt wie die Autorisierungsdaten und die Autorisierungsregeln zu verwalten sind.

Für weitere Informationen zu “WS-Security – SOAP Message Security” wird auf folgende Quelle verwiesen:[OASIS01]

Die Kombination dieser Sicherheitsspezifikationen ermöglicht es viel mehr Szenarien zu entwickeln, die mittels Basis Sicherheitsmechanismen, wie Sicherheit auf Transportebene oder XML Dokumentenverschlüsselung, schwer oder sogar unmöglich zu implementieren sind.

4.3. CICS und SOAP Nachrichtensicherheit

Der im CICS Transaction Server vorhandene Security Handler unterstützt folgende Sicherheitsfunktionen, um einmal eine User ID einer einkommenden Nachricht zu entnehmen. Bereits erläuterte Beispiele hierfür waren die HTTP Basisauthentifizierung, das X.509 Zertifikat und die Kommunikation mit einem vertrauenswürdigen Dritten. Zum

anderen bietet es verschiedene Möglichkeiten, einer ausgehenden Nachricht einen Security Token anzuhängen wie z.B. X.509 Zertifikat und Kommunikation mit einem vertrauenswürdigen Dritten.
Es bietet auch die Möglichkeit die Signaturen von einkommenden Nachrichten zu prüfen, sowie die Generierung von Signaturen für den SOAP Body bei ausgehenden Nachrichten.
Eine weitere Funktion ist das Entschlüsseln von verschlüsselten Daten der einkommenden Nachrichten, sowie das Verschlüsseln des Inhaltes von dem SOAP Body der ausgehenden Nachrichten.

4.4 SOAP Message Security – Optionen

Es gibt verschiedene Optionen, die der „CICS WS-Security Support“ ermöglicht und welcher benutzt werden soll hängt von dem Grad der Sicherheit ab, der für die Daten benötigt wird und den Weg, den die Daten beschreiten.
Folgende SOAP Nachrichten Security Optionen werden in dem folgenden Abschnitt erläutert:

- AUTHENTIFIZIERUNG
- VERSCHLÜSSELUNG
- SIGNIERUNG

4.4.1 Authentifizierung

WS-Security bieten einen allgemeinen Mechanismus an, um einzelne Nachrichten zu authentifizieren in dem den Nachrichten Security Token mitgegeben werden. Es ist dabei nicht erforderlich, dass man einen bestimmten Security Token benutzt. Vielmehr standen beim Design die Erweiterbarkeit und die Unterstützung verschiedener Security Token Formate im Vordergrund, um eine möglichst große Anzahl von Authentifizierungsmechanismen zu bieten.
Das folgende Beispiel ist eine SOAP Anforderung (aus dieser Diplomarbeit), ohne Security.

```
<soapenv:Envelope
     xmlns:q0="http://www.getNameLengthI.com/schemas/getNameLen
               gthIInterface"
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <q0:getNameLengthRequest>
      <q0:Request>
        <q0:firstName>Usama</q0:firstName>
        <q0:name>Abu Al Aaish</q0:name>
      </q0:Request>
    </q0:getNameLengthRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Beispiel 4.1 SOAP Nachricht ohne Security

Wie man in diesem Beispiel 3.1 sehen kann, besitzt diese SOAP Nachricht gar keine Header. Um WS-Security anzuwenden, wird nun ein SOAP Security Header eingefügt.

WS-Security definiert einen Satz an Vokabeln, die in einem SOAP Envelope benutzt werden kann. Das XML-Element <wsse:Security> ist der Container für sicherheitsrelevante Informationen. (wsse steht dabei für Web Service Security Extension). Der Präfix wsse ist dabei willkürlich gewählt. Auch jedes andere Präfix kann benutzt werden vorausgesetzt das es mit dem Namespace Definition übereinstimmt.

Wenn man WS-Security für die Authentifizierung benutzt, wird ein Security Token innerhalb des SOAP Headers eingebettet und wird von Nachrichtensender zum eigentlichen Nachrichtenempfänger geleitet. Auf der Empfängerseite obliegt es der Verantwortung des Server Security Handler den Security Token zu authentifizieren und die ID des Aufrufers einzurichten.

Im folgenden Beispiel wird dieselbe SOAP Nachricht gezeigt, diesmal jedoch mit der Authentifizierung. Wie man sieht befinden sich ein Benutzername und ein Passwort innerhalb des Elementes <UsernameToken>.

```
<soapenv:Envelope
     xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
     xmlns:q0="http://www.getNameLengthI.com/schemas/getNameLen
     gthIInterface"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
          xmlns:wsse="http://docs.oasis-
          open.org/wss/2004/01/oasis-200401-wss-wssecurity-
          secext-1.0.xsd">
          <wsse:UsernameToken>
               <wsse:Username>aaish</wsse:Username>
               <wsse:Password Type="http://docs.oasis-
                    open.org/wss/2004/01/oasis-200401-wss-
                    username-token-profile-1.0#PasswordText">
                    MeinPassword
               </wsse:Password>
          </wsse:UsernameToken>
     </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <q0:getNameLengthRequest>
          <q0:Request>
              <q0:firstName>Usama</q0:firstName>
              <q0:name>Abu Al Aaish</q0:name>
          </q0:Request>
     </q0:getNameLengthRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Beispiel 3.2 SOAP Nachricht mit Username Security Token im Header

Das `<UsernameToken>` Element der SOAP Nachricht in dem Beispiel beinhaltet die Credentials, die benutzt werden um den User zu authentifizieren.

Die einfachste Form des Security Token ist der UsernameToken, welches benutzt wird um ein Username und ein (optionales) Passwort zur Verfügung zu stellen. Ein „Signed Security Token“ ist eins, das kryptografisch von einer bestimmten dazu befugten Instanz signiert wurde. Ein Beispiel für eine Signed Security Token wäre das X.509 Zertifikat.

Das Anwenden von Security Token für WS-Security wird in verschiedenen sogenannten Profiles, wie zum Beispiel dem Username Token Profile und dem X.509 Token Profile definiert.

Weitere Informationen über Security Token Standards findet man auf folgenden Seiten:

- Web Services Security: UsernameToken Profile V1.0 (March 2004):
  [OASIS02]
- Web Services Security: X.509 Token Profile V1.0 (March 2004):
  [OASIS03]

Wenn SOAP Nachrichten Authentifizierung in CICS implementieren werden soll, besteht die Möglichkeit unter folgenden Optionen zu wählen:

- Basic Authentification

  Im Server Provider Modus akzeptiert CICS für die Authentifizierung einen UsernameToken im SOAP Nachrichten Header der einkommenden SOAP Nachrichten. Der UsernameToken enthält einen Username Element und ein Password Element. CICS überprüft den erhaltenen Benutzername und das Passwort mittels eines externen Sicherheitsmanager wie RACF. Sollte dies erfolgreich sein, platziert CICS den Usernamen in dem Container DFHWS-USERID und verarbeitet die SOAP Nachricht in der Pipeline. Wenn es CICS nicht möglich sein sollte den UsernameToken zu überprüfen, dann gibt CICS eine SOAP Fehlermeldung an den Service Requester zurück.

- X.509 Certificate Authentification

  Ein X.509 Zertifikat, das für das Signieren verwendet wird, kann ebenso für die Authentifizierung angewendet werden.
  Im Server Provider Mode bildet CICS das Zertifikat zu einer RACF User ID ab und legt die User ID in dem Container DFHWS-USERID ab.

- Trusted Authentication

  In Server Provider Pipelines kann CICS einen Username Token oder ein X.509 Zertifikat im SOAP Message Header vertrauen. Bei dem Benutzen eines Username Token ist ein Password überflüssig. Wenn ein X.509 Zertifikate verwendet wird, dann wird das Zertifikat zu einer RACF User ID abgebildet. CICS vertraut der Identität des Providers und legt die User ID in dem Container DFHWS-USERID ab.

Diese Option ist eine Form der „Identity Assertion“ bei welcher eine bereits authentifizierte Benutzeridentität an CICS weitergeleitet wird und CICS die Identität, ohne eine erneute Authentifizierung durchführen zu müssen, akzeptiert.

- Advanced Authentication

  In Service Provider (oder auch Requester) Pipelines können Security Token für Authentifizierungszwecke mittels einem Security Token Service (STS) überprüft oder ausgetauscht werden. Das STS ermöglicht CICS das Akzeptieren und Senden von Nachrichten, die Security Token in dem Message Header enthalten, die jedoch normalerweise nicht unterstützt werden; zum Beispiel LTPA oder Kerberos Tokens oder SAML assertions.

  Für einkommende Nachrichten besteht die Möglichkeit zwischen der Überprüfung und dem Austausch von Security Token zu wählen. Wenn die Wahl auf das Austauschen von Security Tokens fällt, dann muss CICS einen Username Token vom STS erhalten. Für abgehende Nachrichten ist nur der Austausch von Username Token für ein Security Token möglich.

4.4.2 Signieren von SOAP Messages

Integrität wird auf SOAP Nachrichten angewendet, damit sichergestellt wird, dass keine illegalen Modifikationen der Nachricht vorgenommen werden, während diese unterwegs sind. Im Wesentlichen wird Integrität ermöglicht in dem eine digitale XML Signatur auf den Inhalt der SOAP Nachricht generiert wird. Sollten die Daten illegal verändert werden, würde die Signatur nicht länger gültig sein.

Die Erstellung einer Signatur basiert auf einem Schlüssel, den nur der Sender hat (und geheim hält). Ein Sniffer hat diesen Schlüssel nicht, und kann somit diese Signatur nicht erstellen. Wenn der Empfänger die Nachricht erhält, erstellt dieser ebenfalls eine Signatur. Hierfür benutzt er den Inhalt der Nachricht. Nur wenn beide Signaturen übereinstimmen, dann erkennt der Empfänger diese Nachricht an. Sollten sie nicht übereinstimmen, dann wird eine SOAP Fehlermeldung an den Sender diese Nachricht geschickt.

Für eine Nachricht, die an CICS verschickt wurde, kann der von CICS zur Verfügung gestellte Security Message Handler (der sich in der Pipeline befindet), die Digitale Signatur einzelner Elemente im SOAP <Header> und dem <BODY> überprüfen. Der Security Handler überprüft folgende Elemente:

- Überprüfung von signierten Elementen, die es im SOAP <Header> vorfindet
- Überprüfung von signierten Elementen im SOAP <Body> vorfindet. Wenn der Handler so konfiguriert wurde, dass es einen signierten Body erwartet, dann wird CICS jede nicht signierte SOAP Nachricht mit einer Fehlermeldung zurückgewiesen.

Ein Vorteil von digitalen XML Signierungen gegenüber SSL/TLS ist der, dass die Signatur vor Tampering geschützt wird, selbst wenn sie mehrere Intermediary Server durchläuft. Wenn jedoch ein Prozess, der auf einem Intermediary Server läuft und den Inhalt von Nachrichten verändert, dann sollte darauf geachtet werden, dass dieser Prozess nicht die Signatur, die sich in der Nachricht befindet, verändert, da ansonsten die Gültigkeitsprüfung der Signatur in CICS fehlschlagen würde.
Ein Beispiel für eine Signierte SOAP Nachricht befindet sich in Appendix A.

4.4.3 Verschlüsselung von SOAP Nachrichten

XML Verschlüsselung wird auf SOAP Nachrichten angewendet, um sicherzustellen, dass nur der vorgesehene Empfänger der Nachricht die Nachricht lesen kann.

Geheimhaltung wird durch das Verschlüsseln der Inhalte einer SOAP Nachricht mittels XML Encryption erreicht. Wenn die SOAP Nachricht einmal verschlüsselt ist, dann kann nur ein Service, der den Schlüssel kennt, diese Nachricht entschlüsseln und lesen.

CICS benutzt bei einkommenden Nachrichten den von CICS bereitgestellten Security Message Handler. Dieser kann einzelne Elemente im SOAP <Body> und Elemente des SOAP <Header> entschlüsseln, sollte der SOAP Body ebenfalls verschlüsselt sein.
Der Security Message Handler entschlüsselt immer folgende Elemente:

- Elemente die es im <Header> vorfindet und zwar in der Reihenfolge in der sie stehen.

- Elemente im SOAP <Body>. Sollte der Handler so konfiguriert sein, dass es einen Verschlüsselten Body erwartet, dann wird CICS jedesmal eine SOAP Fehlermeldung werfen, sollte der SOAP Body nicht verschlüsselt sein.

Ein Vorteil der Benutzung von XML Encryption gegenüber SSL/TLS ist der, dass Verschlüsselte Nachrichten ihre Geheimhaltung aufrechterhalten, selbst wenn die Nachricht mehrere Intermediary Server durchläuft. Durch SSL/TLS ist die Nachricht während dem Transport gesichert, wenn die Nachricht jedoch von (sich möglich auf dem Transportweg befindenden) Intermediaries empfangen und weitergeleitet werden kann, dann kann eine End-To-End Kommunikation nicht garantiert werden.

XML Encryption ermöglicht es auch nur bestimmte vertrauliche Teile Nachricht zu verschlüsseln, anstatt die ganze Nachricht zu verschlüsseln. Ein wichtige Faktor, der berücksichtigt werden sollte, wenn man Intermediaries benutzt ist der, dass die XML Encryption eventuell ein Routing, dass auf den Inhalt basiert (content-based routing) der SOAP Nachrichten durchführt, weil es den Intermediary Server nicht möglich ist, alle Teile des Message Body zu lesen.

Die XML Encryption der SOAP Nachrichten verursacht einen erhöhten CPU Verbrauch. Der CICS Support für XML Encryption ist von den Integrated Cryptograhic Service Facility (ICSF) Services abhängig und deshalb sind die Konfiguration und das Startup eine Voraussetzung, um diesen Support überhaupt benutzen zu können. Für weitere Informationen Rund über ICSF und sowie deren Konfiguration wird auf die Literatur [IBM01] (Seite 219 ff.) verwiesen.
Ein Beispiel für eine SOAP Nachricht mit Verschlüsselung befindet sich in Appendix B.

4.4.4 Konfiguration des von CICS bereitgestellten Security Handlers

Um WS-Security im CICS Transaction Server zu implementieren, muss ein <wsse_handler> in die Konfigurationsdatei (der entsprechenden Pipeline) eingefügt werden. Dieses Element befindet sich innerhalb des <service_handler_list> Elementes und beinhaltet das <dfhwsse_configuration> Element, welches die Konfigurationsinformationen für den Security Handler „DFHWSSE1“ zur Verfügung stellt, welches Support für die Absichern von Web Services ermöglicht.

<dfhwsse_configuration> besitzt das Attribut „version“, das als Wert einen Integer hat, der für die Version der Konfigurationsinformation steht. Nur der Wert „1“ ist ein gültiger.

Desweiteren beinhaltet es folgende Elemente:

1. Eines der folgenden Elemente:

a) Ein optionales <authentication> Element.

- In einer Service Requester Pipeline bestimmt dieses Element den Typ der Authentifizierung, die der Security Handler auf abgehenden SOAP Nachrichten anwenden wird.
- In einer Service Provider Pipeline bestimmt das Element <authentication>, ob CICS den Security Token für einkommende SOAP Nachrichten benutzt, um die User ID zu ermitteln unter der die weitere Verarbeitung laufen soll.

Das <authentication> Element hat zwei Attribute: „trust“ und „mode“.
Diese Attribute ermitteln, ob „Asserted Identity“ benutzt wird und die Kombination von Security Tokens in SOAP Nachricht benutzt werden.
Das „Trust“ Attribut kann entweder auf „none“, „basic“ , „signatur“ und neu in der CICS TS Version 3.2 auf „blind“ gesetzt werden. Das „mode“ Attribut kann auf „none“, „basic“ oder „signatur“ gesetzt werden.

Zum Beispiel könnte ein „Trust Token“ ein UsernameToken oder ein X.509 Token sein. Das Trust Token wird für dafür benutzt, um zu überprüfen, ob der Sender die richtigen Rechte für eine Asserted Identity besitzt und ob der Identity Token die Asserted Identity (User ID) enthält, unter welcher die Anfrage laufen wird.

Anmerkung: Identity Assertion (Zusicherung der Identität):
Wenn die Authentifizierungsmethode IDAssertion verwendet wird, ist das generierte Sicherheitstoken ein Element <wsse:UsernameToken>, das ein Element <wsse:Username> enthält.
Auf der Seite des Anforderungssenders wird ein Callback-Handler aufgerufen, der das Sicherheitstoken generiert. Auf der Seite des Anforderungsempfängers wird die Gültigkeit des Sicherheitstoken geprüft.

Für weitere Informationen über die Bedeutung und die möglichen Kombinationen der Trust und Mode Attribute, verweise ich auf die Literaturen:

- [IBM03]
- Security Szenarien: [IBM05] (Kapitel 8 und Kapitel 9)

Das <authentication> Element kann folgende Elemente Enthalten:

- <certifcate_label>

  Optional. Beschreibt das Label, das in Verbindung mit einem digitalen X.509 Zertifikat in Verbindung steht. In einer Service Provider Pipeline wird dieses ignoriert.

- <suppress/>

  Optional. Der Handler eines Service Providers wird keinerlei Security Token in der Nachricht benutzen, um zu bestimmen unter welcher User ID es laufen wird. Der Handler eines Service Requesters wird der SOAP Nachricht keine Security Tokens anfügen, die für die Authentifizierung benötigt werden.

- <algorithm>

  Beschreibt die URI des Signaturalgorithmus. Wenn die Kombination der Werte der „Trust“ und „Mode“ Attribute zeigen, dass die Nachrichten signiert sind, muss dieses Element spezifiziert werden. In der Folgenden Tabelle werden die Algorithmen samt URI gezeigt, die hierfür in Frage kommen.

Algorithmus	URI
Digital Signature Algorithmus mit Secure Hash Algorithmus 1 (DSA mit SHA1)	http://www.w3.org/2000/09/xmldsig#dsa-sha1
Rivest-Shamir-Adleman Algorithmus mit Secure Hash Algorithmus 1 (RSA mit SHA1)	http://www.w3.org/2000/09/xmldsig#rsa-sha1

Abbildung 3.6: Signaturenalgorithmen für einkommende SOAP Nachrichten

b) Ein optionales `<sts_authentication>` Element.

Das Action-Attribut dieses Elementes beschreibt welche Art von Anfrage an den Security Token Service (STS) gesendet werden soll: *issue* oder *validate (ausstellen* oder *überprüfen)*

- Wenn nach einer Ausstellung eines ID Token angefragt wird, dann benutzt CICS die Werte in den verschachtelten Elementen, um ein ID Token des spezifizierten Typs anzufragen.
- Wenn nach einer Überprüfung eines ID Token angefragt wird, dann benutzt CICS die Werte in den verschachtelten Elementen, um den STS für eine Überprüfung des ID Token des spezifizierten Typs anzufragen.

Wenn man ein `<sts_authentication>` Element spezifiziert, dann muss ebenso ein `<sts_endpoint>` Element spezifiziert werden. Ist dieses Element vorhanden, dann benutzt CICS die URI in dem `<endpoint>` Element, um eine Anfrage an den STS zu senden.

Innerhalb eines `<sts_authentication>` Element werden folgende Elemente codiert:

- Ein `<auth_token_type>` Element. Dieses Element wird benötigt, wenn ein `<sts_authentication>` Element in einer Service Requester Pipeline oder optional in einer Service Provider Pipeline, spezifiziert wird.

In einer Server Requester Pipeline ermittelt das <auth_token_type> Element den Typ des Tokens, das der STS ausstellen wird, wenn CICS diesem die User ID, die sich im DFHWS-USERID Container befindet, zuschickt. Den Token, den CICS erhalten von dem STS dann erhält, wird im Header der ausgehenden Nachricht untergebracht.

In einer Service Provider Pipeline wird das <auth_toke_type> Element benutzt, um zu bestimmen welches ID Token CICS der Nachricht entnimmt und dann dem STS zu Überprüfung oder zum Austausch, vorlegt. CICS benutzt dabei den ersten ID Token des spezifizierten Typs im Nachrichten Header. Sollte man dieses Element nicht spezifizieren, dann benutzt CICS den ersten ID Token, den es im Nachrichten Header findet.

- Nur in einer Service Provider Pipeline: Ein optionales, leeres <suppress/> Element. Wenn dieses Element spezifiziert wird, dann versucht der Handler nicht ein Security Token in der Nachricht zu ermitteln, unter deren User ID die Sache laufen wird, inklusive der ID Token, die von dem STS zurückgegeben wird.

2. Ein optionales <sts_endpoint> Element.

   Dieses Element wird nur dann benutzt, wenn auch das <sts_authentication> Element benutzt wird.

   Im <sts_endpoint> Element, wird ein <endpoint> Element codiert. Dieses Element beinhaltet eine URI, die auf den Standort des STS im Netzwerk zeigt.

   Der STS Endpoint kann als HTTP Endpoint oder als WebSphere MQ Endpoint (in dem das JMS Format der URI benutzt wird), spezifiziert werden.
   (Für mehr Sicherheit wird empfohlen für die Verbindung zum STS entweder SSL oder TLS zu benutzen.)

3. Ein optionales, leeres `<expect_signed_body/>` Element:

   Das `<expect_signed_body/>` Element zeigt an, dass der `<body>` einkommender Nachrichten signiert sein muss. Sollte der Body der einkommenden Nachricht nicht richtig signiert sein, weist CICS diese mit einer Fehlermeldung zurück.

4. Ein optionales, leeres `<expect_encrypted_body/>` Element:

   The `<expect_encrypted_body/>` Element zeigt an, dass der `<body>` einkommender Nachrichten verschlüsselt sein muss. Wenn der Body der einkommenden Nachrichten nicht richtig verschlüsselt ist, dann weist CICS diese mit einer Fehlermeldung zurück.

5. Ein optionales `<sign_body>` Element:

   Wird dieses Element verwendet, dann signiert CICS den `<body>` ausgehender Nachrichten. Es beinhaltet folgende Elemente:

   - `<algorithm>`

     Beschreibt die URI des Algorithmus, der für die Signierung des Body der SOAP Nachricht verwendet wurde.

     CICS unterstützt dabei die folgenden Signaturalgorithmen für ausgehende SOAP Nachrichten:

     - Rivest-Shamir-Adleman Algorithmus mit Secure Hash Algorithm 1 (RSA mit SHA1), dessen Spezifikation sich unter folgenden URI befindet: http://www.w3.org/2000/09/xmldsig#rsa-sha1

   - `<certificate_label>`

     Beschreibt das Label des mit einem digitalen X.509 Zertifikat in Verbindung steht. Das digitale Zertifikat sollte den privaten Schlüssel beinhalten, sollte dieser für das Signieren der Nachricht benutzt worden sein. Der Öffentliche Schlüssel, der in Verbindung mit dem privaten Schlüssel steht, wird dann in der SOAP Nachricht mitgeschickt, was eine Überprüfung Signatur erst ermöglicht.

6. Ein optionales `<encrypt_body>` Element:

   Wenn dieses Element vorkommt, dann wird CICS den `<body>` ausgehender Nachrichten verschlüsseln. Es beinhaltet folgende Elemente:

   - `<algorithm>`

     Beschreibt die URI, die den Algorithmus feststellt, der benutzt wird, um den Body der SOAP Nachricht zu verschlüsseln. Die in der folgenden Tabelle gezeigten Verschlüsselungsalgorithmen werden von CICS unterstützt:

Algorithmus	**URI**
Triple DES im cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
AES mit einer Schlüssellänge von 128 bits im Cipher Block Chaining Modus	http://www.w3.org/2001/04/xmlenc#aes128-cbc
AES mit einer Schlüssellänge von 192 bits im Cipher Block Chaining Modus	http://www.w3.org/2001/04/xmlenc#aes192-cbc
AES mit einer Schlüssellänge von 256 bits im Cipher Block Chaining Modus	http://www.w3.org/2001/04/xmlenc#aes256-cbc

Abbildung 3.7: Von CICS unterstützten Verschlüsselungsalgorithmen

   - `<certificate_label>`

     Beschreibt das Label der in Verbindung mit einem digitalen X.509 Zertifikat steht. Das digitale Zertifikat sollte den Öffentlichen Schlüssel des Empfängers der SOAP Nachricht beinhalten, so dass es mit dem privaten Schlüssel wieder entschlüsselt werden kann, sobald die Nachricht erhalten wurde.

7. Nur in der Provider Pipeline anwendbares, ein optionales `<reject_signature/>` Element:

   Wenn dieses Element vorhanden ist, wird CICS jede Nachricht zurückweisen, die in deren Header ein Zertifikat enthält, das nur einen Teil oder den komplette Message Body signiert hat. Der Web Service Requester erhält in diesem Fall eine SOAP Fehlermeldung.

Daher wird empfohlen das <reject_signature/> Element nur dann zu benutzen, wenn CICS keine signierten SOAP Nachrichten verarbeiten soll und dient dem Schutz vor den „Denial of Service“ Attacken.

8. Nur in der Provider Pipeline, ein optionales <reject_encryption/> Element:

   Ist diese Element vorhanden, dann weist CICS jede Nachricht zurück, die entweder nur teilweise oder vollständig verschlüsselt ist. In diesem Fall wird ebenfalls eine SOAP Fehlermeldung generiert und diese dem Web Service Requester zugeschickt. Daher wird empfohlen das <reject_encryption/> Element nur dann zu benutzen, wenn CICS keine verschlüsselten SOAP Nachrichten verarbeiten soll. Dies kann sehr nützlich sein, wenn vor den „Denial of Service“ Attacken geschützt werden soll.

```
   .          .
   .          .
   .          .
<wsse_handler>
   <dfhwsse_configuration version="1">
     <authentication trust="blind" mode="basic">
          </authentication>
  </dfhwsse_configuration>
</wsse_handler>

.          .
.          .
.          .
```

Beispiel 3.3 Codebeispiel für wsse_handler einer Provider Pipeline Configfile

Das <wsse_handler> Element beinhaltet ein <dfhwsse_configuration> Element, dass Informationen über die Konfiguration des Security Handler bereitstellt.

In Zusammenhang mit STS gilt zusätzlich:
Würde ein <sts_authentication> Element verwendet werden, so würde das action=“issue“ Attribut veranschaulichn, dass CICS eine Security von dem STS anfragen wird.

Ein <auth_token_type> würde spezifizieren, welches ID Token CICS vom dem SOAP Header entnimmt und dem STS zusendet.

Ein <sts_endpoint> Element beinhaltet ein Kindelement, <endpoint>, welches die URI beinhaltet, die auf den Standort des STS im Netzwerk zeigt. Dies ist ein HTTPS Endpunkt.

4.4.5 Benutzerdefinierte Security Handlers

In eigenen Fällen will man auch eigene Sicherheitsprozeduren und Sicherheitsverarbeitung anwenden. Daher besteht auch die Möglichkeit einen benutzerdefinierten Security Handler zu schreiben, der die gesicherten SOAP Nachrichten verarbeitet.

Dabei muss entschieden werden, welches Maß an Sicherheit der Security Handler unterstützen soll und es sollte sichergestellt werden, dass eine entsprechende SOAP Fehlermeldung zurückgegeben wird, wenn die Nachricht Sicherheitsaspekte beinhaltet, die nicht unterstützt werden. (z.B. Es wird ein Teil der SOAP Nachricht verschlüsselt, jedoch ist der Security Handler für Verschlüsselung nicht konfiguriert. In diesem Fall müsste eine Fehlermeldung an den Sender geschickt werden, die besagt, dass die Verschlüsselung nicht unterstützt wird.)
Ein Beispiel für solch einen Benutzergeschriebenen Security Handler findet man unter:
[IBM01]

4.4.6 Vergleich von Sicherheit auf Transport Level und SOAP Nachrichten Sicherheit

Es besteht die Möglichkeit Web Service Security auf zwei Ebenen zu implementieren:

- Auf der Transportebene mittels SSL
- Auf der SOAP Nachrichten Ebene.

Wenn die Web Service Umgebung einfach ist (z.B. erstreckt sie sich nicht über mehrere Knoten), dann sollte eine Sicherheitslösung, die auf Transport-Level Security basiert ausreichen. Für komplexere Szenarien jedoch würde dies nicht ausreichen.

Der folgende Abschnitt dient daher als Leitfaden, um zum einen die Entscheidung zu erleichtern, sich für einen Typ der Sicherheitslösung zu entscheiden und diese zu implementieren.

- Folgende Gesichtspunkte sprechen für eine Sicherheit auf Transport-Level, um die CICS WS Environment abzusichern:

  - In der Web Services Environment werden keine Intermediaries benutzt. Sollten sich jedoch Intermediaries auf dem Transportweg befinden, kann trotzdem garantiert werden, dass wenn die Daten nicht verschlüsselt sein sollten, auf sie trotzdem von keinen unberechtigtem Dritten (sei es nun ein Knoten oder Prozess) zugegriffen werden kann.

  - Der Transport ist nur HTTP –basierend.

  - Sollte Performance ein wichtiger Punkt sein:
    SSL/TLS ist eine ausgereifte Technologie, die über eine lange Zeitspanne hinweg, immer wieder optimiert wurde. Es gibt daher Möglichkeiten die Performance zu optimieren, in dem man beispielsweise dauerhafte TCP/IP Verbindungen und SSL Session IDs wiederverwendet. Diese Optimierungen bedeuten, dass „teure“ (im Sinne von zeitaufwendig) Sicherheitsfunktionen, wie SSL Handshaking, vermieden werden können, wie sie bei einem Service Request und dem daraufhin folgenden eingeleiteten Handshake auftreten können.
    WS-Security Support, zum Vergleich, sind komplett zustandslose und „teure“ Sicherheitsfunktionen, wie zum Beispiel die Überprüfung von digitalen Signaturen, die bei jedem Service Request wiederholt werden.

  - Der Web Service Client ist ein Stand-Alone Java Programm.
    WS-Security kann nur auf Clients angewendet werden, die in einer Web Service Umgebung laufen, die die WS-Security Spezifikation unterstützen. (zum Beispiel, ein WebSphere Application Server mit Web Service Feature Pack und als Anwendungsprogramm z.B. WebSphere Application Server Toolkit 6.1.1).

- Unter folgenden Gesichtspunkten ist eine WS-Security angebracht (zusätzlich zu der Transport-Level Security möglich):

- Es werden Intermediaries verwendet, unter denen es auch einige gibt, die man als nicht vertrauenswürdig (untrusted) erachtet.
  Security Credentials, die sich in einer SOAP Nachricht befindet, können über einige Intermediaries durchgereicht werden. Diese Vertraulichen Informationen innerhalb der aktuellen SOAP Nachricht würden zu einem Overhead führen, die durch das Verschlüsseln und Entschlüsseln via SSL an jedem Intermediary Knoten entsteht.
  Außerdem kann ein Intermediary dem CICS ein Authentifizierungsservice anbieten, so dass ein Intermediary Server, den Web Service Client authentifiziert und dann CICS eine Asserted ID zukommen lässt.

- Es kommen verschiedene Transportprotokolle zum Einsatz.
  WS-Security ist unabhängig vom darunterliegenden Transport Protokoll.

- Der Web Service Partner unterstützt WS-Security und es wurde entschieden, dass Security Tokens, den WS-Security Spezifikationen entsprechend, benutzt werden.
- Falls eigene Sicherheitsprozeduren- und Verarbeitungen verwendet werden sollen, kann diese realisiert werden, in dem ein benutzerdefiniertes Message Handler Programm geschrieben wird, der die SOAP Nachrichten mit den Sicherheitsaspekten in der Pipeline verarbeiten wird.

Kapitel 5 - CICS – SSL

Um einen sicheren Zugriff auf CICS zu gewährleisten, besteht die Möglichkeit SSL in Zusammenhang mit CICS zu benutzen. In diesem Kapitel wird erläutert, welche Idee dahinter steckt und was seitens des Clients, sowie des Servers, realisiert werden muss, damit CICS SSL benutzt werden kann.

5.1 Transport Sicherheit unter Benutzung von HTTP als Transportprotokoll

Wenn CICS Web Services mittels HTTP aufgerufen werden, kann eine Basis Authentifizierung benutzt werden, um einen Web Service Client zu authentifizieren. SSL/TLS kann angewandt werden, um sowohl den Web Service Client zu authentifizieren, als auch Nachrichtenintegrität und Vertraulichkeit zu gewährleisten.

5.1.1 Basisauthentifizierung

HTTP Basisauthentifizierung ist eine einfaches „Challenge and Response“ Mechanismus mit dem ein Server Authentifizierungsinformationen (Benutzer ID und Password) von einem Client anfordern kann. Der Client übergibt dem Server die Authentifizierungsinformationen in einem HTTP Autorisierungsheader. Die Authentifizierungsinformation wird in base-64 verschlüsselt. Das Basisauthentifizierungsprotokoll wird im RFC2617 spezifiziert [RFC01].

Das AUTHENTICATE Attribut in der CICS TCPIPSERVICE Ressourcen Definition spezifiziert die Authentifizierung und das Identifikationsschema, um es für einkommende TCP/IP Verbindungen des HTTP Protokolls anzuwenden. Man kann die HTTP Authentifizierung aktivieren, in dem das Attribut AUTHENTICATE auf den Wert BASIC gesetzt wird.

Eine CICS Service Provider Anwendung kann durch HTTP Basisauthentifizierung geschützt werden. Jedoch kann das HTTP Basisauthentifizierungsschema nur dann in Betracht gezogen werden, wenn die Verbindung zwischen dem Web Service Client und dem Web Service Provider (in diesem Fall, der CICS Umgebung) sicher ist. Sollte die Verbindung nicht sicher sein, dann kann dieses Prinzip nicht ausreichend Sicherheit bieten, um einen nicht autorisierten Benutzer davon abzuhalten die Authentifizierungsdaten eines Servers

herauszufinden und diese zu Verwenden. Wenn die Möglichkeit besteht, dass ein Passwort abgefangen werden kann, sollte die Basisauthentifizierung in Verbindung mit einer SSL/TLS benutzt werden, so dass von der SSL Verschlüsselung gebraucht gemacht wird, um die User ID und die Passwort Informationen zu schützen.

5.2 CICS Unterstützung für SSL/TLS

Die Secure Socket Layer (SSL) und die Transport Layer Security (TLS) Protokolle bieten Vertraulichkeit und Datenintegrität zwischen zwei Anwendungen, die über das Internet kommunizieren. CICS benutzt System SSL, um sowohl SSL 3.0 als auch TLS 1.0 Protokolle zu unterstützen. Eine CICS Service Provider Anwendung kann durch HTTPS (also einer HTTP Verbindung, die auf einer SSL/TLS Verbindung beruht) gesichert werden.
HTTPS Verbindungen werden automatisch das TLS 1.0 Protokoll verwenden, außer der Client schreibt ausdrücklich SSL 3.0 vor.

Für weiterführende Informationen über die Funktionsweise von SSL/TLS siehe:

- [IBM03] Seite 116 ff
- [RFC02]

HTTPS bringt folgende Vorteile mit sich:

- Es bietet einen schnellen und sicheren Transport für CICS Web Services.

- Es bietet für die Authentifizierung sowohl die Möglichkeit durch HTTP Basisauthentifizierung an, als auch die Möglichkeit durch ein Client X.509 Zertifikat zu authentifizieren.

- Es bietet Integrität für die Daten an, die von dem Service Requester an den Service Provider übergeben werden.

- Es bietet eine Geheimhaltung für die Daten an, die von dem Service Requester an den Service Provider übergeben werden, in dem effiziente Secret Key Kryptographie benutzt wird.

- Es kann die Kosten, die durch SSL Handshake entstehen, durch Benutzung von Kryptographischer Hardware, erheblich reduzieren. Die Verschlüsselungseinstellungen können eine Benutzer definierte Einstellung so vorgenommen werden, dass nur die Cipher Suites zum Einsatz kommen, die auch von der Integrated Cryptographic Service Facility (ICSF) benutzt werden.

- Es ist ausgereift und bei den meisten Herstellern ähnlich implementiert, was dadurch zu wenigen Kompatibilitätsproblemen führt. Wie SSL/TLS in einer CICS TS V3.2 Umgebung zu aktivieren ist wird im Folgenden erläutert.

Wenn CICS der Service Provider ist, dann muss ein Server Zertifikat beschafft werden. Diese X.509 Zertifikat kann von einer Zertifizierungsstelle (wie Verisign unter der Webadresse http://www.verisign.com/) beantragt werden.
Alternativ kann RACF benutzt werden, um Server Zertifikate zu erstellen. Wenn Server Zertifikate mittels RACF erstellt werden, muss der Client so konfiguriert werden, dass dieser die Server Zertifikate erkennt.

5.3 CICS SSL Benutzung

In diesem Abschnitt wird zuerst ganz allgemein erläutert, wie CICS konfiguriert und was in RACF erstellt werden muss, damit serverseitiges SSL benutzen werden kann. Im weiteren Verlauf werden dann auf die einzelnen Punkte eingegangen.

Um CICS SSL zu benutzen, wird wie folgt vorgegangen:

1. **Zertifikat erstellen**
   Diese könnten man beispielsweise von einer Zertifizierungsstelle wie www.verisign.com erhalten oder einfach in RACF mittels des Befehls RACDCERT erstellt werden.
   Für detaillierte Informationen dazu wird auf folgende Quelle verwiesen:
   [IBM06] Seite 557 ff.

2. **Key Ring erstellen**

   Ein Key Ring muss in der RACF Datenbank erstellt werden. Ein King Ring enthält

   a. die öffentlichen und privaten Schlüssel
   b. das Server Zertifikat
   c. Für jeden Client mit dem kommuniziert werden soll, jeweils ein Zertifikat der diesen autorisiert (authority certificate)

3. **CICS JCL definieren**

   Es muss sichergestellt werden, dass die CICS Region, entweder mittels den JOBLIB und STEPLIB Beschreibungen oder durch die Benutzung der System Link Bibliothek, Zugriff auf die z/OS System SSL Bibliothek namens „SIEALNKE“, hat.
4. **CICS SIT Parameter definieren**
5. **TCPIPSERVICE Ressourcen definieren**

5.3.1 Key Ring

In CICS werden die benötigten Server Zertifikate und die entsprechenden Informationen über die Zertifizierungsstellen in einem Key Ring in der RACF Datenbank aufbewahrt. Der Key Ring beinhaltet das öffentliche und private Schlüsselpaar des Systems zusammen mit den Server Zertifikaten und denen aller Zertifizierungsstellen, die die Zertifikate, der von den Client empfangenden Zertifikate, unterzeichnet haben.
Bevor man CICS mit SSL benutzen kann, muss man einen Key Ring erstellen, der ein öffentliches und privates Schlüsselpaar und ein Server Zertifikat enthält.

Der Befehl RACDCERT installiert und pflegt die privaten Schlüssel der Public Key Infrastruktur (PKI) und die Zertifikate in RACF. RACF unterstützt mehrfache private Schlüssel der PKI und Zertifikate, die als Gruppe verwaltet werden. Diese Gruppen nennt man Key Ringe.

Optional kann man ein Zertifikat im Key Ring als Standard Zertifikat bestimmen. Wenn ein Client-, oder Serveranfrage ein Zertifikat von CICS anfordert, wird das Standardzertifikat benutzt, außer man hat es anderweitig spezifiziert:

- Für einkommende HTTPS Anfragen, spezifiziert man das Zertifikat in der TCPIPSERVICE Ressourcendefinition
- Für abgehende HTTPS Anfragen spezifiziert man das Zertifikat in der URIMAP Ressourcendefinition.

Für weitere Informationen über das Erstellen von Key Ringen und Zertifikaten wird auf CICS [IBM07] verwiesen.

5.3.2 Neue Zertifikate erstellen

Zertifikate und Key Ringe werden mittels des RACDCERT Befehls erstellt.
Der Key Ring, der im KEYRING Systeminitialisierungsparameter spezifiziert wird und das Zertifikat in diesem Key Ring müssen in Verbindung mit der CICS Region User ID stehen.
Zunächst wird in dem RACDCERT Befehl die CICS Region User ID spezifiziert, der benutzt wird, um den Key Ring und das Zertifikat zu erstellen.

```
RACDCERT ID(cics-region-userid) ADDRING(ringname)
RACDCERT ID(cics-region-userid) ADD('datasetname') TRUST
```

Der User, der diesen Befehl anwenden will, benötigt UPDATE Rechte für die IRR.DIGTCERT.ADDRING Ressource in der FACILITY Klasse.

Der CICS Region User ID müssen READ Rechte zugewiesen worden sein, damit diese Zugriff auf die IRR.DIGTCERT.LISTRING Ressource in der FACILITY Klasse hat, und Zugriff auf IRR.DIGTCERT.*, um den Key Ring zu benutzen.

Anmerkung: CICS bietet ein in CLIST geschriebenes Skript namens DFH$RING an, das als Beispiel dienen soll, um somit das Erstellen von einem RACF KEYRING für CICS SSL, der Benutzung von EJB und IPCONNs vereinfachen soll. Das CLIST Beispielskript befindet sich in der Bibliothek *CICSTS32.CICS.SDFHSAMP.*

Die Ausführung von *DFH$RING clist* erfolgt folgendermaßen:

```
EXEC 'CICSTS32.CICS.SDFHSAMP(DFH$RING)' +
     'firstname lastname webservername  [ FORUSER(userid) ] '
```

Dieser Befehl erstellt exemplarisch einen Key Ring mit dem Namen *firstname*.lastname, welches *userid*.userid gehört. Sollte irgendein anderes Key Ring mit diesem Namen existieren, so wird der Existierende, durch den neuen Key Ring mit diesem Namen, ersetzt.

Wenn man einen Key Ring mittels DFH$RING erstellt, dann beschreibt das FORUSER Schlüsselwort die CICS Region User ID der CICS Region, in dem man gerade den Key Ring erstellt.
Der FORUSER(*userid*) Parameter ist optional. Sollte dieser ausgelassen werden, dann wird der Key Ring dem User, der die DFH$RING Clist ausführt, zugewiesen. Der DFH$RING Clist spezifiziert Initialisierungswerte für die folgenden internen Variablen, welche eventuell vor dem Benutzen benutzerdefiniert werden können:

`organization`, `department`, `city`, `state`, und `country`
werden benutzt, um die entsprechenden Felder des vollqualifizierten Namens (distinguished name, DN) der generieten Zertifikate zu definieren.

`certifier` wird für das Label eines selbst-signiertes CA Zertifikates angewendet, um die anderen Zertifikate zu signieren. Sollte ein Zertifikat mit diesem Label nicht bereits existieren, dann wird es erstellen, wenn DFH$RING das erste Mal ausgeführt wird.
Der Ring, der durch DFH$RING erstellt wird, beinhaltet Zertifikate mit folgenden Labels:

lastname-Web-Server

Das ist für den Gebrauch in dem CERTIFICATE Attribut des TCPIPSERVICEs mit Attribut PROTOCOL das auf http gesetzt, gedacht. Der vollqualifizierte Name (DN) innerhalb des Zertifikates enthält den „Common Name“ *webservername*, welcher derselbe sein sollte, wie der IP Name, der in Zusammenhang mit der Verbindung steht. Web Browser überprüfen gewöhnlich den „Common Name“ (CN) nach Übereinstimmungen mit dem IP Namen des Servers, von welchem man es erhalten hat.

lastname-Default-Certificate

Dieses Zertifikat wird als Standardzertifikat für den Key Ring markiert und dadurch wird es für alle TCPIPSERVICEs (und CORBASERVERs) benutzt, die nicht einen

CERTIFICATE Attribut benutzen. Dieses Zertifikat enthält ebenfalls ein Common Name des *webservername.*

Folgende Zertifikate werden benötigtet, um Client Zertifikate die erhalten wurden auf Gültigkeit zu überprüfen, und die von diesen angegebenen Zertifizierungsstellen signiert wurden:

- VeriSign Class 1 Primary CA,
- VeriSign Class 2 Primary CA,
- IBM® World Registry CA.

Zertifikate anderer Zertifizierungsstellen oder Zertifikate, die selbst ausgestellt wurden, müssen deren Zertifikate manuell dem Key Ring hinzugefügt werden, in dem der `RACDCERT CONNECT` Befehl benutzt wird. Wenn auf diese Art und Weise ein Zertifikat dem Key Ring hinzugefügt wird, muss USAGE(PERSONAL) spezifiziert werden. Wenn man USAGE(SITE) spezifiziert, dann kann das Zertifikat nicht in CICS benutzt werden.

Um einen RACF KEYRING in der RACF Datenbank zu erstellen, der für das Benutzen in der CICS Region geeignet sein soll, müssen folgende Rechte den entsprechenden Profilen in der FACILITY Klasse erteilt sein:

CONTROL

- IRR.DIGTCERT.GENCERT (ermöglicht es, dass Zertifikate von einem CERTAUTH Zertifikat signiert werden)
- IRR.DIGTCERT.ADD bei der ersten Ausführung (ermöglicht es, dass ein CERTAUTH Zertifikat generiert wird)
- IRR.DIGTCERT.CONNECT (um CERTAUTH Zertifikate für andere Benutzer benutzbar zu machen)

UPDATE

- IRR.DIGTCERT.CONNECT (um CERTAUTH Zertifikate in dem Key Ring benutzbar zu machen)
- IRR.DIGTCERT.* (um Zertifikate für andere Benutzer zu verwalten)

READ

IRR.DIGTCERT.* (um Zertifikate zu verwalten, die unter der eigenen User ID laufen)

Es besteht die Möglichkeit, das Informationen über Zertifikate der eigenen User ID hinzugefügt werden, wenn über READ Rechte für die IRR.DIGTCERT.ADD Profile in der FACILITY Klasse verfügt wird. Das Hinzufügen von Informationen über Zertifikate anderer Benutzer ist möglich, wenn über UPDATE Rechte für das IRR.DIGTCERT.ADD Profil in der FACILITY Klasse verfügt wird. Durch das Zugriffsrecht RACF SPECIAL, kann der Befehl RACDCERT ADD für jede User ID ausgeführt werden. Ebenso ist es mit diesem SPECIAL Zugriffsrecht möglich, ein digitales Zertifikat für jeden RACF definierten User, für jede Zertifizierungsstelle oder für jedes (Web)Site Zertifikat zu generieren.

IRR.DIGTCERT.* beinhaltet den Sternchen Platzhalter, und steht für einen unspezifiziertes (generic) Profil. Um unspezifizierte Profile in einer FACILITY Klasse zu ermöglichen, sollte zuerst folgendes durchgeführt werden:

```
SETROPTS GENERIC(FACILITY)
```

Anschließend kann wie folgt vorgegangen werden:

```
RDEFINE FACILITY(IRR.DIGTCERT.*)
RDEFINE FACILITY(IRR.DIGTCERT.ADD)
RDEFINE FACILITY(IRR.DIGTCERT.CONNECT)
RDEFINE FACILITY(IRR.DIGTCERT.GENCERT)
```

Abhängig davon, ob die FACILITY Klasse in RACLIST aufgelistet ist, sollte dann einer der weiteren Schritte durchgeführt werden:

```
SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS GENERIC(FACILITY) REFRESH
```

Um eine User ID oder group *ringuser* die Rechte für das Ausführen der Befehle, die sich in DFH$RING befinden, zu erteilen, sollte dermaßen vorgegangen werden:

```
PERMIT IRR.DIGTCERT.* CLASS(FACILITY) ID(ringuser)
ACCESS(READ)
```

```
PERMIT    IRR.DIGTCERT.CONNECT    CLASS(FACILITY)    ID(ringuser)
ACCESS(UPDATE)(für sich selbst)
```

```
PERMIT    IRR.DIGTCERT.CONNECT    CLASS(FACILITY)    ID(ringuser)
ACCESS(CONTROL)(für andere User)
```

```
PERMIT    IRR.DIGTCERT.GENCERT    CLASS(FACILITY)    ID(ringuser)
ACCESS(CONTROL)
```

Dem ersten User des DFH$RING, dem *certauser*, muss das Recht für das Erstellen eines Zertifizierungsstellen Zertifikat zugeteilt werden, welches dann benutzt wird, um alle anderen Zertifikate, die von DFH$RING erstellt worden sind, zu signieren. Dies erfordert:

```
PERMIT     IRR.DIGTCERT.ADD     CLASS(FACILITY)     ID(certauser)
ACCESS(CONTROL)
```

Anmerkung: Die obigen Zugriffsrechte sollte nur Usern zugeteilt werden, die CICS Systems administrieren, und nicht generell CICS User.

Nach dem Ausführen der RACDCERT Befehle, welche Zertifikate oder Key Ringe updaten, wenn die DIGTCERT und DIGRING Klassen in RACLIST aufgelistet sind, sollte man folgendes durchführen:

```
SETROPTS RACLIST(DIGTCERT DIGTRING) REFRESH
```

Anmerkung: Nachdem man ein Key Ring update durchgeführt hat, sollte die CICS Region neu gestartet werden, um die Veränderungen wirksam zu machen. Dies ist nötig, weil CICS System SSL Services benutzt, um die SSL Umgebung zu initialisieren und daher muss es SSL schließen und wieder initialisieren, um die Veränderungen wirksam zu machen.

5.3.2.1 RACF User ID in Verbindung mit einem Zertifikat bringen

Das Client Zertifikat kann für das Ermitteln der User ID benutzt werden, wenn das Zertifikat in Verbindung mit einer RACF User ID steht.
Um ein Zertifikat mit einer RACF User ID in Verbindung zu bringen, gibt es zwei Möglichkeiten:

- Benutzer können ihre Zertifikate online mittels eines Browers registrieren. Damit es Clients überhaupt ermöglicht wird ihre Zertifikate selbst zu registrieren, muss AUTHENTICATE(AUTOREGISTER) in der TCPIPSERVICE Definition gesetzt werden. Benutzer, die sich mit CICS über solch einen TCPIPSERVICE verbinden, müssen einen Client Zertifikat haben. Wenn dieses Zertifikat bereits in Verbindung mit einem Benutzer registriert ist, dann wird dieser Benutzer benutzt; wenn nicht, dann wird der Client aufgefordert sich mit einer User ID und einem Passwort, mittels der HTTP Basisauthentifizierung, zu identifizieren. Wenn der Client eine gültige User ID und Passwort eingibt, dann wird die User ID zu dem Zertifikat registriert und der Client wird nicht noch einmal aufgefordert ein Password einzugeben.
  Für detaillierte Informationen zu den Regeln wird auf [IBM07] auf den Seiten 236 ff verwiesen.

- Sobald ein Zertifikat so registriert wurde, kann es für alle einkommenden TCP/IP Verbindungen benutzt werden. Zum Beispiel könnte dasselbe Zertifikat sowohl Benutzer von IIOP Anfragen, als auch HTTP Anfragen authentifizieren und identifizieren.

- Benutzung des RACDCERT Befehls. Wenn nicht erwünscht ist, dass die Clients ihre eigenen Zertifikate registrieren, müssen diese Zertifikate mit dem RACDCERT Befehl registriert werden. Bevor man RACDCERT ausführt, muss jedoch das Zertifikat, das für eine weitere Verarbeitung benutzt werden soll, in eine von TSO aus erreichbare, sequentielle MVS Datei mit dem Format RECFM=VB, runtergeladen werden. Der Syntax von RACDCERT ist:

```
RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
```

  Dabei ist ‚datasetname' der Name des Data Sets, der das Client Zertifikat enthält und

‚userid' die User ID, die in Verbindung mit dem Zertifikat steht. Wenn der optionale ID (userid) Parameter ausgelassen wird, wird das Zertifikat mit dem User, der den RACDCERT Befehl abgesetzt hat, in Verbindung gebracht.

Das Hinzufügen von Informationen über das eigene Zertifikat ist dann möglich, wenn das READ Zugriffsrecht, für das `IRR.DIGTCERT.ADD` Profil in der FACILITY Klasse, zugeteilt wurde. Das Hinzufügen von Informationen über Zertifikate andere User IDs ist dann möglich, wenn das UPDATE Zugriffsrecht für das `IRR.DIGTCERT.ADD` Profil in der FACILITY Klasse zugeteilt wurde oder über die RACF SPECICAL Zugriffsreche verfügt werden. Für weitere Informationen über den RACDCERT Befehl, inklusive des erlaubten Datenformat des runtergeladenen Zertifikat im Dataset siehe [IBM10].

5.3.2.2 Ein Zertifikat als nicht vertrauenswürdig erachten

Sollte ein Zertifikat bereits in der RACF Datenbank registriert sein, dass jedoch nicht für den Gebrauch für Clients in Frage kommen soll, dann kann dieses Zertifikat als UNTRUSTED markiert werden, in dem man den RACDCERT Befehl benutzt. Um das zu tun, sollte zuerst

```
RACDCERT ID(userid) LIST
```

ausgeführt werden, um das Label zu finden, dass zu diesem, zu verändernden, Zertifikat gehört, und anschließend

```
RACDCERT ID (userid) ALTER(LABEL(label)) NOTRUST
```

um das Zertifikat als nicht vertrauenswürdig zu markieren. Clients werden dann daran gehindert mit diesem Zertifikat CLIENTAUTH Verbindungen aufzubauen.

5.3.3. Systeminitialisierungsparameter bezüglich SSL

Um SSL/TLS in einer CICS TS V3.2 Umgebung überhaupt verwenden zu können, müssen die Werte der folgenden Systeminitialisierungsparameter bestimmt werden:

- ✓ **CRLPROFILE**=*PROFILENAME*

  Bestimmt den Namen des Profils (246 Großbuchstaben), der benutzt wird um CICS für den Zugriff auf die Certificate Revocation List (CRLs) zu autorisieren, welche sich in einem LDAP Server befinden. CRLSs (Zertifikatsperrlisten) ermöglichen es festzustellen, ob ein Zertifikat gesperrt oder widerrufen wurde und warum.

  Wenn der CRLPROFILE Parameter ausgelassen oder unzulässig ist, oder das spezifizierte Profil unzulässige Daten enthält, oder, wenn der LDAP Server, der durch das Profil identifiziert wird, unerreichbar ist, dann überprüft CICS während des SSL Handshakes nicht den annullierten Status der Zertifikate.

- ✓ **ENCRYPTION**={*STRONG* | *WEAK* | *MEDIUM*}

  Bestimmt die Cipher Suite, die CICS für sichere TCP/IP Verbindungen benutzt. Wenn eine sichere Verbindung zwischen einem Service Requester und einem Service Provider aufgebaut wird, dann wird die sicherste Cipher Suite, die von beiden unterstützt wird, angewendet.

  - ➢ **ENCRYPTION**=*STRONG*:

    Wird dann eingesetzt, wenn das andere System dies benötigt. Man muss jedoch den Overhead in Kauf nehmen, der durch das Benutzen einer starken Verschlüsselung entsteht.

  - ➢ **ENCRYPTION**=*WEAK*:

    Wird eingesetzt, wenn die Länge der Schlüssel, die man bei der Verschlüsselung verwenden will, bis zu 40 bits lang sein sollen.

  - ➢ **ENCRYPTION**=MEDIUM:

    Wird handgehabt, wenn die Länge der Schlüssel, die man bei der Verschlüsselung benutzen will, bis zu 56 bits lang sein sollen.

Wenn in CICS die Transaktion namens CEDA verwendet wird , um einen TCPIPSERVICE oder einer URIMAP Ressource einzusetzen, dann initialisiert CICS automatisch die CIPHERS Attribute der Ressourcendefinition, die mit einer Standardliste der akzeptierten Cipher Suites versehen ist. Der Inhalt dieser Standardliste hängt von den Werten des ENCRYPTION Parameters ab.

- ✓ **KEYRING**=*keyring-Name*

  Bestimmt den Namen des Key Rings in der RACF Datenbank, die die Schlüssel und die Zertifikate enthält, die von CICS verwendet werden. Dieser muss der CICS Region User ID zugeordnet sein.

  Wenn CICS den KEYRING Parameter in der Systeminitialisierungstabelle findet, weiß es, dass eine Erstellung von SSL/TLS benötigt wird und erstellt daher eine offene Transaktionsumgebung (OTE) TCB, die man SP TCB nennt, welche für die eigene Socket PThread Tasks benutzt wird. Der SP TCB führt einen Pool von S8 TCPs, die für die Erstellung von SSL Verbindungen angewandt werden. Jede SSL Verbindung wendet einen S8 TCP an, welches von dem SSL Pool alloziiert wird und einen a UNIX® PThread benötigt. Alle S8 TCPs laufen unter einen einzigen LE Enklave, welche dem SP TCB zugeordnet ist und den SSL Cache enthält.

- ✓ **MAXSSLTCBS**={8 | *Anzahl*}

  Bestimmt die maximale Anzahl der S8 TCBs, die CICS zur Verfügung stehen, um SSL Verbindungen zu verarbeiten. Die S8 TCBs werden in dem SSL Pool erstellt und verwaltet.

  Nur für die Zeit, die S8 TCBs benötigen um eine SSL Verbindung aufzubauen, werden sie für Transaktionen gesperrt. Nach dem der SSL Verbindungsaufbau komplett ist, wird das TCB wieder in dem SSL Pool freigegeben, damit man es wiederverwenden kann.

  Eine Erhöhung der Anzahl von verfügbaren TCBs ermöglicht es mehrere simultane SSL Verbindungen herzustellen. Jedoch führt eine zu hohe Anzahl von TCBs dazu, dass es sich auf den Speicher auswirkt. Der maximale Wert, den man für MAXSSLTCBS angeben kann ist 1024.

- ✓ **SSLDELAY**={600 | *Anzahl*}

  Bestimmt die Dauer in Sekunden für die CICS eine Session ID für SSL Verbindungen im Cache aufbewahrt. Session IDs sind Token, die eine sichere Verbindung zwischen CICS und dem SSL Client darstellen. Die Session ID wird während dem SSL-Handshake zwischen CICS und dem SSL Client erstellt und ausgetauscht. Der Wert des SSLDELAYs ist eine Anzahl von Sekunden zwischen 0 bis 86400, wobei 600 der Standardwert ist. Wenn der Wert des SSLDELAY Parameters erhöht wird, bleibt die Session ID länger im Cache, und dadurch wird die Zeit, die benötigt wird, um SSL auszuhandeln (SSL Handshake), optimiert.

  Der SSLDELAY Parameter erscheint nur, wenn der SSLCACHE Parameter auf den Wert CICS gesetzt ist.

- ✓ **SSLCACHE**={*CICS* | *SYSPLEX*}

  Bestimmt, ob CICS den lokalen SSL Cache in der CICS Umgebung benutzen soll, oder den Cache mit verschiedenen CICS Umgebungen teilen soll, in dem ein Sysplex Session Cache eingesetzt, dass von System SSL zur Verfügung gestellt wird.

  Wenn **SSLCACHE**=*CICS* gesetzt wird, dann durchläuft ein Client, der sich erfolgreich mit der CICS Umgebung 1 auf dem z/OS System 1 verbinden konnte, und sich dann mit der CICS Umgebung 2 auf dem z/OS System 2 verbindet, einen vollen SSL Handshake in beiden Fällen. Dies ist möglich, weil CICS die SSL Session ID in einem Cache speichert, der sich im CICS Adressraum befindet.

  Wenn **SSLCACHE**= *SYSPLEX* gesetzt wird, dann kann eine SSL Sitzung, die in einer CICS Umgebung auf einem System des Sysplex hergestellt wurde, unter Verwendung einer CICS Umgebung auf einem anderen System im Sysplex fortgesetzt werden, solange der SSL Client die Session ID vorweisen kann, die er für die erste Session erhalten hat, während er die zweite Session initialisiert.

  CICS benutzt den Sysplex Session Cache Support, dass von dem System SSL gestartetem Task (GSKSRVR) (einer optionalen Komponente des System SSL) zur Verfügung gestellt wird.

GSKSRVR verarbeitet die folgenden Umgebungsvariablen:

- **GSK_LOCAL_THREADS**

  Diese Variable beschreibt die maximale Anzahl der Threads, welche benutzt werden, um die Anfragen auf Programmaufrufe zu bearbeiten, die von den SSL Applikationen gemacht wurden, welche wiederum auf demselben System laufen, wie der von GSKSRVR gestartete Task.

- **GSK_SIDCACHE_SIZE**

  Diese Variable beschreibt die Größe des Sysplex Session Cache in Megabyte.

- **GSK_SIDCACHE_TIMEOUT**

  Die Variable beschreibt den Sysplex Session Cache Enty timeout in Minuten.

Die gemeinsame Benutzung von Session IDs über verschiedene CICS Umgebungen hinweg, ist besonders dann praktisch, wenn Web Service Anfragen über eine bestimmte Anzahl von CICS Umgebung hinweg geroutet werden, und man dabei Techniken verwendet, die die Auslastung bei TCP/IP Verbindungen genutzt, wie den TCP/IP port sharing oder den Sysplex Distributor. Wenn der Cache gemeinsam über mehrere CICS Umgebungen hinweg genutzt wird, dann kann die Anzahl der vollständigen SSL Handshakes erheblich reduziert werden.

Um den Sysplex Session Cache benutzen zu können, muss jedes System im Sysplex denselben externen Sicherheitsmanager benutzen (zum Beispiel: RACF, den z/OS Security Server) und eine User ID auf einem System des Sysplex, muss auf allen Systemen des Sysplex für diese stehen. Das bedeutet, dass beispielsweise die User ID ABC auf dem System A, die gleichen Rechte wie der User mit der ID ABC auf dem System B besitzt. Der externe Sicherheitsmanager muss die Funktion RACROUTE REQUEST=EXTRACT, TYPE=ENVRXTR und RACROUTE REQUEST=FASTAUTH unterstützen.

Caching über einem Sysplex hinweg, kann nur dann funktionieren, wenn Regions SSL Verbindungen über dieselbe IP Adresse aufbauen.

In *Kapitel 5 – Realisierung* wird demonstriert, wie die CICS Umgebung für die Unterstützung von SSL Verbindung von einem Web Service Client, der in einem WebSphere Application Server läuft, aktiviert wird.

5.4 Definition einer TCPIPSERVICE Ressource für SSL

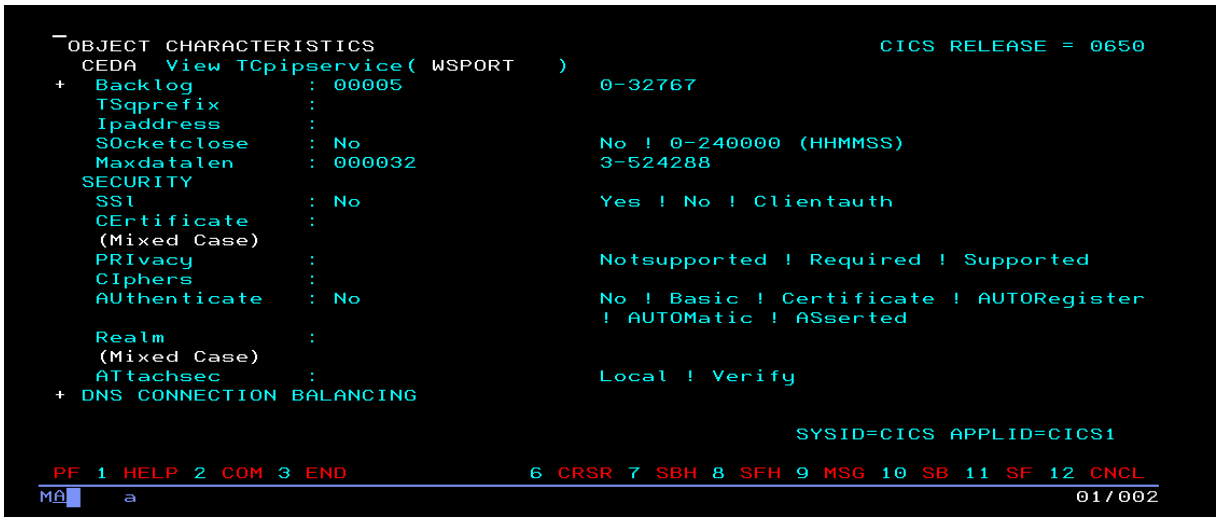

Abbildung 4.1: Ausschnitt aus einer TCPIPSERVICE Ressource

Die folgenden Attribute der TCPIPSERVICE Ressourcen Definition beziehen sich auf das Benutzen von SSL für einkommende Web Service Anfragen, wenn man HTTP für den Transport benutzt.

AUTHENTICATE(*NO* | *BASIC* | *CERTIFICATE* | *AUTOREGISTER* |*AUTOMATIC*)

Beschreibt das Schema, welches für die Authentifizierung benutzt werden soll.

- NO

  Der Client braucht nicht die Informationen für die Authentifizierung und Identifizierung zu senden. Wenn der Client jedoch ein gültiges Zertifikat schickt, dass

schon im Security Manager registriert ist, und diese mit einer User ID assoziiert ist, dann identifiziert die User ID den Client.

- BASIC

  HTTP Basic Authentifizierung wird angewendet, um eine User ID und ein Password von einem Client zu erhalten.

- CERTIFICATE

  SSL Client Zertifikat Authentifizierung wird benutzt, um den Client zu authentifizieren und zu identifizieren. Der Client muss ein gültiges Zertifikat schicken, dass bereits in RACF registriert ist und mit einer User ID in Verbindung steht. Wenn kein gültiges Zertifikat empfangen wird, oder wenn das Zertifikat nicht mit einer User ID assoziiert ist, dann wird die Verbindung zurückgewiesen.

  Wenn sich in User erfolgreich authentifiziert hat, dann identifiziert die User ID, die mit dem Zertifikat assoziiert ist, den Client.

  Hinweis: Wenn AUTHENTICATE(CERTIFICATE) oder AUTHENTICATE(AUTOREGISTER) spezifiziert wird, dann muss SSL(CLIENTAUTH) spezifiziert werden.

- AUTOREGISTER

  SSL Client Zertifikat Authentifizierung wird benutzt, um den Client zu authentifizieren und zu identifizieren.

  - Wenn der Client ein gültiges Zertifikat sendet, das *nicht* im Sicherheitsmanager registriert ist, dann wird die HTTP Basis Authentifizierung benutzt, um die User ID und das Password des Users zu erhalten. Wenn das Password gültig ist, dann registriert CICS das Zertifikat mit dem Sicherheitsmanager und assoziiert es mit der User ID. Die User ID identifiziert den Client.

- Wenn der Client ein gültiges Zertifikat versendet, dass *bereits* im Sicherheitsmanager registriert ist, und welche mit einer User ID assoziiert ist, dann identifiziert dieses User ID den Client.

➢ AUTOMATIC

Dieser Wert vereinigt die Funktionen von AUTOREGISTER und BASIC.

Wenn der Client ein Zertifikat schickt, dass bereits beim Sicherheitsmanager registriert und mit einer User ID assoziiert ist, dann identifiziert die User ID den Client.

- Wenn der Client ein Zertifikat schickt, dass nicht im Sicherheitsmanager registriert ist, dann wird HTTP Basis Authentifizierung benutzt, um eine User ID und Password vom Client zu erhalten. Vorausgesetzt, dass das Passwort gültig ist, registriert CICS das Zertifikat mit dem Sicherheitsmanager und assoziiert es mit der User ID. Die User ID identifiziert den Client.

- Wenn der Client kein Zertifikat verschickt, dann wird die HTTP Authentifizierung benutzt, um die User ID und das Password von dem Benutzer zu erhalten. Wenn der Endbenutzer erfolgreich authentifiziert wurde, dann identifiziert die bereitgestellte User ID den Client.

CERTIFICATE

Beschreibt das Label eines X.509 Zertifikat, dass während eines SSL Handshakes als Server Zertifikat benutzt wird. Wenn dessen Attribute weggelassen werden, wird das Standardzertifikat, das im Key Ring der CICS Region User ID definiert wurde, angewendet.

CIPHERS

Spezifiziert einen bis zu 56 Hexadezimal Zeichen langen String, der als Liste, die aus bis zu 28 2-Zeichen langen Cipher Suite Zahlen besteht, interpretiert wird.

Wenn CEDA benutzt wird, um die Ressourcen zu definieren, dann initialisiert CICS automatisch diese Attribute mit einer Standardliste (aus Zahlen bestehend), die akzeptiert werden. Der Inhalt der Liste hängt von dem Grad der Verschlüsselung ab, welches von dem ENCRYPTION Parameter der Systeminitialisierung abhängt.

- Für ENCRYPTION=WEAK, beträgt der Standardwert 03060102.
- Für ENCRYPTION=MEDIUM, beträgt der Standardwert 0903060102.
- Für ENCRYPTION=STRONG, beträgt der Standardwert 050435363738392F303132330A1613100D0915120F0C03060201.

Man kann die Standardliste der Cipher Suite so einstellen, dass der Grad der Verschlüsselung der bei Aushandlung des SSL/TLS Handshake benutzt wird, minimiert oder auch maximiert wird,.

PORTNUMBER(*port***)**

Beschreibt die Nummer des Ports, auf welchen CICS auf einkommende HTTPS Anfragen lauscht.

SSL (*NO* | *YES* | *CLIENTAUTH*)

Beschreibt, ob der TCPIPSERVICE für die Verschlüsselung und Authentifizierung benutzt werden soll.

- *NO*

  SSL wird nicht benutzt.
- *YES*

  Eine SSL Sitzung wird verwendet. CICS wird ein Server Zertifikat an den Client schicken.
- *CLIENTAUTH*

  Eine SSL Sitzung wird angewendet. CICS wird ein Server Zertifikat zum Client schicken und der Client muss ein Client Zertifikat zum CICS versenden.

5.5. SSL Optimierung

Die Implementierung von SSL führt zu einer zunehmenden CPU Auslastung. Man sollte SSL nur für Anwendungen benutzen, die dieses Maß an Sicherheit benötigen. Für diese Anwendungen sollte man folgende Techniken in Betracht ziehen, um die Performance für SSL zu optimieren.

- Das Verwenden von kryptographische Hardware

- Den Wert der CICS System Parameter SSLDELAY der Systeminitialisierungstabelle erhöhen, so dass die Session IDs länger im SSLCACHE verweilen, was sich auf einen Teil des SSL Handshakes auswirkt.
- Den Wert des CICS System Parameters MAXSSLTCBS der Systeminitialisierungstabelle erhöhen, so dass es mehr S8 TCBs in dem SSL Pool für die SSL Handshake Verhandlungen gibt.
- Den CICS SSLCACHE Parameter der Systeminitialisierungstabelle benutzen, um SSL Cache querüber ein Sysplex zu implementieren, wenn Web Service Anfragen über eine Anzahl von CICS Umgebungen geroutet werden. Wenn jedoch CICS als einzelne Umgebung benutzt wird, dann sollte
- SSLCACHE(CICS) im Gegensatz zu SSLCACHE(SYSPLEX) spezifiziert werden, um den zusätzlichen Aufwandskosten zu vermeiden, die entstehen, wenn man die SSL Session ID gemeinsam nutzbar machen will.

Durch das Setzen von SOCKETCLOSE NO in der TCPIPSERVICE Definition hält man den Socket offen. Dies ist auch der Standard für dauerhafte HTTP 1.1 Sitzungen und machen das Durchführen eines SSL Handshake bei der zweiten Anfrage oder folgenden HTTP Anfragen somit unnötig.

Man sollte nur dann die Client Authentifizierung durch das Setzen von SSL(CLIENTAUTH) in der TCPIPSERVICE Definition anwenden, wenn es wirklich notwendig ist, dass sich die Clients mittels eines Client Zertifikates identifizieren sollen.

Client Authentifizierung benötigt während eines SSL Handshakes viel mehr Netzwerkübertragungen und mehr Verarbeitungsschritte seitens CICS, um das empfangene Zertifikat zu verarbeiten.

Kapitel 6 - Realisierung

In diesem Abschnitt wird ein Szenario dargelegt, anhand dessen demonstriert wird, wie die in den vorrangegangen Kapitel erläuterten Sicherheitsaspekten umgesetzt werden können. Zuerst wird die Ausgangssituation (Aufgabenstellung) aufgezeigt und anhand dieser wird erläutert, wie die einzelnen Sicherheitseinrichtungen implementiert werden können.

6.1 Erste Schritte

Auf dem universitätseigenen IBM z9 Business Server Model S07 BC befinden sich 3 LPARs (Logical Partitions). Auf einem der LPARs läuft das Betriebssystem z/OS in der Version 1.8. Einen Link für weitere Informationen rund um den Rechner findet man im Literaturverzeichnis unter [UniTue].

Unter der zahlreichen Software, die auf diesem Mainframe installiert ist, wird in dieser Diplomarbeit vor allem auf folgende Software eingegangen:

- CICS Transaction Server v.3.1
- WebSphere Application Server 6.1
- RACF

Software Update

Zu Beginn diese Diplomarbeit lagen diese Software in den eben aufgezählten Versionen vor. Da jedoch für die Realisierung von Sicherheit in Zusammenhang mit Web Services weitere Features benötigt werden, wurden der CICS Transaction Server und der WebSphere Application Server mit freundlicher Unterstützung von Frau Arnold, Frau Schmidt und Herrn Denneler, aus dem IBM Labor Böblingen, auf folgende Versionen aktualisiert:

- CICS Transaction Server v3.2 (enthält aktuelle Fixes)
- WebSphere Application Server v7.0 (incl. Web Service Feature Packs).

Der Web Service Provider:

Frau Arnold erstellte ein in Cobol geschriebenes Programm. Dieses benötigt als Eingabe zwei Strings (Name, Vorname) und gibt als Ausgabe „Hallo *VORNAME*“ + „*Länge der Strings*“

zurück. Mit diesem Programm wurden dann die nötigen Schritte durchgeführt, um diese Applikation in CICS als Web Service Provider zur Verfügung zu stellen. Ein Link zu einer Video-Demo, die veranschaulicht wie aus einer Cobol Anwendung ein Web Service mittels der Software WebSphere Developer for System z erstellt werden kann, kann im Literaturverzeichnis unter [IBM08] und [IBM09] gefunden werden.

Dieses (Cobol) Programm wurde bewusst einfach gehalten, weil der Schwerpunkt dieser Diplomarbeit nicht auf die Implementierung von komplexen Web Services liegen soll, sondern auf die Untersuchung der Umsetzungsmöglichkeiten von möglichen Sicherheitseinrichtungen.
Durch den Vorgang, eine Cobol-Anwendung als Web Service Provider in CICS zur Verfügung zu stellen, sind einige Dateien und Ressourcen generiert worden, von denen für das weitere Vorgehen der Diplomarbeit folgende von *Wichtigkeit* sind:

Dateien:

- getNameLength.wsbind
- getNameLength.wsdl

CICS Ressourcen:

- TCPIPSERVICE(WSPORT)
- PIPELINE(SECPIPRO)

Serviceadresse (Service endpoint):

- http://134.2.205.54:3602/secure/getNameLength

Der Java Test Client

Frau Schmidt erstellte auf dem WebSphere Application Server v7.0 auf dem Mainframe in einem EAR Project, ein Java Test Client, dass als Java Server Page über einen Browser zur Verfügung steht.

Adresse des Test Clients:
http://134.2.205.54:9527/nameLengthClient/jsps/TestClient.jsp

Im Laufe der Diplomarbeit kamen aus verschiedenen Gründen folgende Software zum Einsatz:

- IBM WebSphere Application Client v6.1
- IBM WebSphere Application Server Toolkit v6.1.1
- IBM WebSphere Application Server v6.1.1 mit Feature Pack for Web Services
- IBM WebSphere Application Server v7
- IBM Rational Application Developer v7
- IBM Rational Developer for System z 7.5.1
- Eclipse Plugin: XML Security

Selbstverständlich werden nicht alle der aufgezählten Software benötigt, um Sicherheitsaspekte für CICS Web Services zu implementieren. Diese Software wurden mehr oder weniger zu Forschungszwecken installiert, um die Unterschiede und Erneuerungen zu testen und vergleichen zu können.

6.2 Basis Sicherheitskonfiguration

Ausgehend von einer CICS Region mit keinerlei Sicherheit werden im Folgenden zunächst die Konfigurationen erläutert, um Transaktionssicherheit zu gewährleisten. Es werden dabei nur bestimmte Typen der Sicherheit für CICS implementiert, wie zum Beispiel die Sicherheit der Ressourcen oder die der Kommandos.

Für umfangreichere Informationen rund um CICS wird dabei auf die Quelle [IBM07] verwiesen.

6.2.1 Weitere SIT Parameters

Folgende SIT Parameter werden wie folgt bestimmt:

- SEC=YES
- SECPRFX=YES
- XTRAN=YES
- XUSER=YES

Durch SEC=YES wird bestimmt, dass die Dienste von RACF die Zugriffe zu den CICS Ressourcen kontrollieren. Desweiteren wurde das „Security Prefixing“ (SECPRFX=YES) für die zu benutzende CICS Region gewählt, was dazu führt, dass das gewählte RACF Sicherheitsprofil nicht andere CICS Regions beeinflusst. Somit kann jede CICS Region ihr individuelles Sicherheitsprofil haben. (Jedoch bedeutet dies auch einen größeren Aufwand für den Sicherheitsadministrator, weil er mehrere Profile definieren muss.)

Durch XTRAN=YES wird es CICS ermöglicht zu kontrollieren wer Transaktionen starten kann und schließlich spezifiziert XUSER=YES, dass CICS eine Überprüfung der Surrogate User durchführt.

6.2.2 CA Zertifikate erstellen

In diesem Unterabschnitt wird auf die Erstellung der CA Zertifikate eingegangen, die in den folgenden Szenarien gebraucht werden. Die Erstellung des Client und Server Zertifikat wird im Szenario selbst erläutert. Hier wird zuerst nur auf die CA Zertifikate eingegangen.

In welcher Beziehung die Zertifikate zueinander stehen, soll folgende Abbildung verdeutlichen.

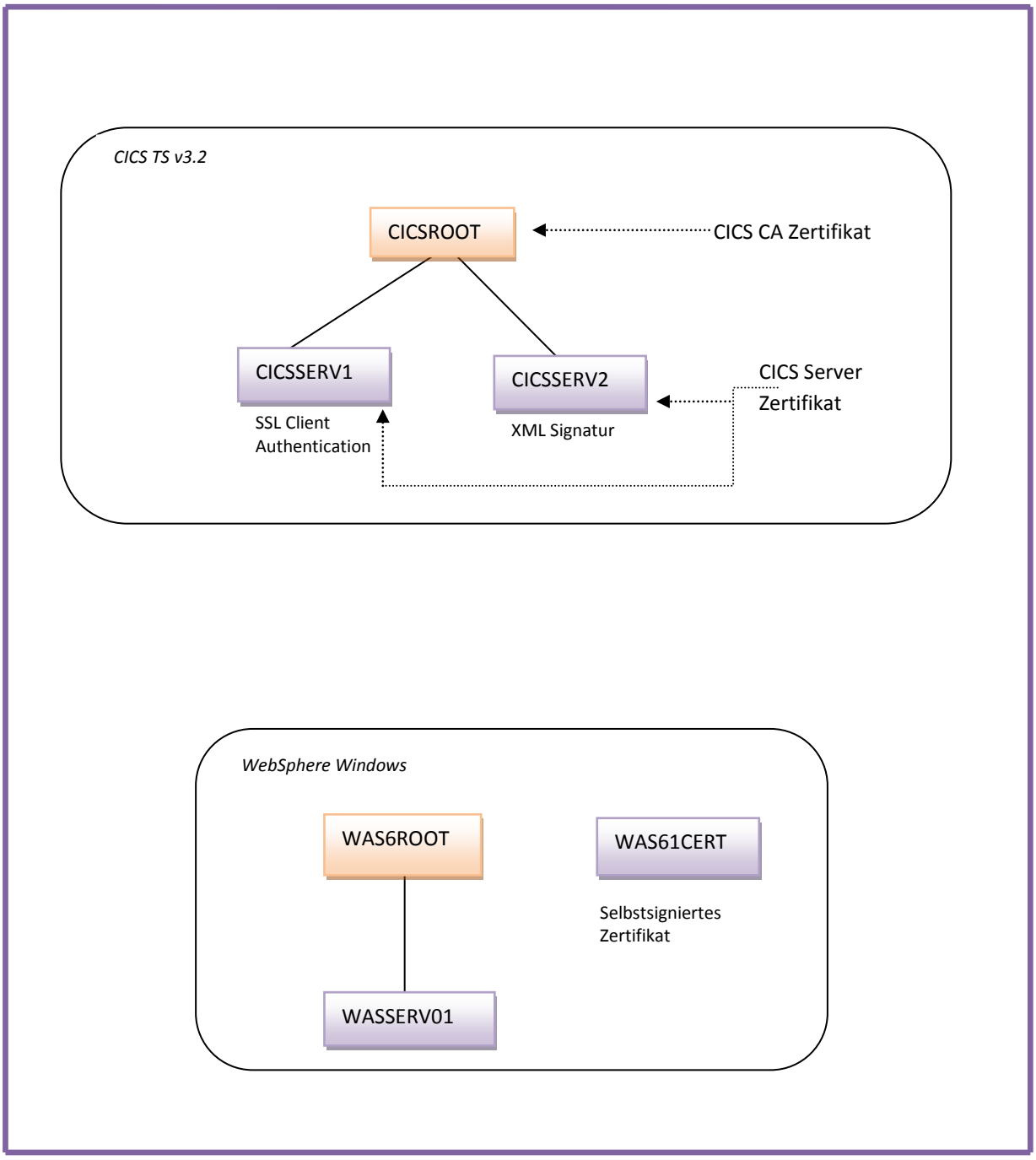

Abbildung 6.1: Zertifikate, die in den Szenarien benutzt werden

Um dieses CICS CA Zertifikat in RACF auszustellen wird der `RACDCERT` Befehl benutzt:

Spalte 80

```
RACDCERT CERTAUTH GENCERT                                          +
SUBJECTSDN(CN('CIWSROOT') T('CICSSERV-Zertifikat’)                 +
OU('CICSSERV-Zertifikat')                                          +
O('IT')                                                            +
L('BW')                                                            +
SP(‘Tuebingen’)                                                    +
C('DE'))                                                           +
WITHLABEL('CICSROOT')                                              +
NOTAFTER(DATE(2010/12/31))                                         +
SIZE(1024)                                                         +
ICSF
```

Abbildung 6.2: RACDCERT Befehl zum Erstellen des CICS CA Zertifikates

Nachdem dieses Zertifikat erstellt wurde, muss es noch in ein Data Set exportiert werden und dann mittels dem `OPUT` Befehl in eine HFS kopiert werden.

```
RACDCERT CERTAUTH EXPORT(LABEL('CICSROOT'))                  +
DSN('CICS.CICSROOT.DER') FORMAT(CERTDER)
OPUT 'CICS.CIWSROOT.DER' '/CICS/CERTIFICATES/CICSROOT.CER' +
BINARY CONVERT(NO)
```

Abbildung 6.3: Exportieren und Kopieren der Zertifikate in eine HFS Datei

6.3 Umsetzung – SSL Aktivieren [MODELL 1]

In Kapitel 4 wurde bereits die Vorgehensweise vorgestellt, wie SSL in CICS nutzbar gemacht werden kann. Im Folgenden wird eine mögliche Umsetzung demonstriert.

Anmerkung:
Es gibt verschiedene Möglichkeiten, wie ein Client den Web Service Provider (CICS) aufrufen kann. Als sehr praktisch haben sich dabei die von IBM WebSphere entwickelten Produkte gezeigt. Daher wird als Client mit der Entwicklungsumgebung Websphere Application Server Toolkit auf der Laufzeitumgebung des WebSphere Application Server 6.1 gearbeitet. Eine weitere Möglichkeit, wie ein Web Service Client für die Kommunikation mit dem Service Provider nutzen kann wäre zum Beispiel Apache Axis, was jedoch in dieser Diplomarbeit nicht weiter untersucht wurde, da dies sonst den Rahmen dieser sprengen würde.

Auch eine Lösung, in der die Verschlüsslungen und Entschlüsselungen für die Kommunikation mit dem Service Provider mittels XML (-Security Plugin für Eclipse) ist denkbar. Davon wird aber dringend abgeraten, da zum einen der Aufwand sehr hoch wird und zum anderen dieser noch sehr kompliziert werden kann, wenn noch eine Logik implementiert werden soll, die die Anfragen und Antworten verarbeitet.

6.3.1 SSL Szenario - Übersicht

In den folgenden Unterabschnitt werden die Maßnahmen erläutert, die vorgenommen werden müssen, um das folgende Szenario zu realisieren:

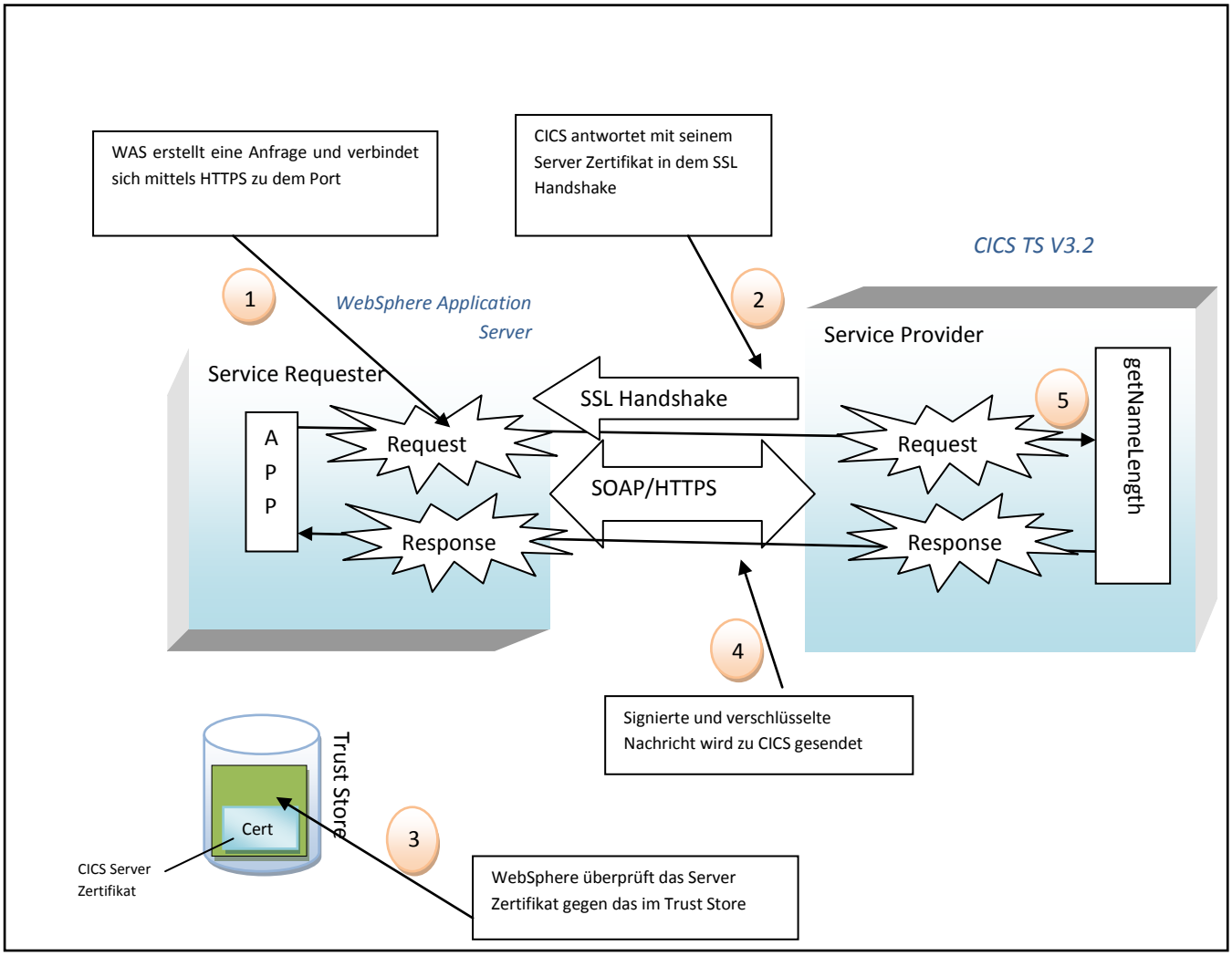

Abbildung 6.4: SSL Szenario

Erläuterung:

1.) Der WAS (WebSphere Application Server) erstellt eine Anfrage und initiiert eine Verbindung zu CICS über HTTPS.

2.) CICS antwortet mit seinem Server Zertifikat. Das ist entweder das Standard Zertifikat des Key Rings der CICS Region, oder das Zertifikat als solches, das im TCPIPSERVICE spezifiziert wurde. Dies ist der Teil des SSL Handshakes, in der die Verschlüsselungsparameter vereinbart werden.

3.) WebSphere empfängt das CICS Server Zertifikat und überprüft, ob die Einzelheiten des Zertifikats mit dem der Quelle übereinstimmen. Als Beispiel wird überprüft, ob

der Hostname übereinstimmt. Dann sucht es in seinen Trust Store nach dem CICS CA Zertifikat. Findet es eine Übereinstimmung, dann wird dem CICS Server vertraut.

4.) Die SOAP Nachricht wird über HTTPS gesendet. Die Verschlüsselung und die Signatur werden auf Transport Level gemacht.

5.) Die Nachricht kommt in CICS an und von dort an wird es von CICS verarbeitet (wie es in Kapitel 2.4.1 beschrieben wurde).

6.3.2 Zertifikate und Key Ringe erstellen

Generierung des Zertifikates und des Key Rings

Zunächst werden ein Zertifikat und ein Schlüsselpaar in RACF mittels RACDCERT generiert:

In der folgenden Abbildung 5.5 ist:

- die CICS Region ID = „CICSRI“,
- das CICS Server Zertifikat (in der Datei) = „CICSSERV1.p12“,
- das CICS CA Zertifikat = „CICSROOT.CER".

```
RACDCERT ID(CICSRI) GENCERT
     SUBJECTSDN(CN('CICSSERV1')
          T('CICSSERV-Zertifikat’)
          OU('Universitaet')
          O('IT’)
          L('BW')
          SP(‘Tuebingen’)
          C('DE'))
WITHLABEL('CICSSERV1')
SIGNWITH(CERTAUTH LABEL('CICSROOT'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
NOTAFTER(DATE(2010/09/30))
SIZE(1024)
PASSWORD('PASSWORD')
```

Abbildung 6.5: RACDCERT – Generierung eines Zertifikates und Public-Key Schlüsselpaares

Erläuterung:

1.) Der `RACDCERT GENCERT` Befehlt erstellt ein Zertifikat und ein Public-Key Schlüsselpaar.
2.) `ID` spezifiziert, dass die CICS Region User ID „CICSRI“ in Verbindung mit dem Zertifikat gebracht werden soll.
3.) `SUBJECTDSN` spezifiziert den DN (distinguished name) des X.509 Zertifikates.
4.) `WITHLABEL` spezifiziert den Namen des Labels, das zum Zertifikat gehört
5.) `SIGNWITH` spezifiziert das Zertifikat mit einem privaten Schlüssel, mit dem das Zertifikat unterzeichnet wird. Das CICS Zertifikat wird mit dem CICS CA Zertifikat signiert, das bereits zuvor ausgestellt wurde.
6.) `KEYUSAGE` spezifiziert wie die Schlüssel, die zum Zertifikat gehören, benutzt werden. Im obigen Beispiel wurde folgendes angewendet:
    - `HANDSHAKE,` weil das Zertifikat für SSL Handshakes verwendet wird
    - `DATAENCRYPT`, weil das Zertifikat für Verschlüsselung benutzt wird
    - `DOCSIGN`, weil das Zertifikat zum Signieren benutzt wird.
7.) `NOTAFTER` spezifiziert das Datum, ab dem das Zertifikat nicht mehr gültig ist.
8.) `SIZE` spezifiziert die Größe des Privaten Schlüssel, ausgedrückt in Dezimal Bits. In diesem Beispiel wurde die Schlüssellänge von 1024 spezifiziert.

Verbindung zum RACF Key Ring

Anschließend wird das Zertifikat mit dem RACF Key Ring verbunden:

```
RACDCERT ID(CICSRI) CONNECT(ID(CICSRI) LABEL('CICSSERV1')
RING(Cics.Cicsserv1))
```

```
RACDCERT ID(CICSRI) LIST
```

Mit dem letzteren Befehl würde dieses Zertifikat angezeigt werden.

6.4 Konfiguration des WebSphere Application Server für SSL Verarbeitung

Im folgenden Abschnitt wird erläutert wie der WebSphere Application Server für serverseitige SSL Verarbeitung konfiguriert wird.

6.4.1 CICS CA Zertifikate dem Trust Store hinzufügen

Um WebSphere für serverseitige SSL Verarbeitung zu konfigurieren, muss zunächst das CICS CA Zertifikat von dem Host heruntergeladen werden und dieses dann dem Trust Store hinzugefügt werden, in dem die WebSphere Administration Console benutzt wird.

Dies kann zum einen über FTP geschehen oder ganz bequem mit der Drag-And-Drop Funktion des Rational Developer for System z.
Nachdem dieses Zertifikat heruntergeladen wurde, muss dieses nun dem WebSphere Trust Store hinzugefügt werden. WebSphere benutzt dieses CA Zertifikat, um das CICS Zertifikat zu überprüfen, das für die Signierung der Response Nachrichten verwendet wurde.

Hierzu wird das Standard WebSphere Trust Store namens „NodeDefaultTrustStore“ verwendet. Das Standard Trust Store verweist auf die Datei trust.p12 im Knoten oder Zellenverzeichnis (Cell directory) des Konfigurationsrepository. Diese Datei ist vom Typ PKCS12.

Im Folgenden wird angenommen, dass das CICS CA Zertifikat in dem Verzeichnis: C:\CICSROOT.CER abgelegt worden ist.

Dieses wird dem Trust Store mittels der Administrationskonsole hinzugefügt. Hierzu werden folgende Schritte ausgeführt:

1.) Wähle **Security → „Verwaltung von SSL-Zertifikaten und Schlüsseln“**
2.) Unter „**Zugehörige Elemente**“ wird „**Keystores und Zertifikate**“ auswählen
3.) Das sich daraufhin öffnende Panel zeigt die Key Stores und Trust Stores an. Von den angezeigten wird nun auf „**NodeDefaultTrustStore**“ geklickt.
4.) Unter „**Weitere Merkmale**“ wird auf „**Unterzeichnerzertifikate“** geklickt und im sich öffnenden Panel „Hinzufügen“ ausgewählt. Dort werden dann, wie in Abbildung 5.6 dargestellt, folgende Werte eingetragen:

Alias: `CICS CA Zertifikat`

Dateiname: `C:\CICSROOT.CER`

Datentyp: `Binary DER-Daten`

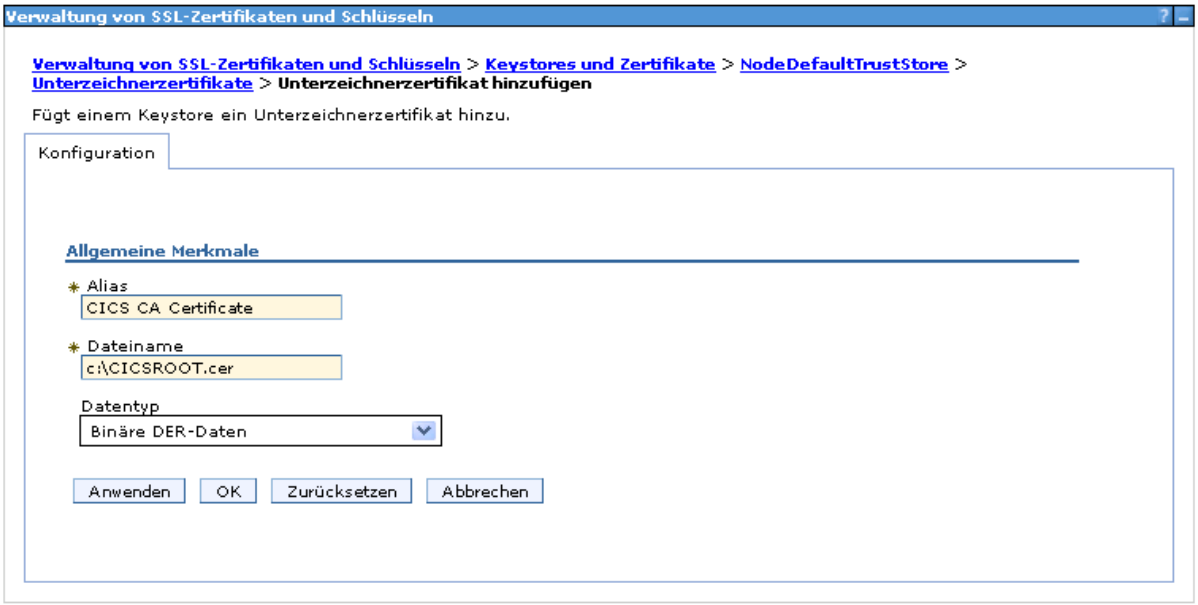

Abbildung 6.6: Hinzufügen des CICS CA Zertifikates zum Trust Store

5.) Anschließend wird mit OK bestätigt. Wie in Abbildung 5.7 dargestellt wird dann das Panel mit den Unterzeichnerzertifikate des Trust Stores angezeigt.

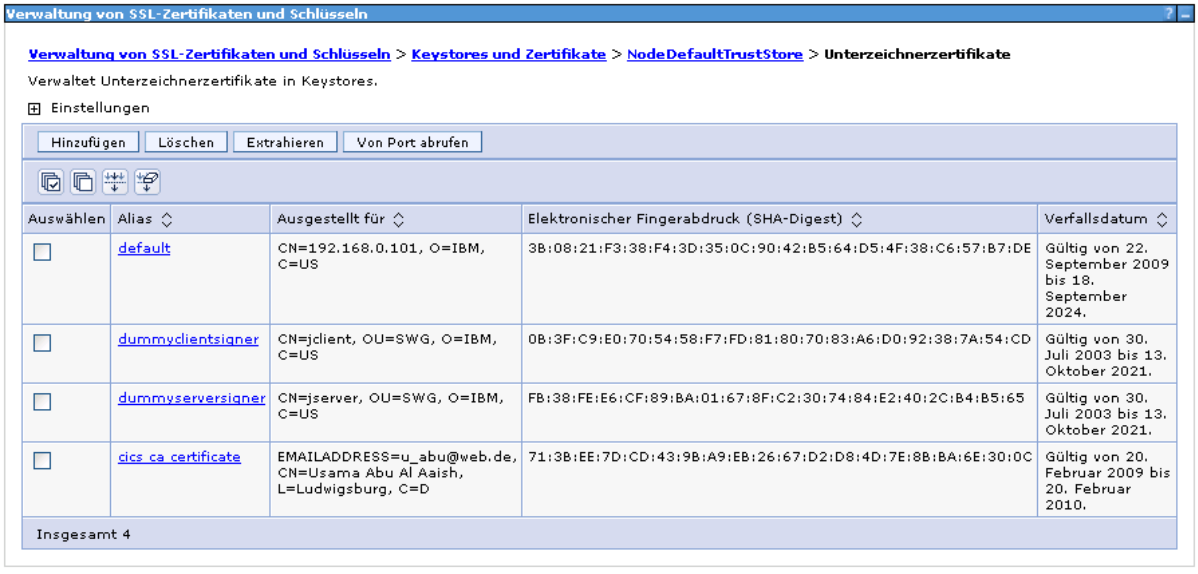

Abbildung 6.7: Unterzeichnerzertifikat im Trust Store

6.4.2 SSL Konfiguration des WebSphere Application Server

WebSphere Application Server wird mit einer Standard SSL Einstellung ausgeliefert. Diese Einstellungen sind mittels der Admin Konsole konfigurierbar.

Im Folgenden wird die Standard WebSphere „NodeDefaultSSLSettings“ SSL Konfiguration benutzt. Hierzu sind diese Schritte notwendig:

1.) Menüpunkt **Sicherheit→ Verwaltung von SSL-Zertifikaten und Schlüsseln**

2.) Unter **Zugehörige Merkmale** wird **SSL-Konfiguration** ausgewählt

3.) Es erscheint dann das Panel, das in Abbildung 5.8 dargestellt ist.

Abbildung 6.8: SSL Konfiguration in der WebSphere AppServer Admin Console

4.) Die folgende Abbildung 5.9 zeigt die Einstellungen für die Standard SSL Konfiguration

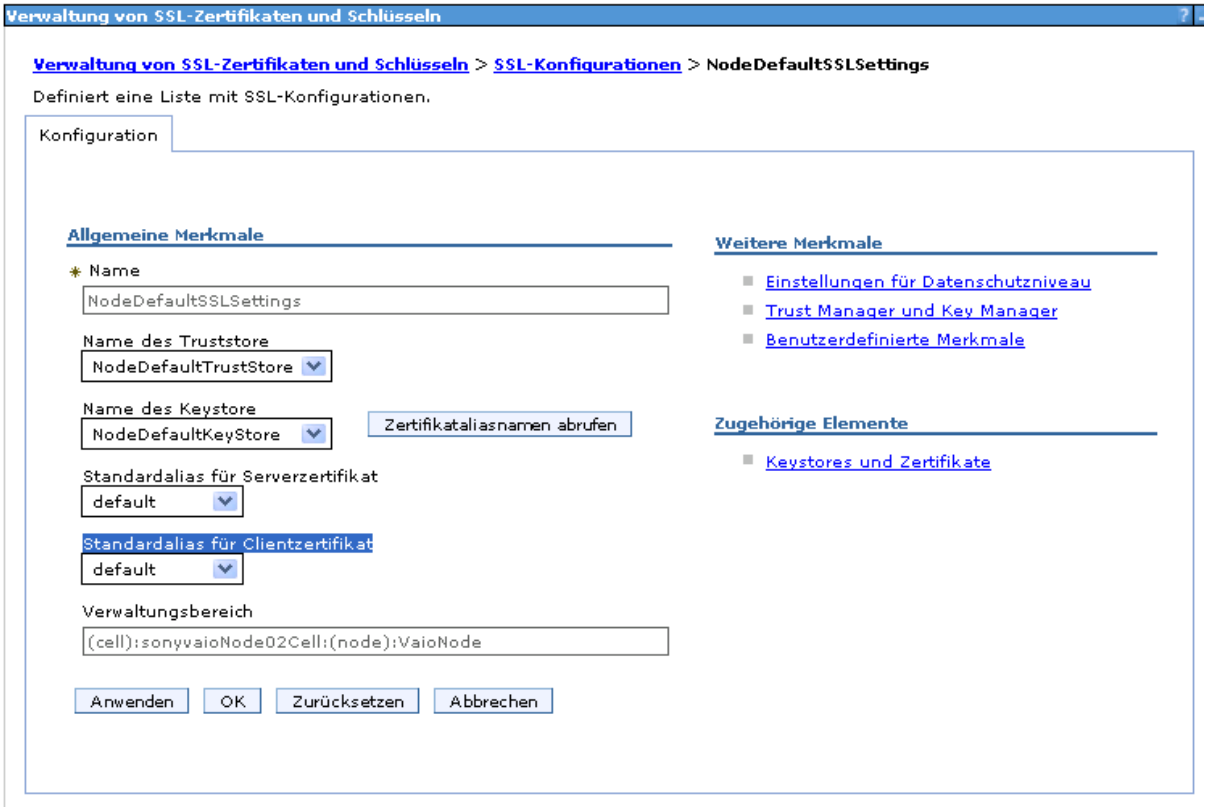

Abbildung 6.9: Standard SSL Einstellung

Folgende Werte werden für die Standard Einstellung gesetzt:

- **Trust store name** ist NodeDefaultTrustStore.
- **Key store name** ist NodeDefaultKeyStore.
- **Default server certificate alias** ist default.
- **Default client certificate alias** ist default.
- **Verwaltungsbereich** ist (cell):sonyvaioNode02Cell:(node):VaioNode . (Dadurch wird eindeutig diese Instanz des WebSphere Application Servers identifiziert)

5.) Unter weitere Merkmale wird nun „**Einstellungen für Datenschutzniveau**“ angeklickt. Das Resultat wird in folgender Abbildung 5.10 dargestellt.

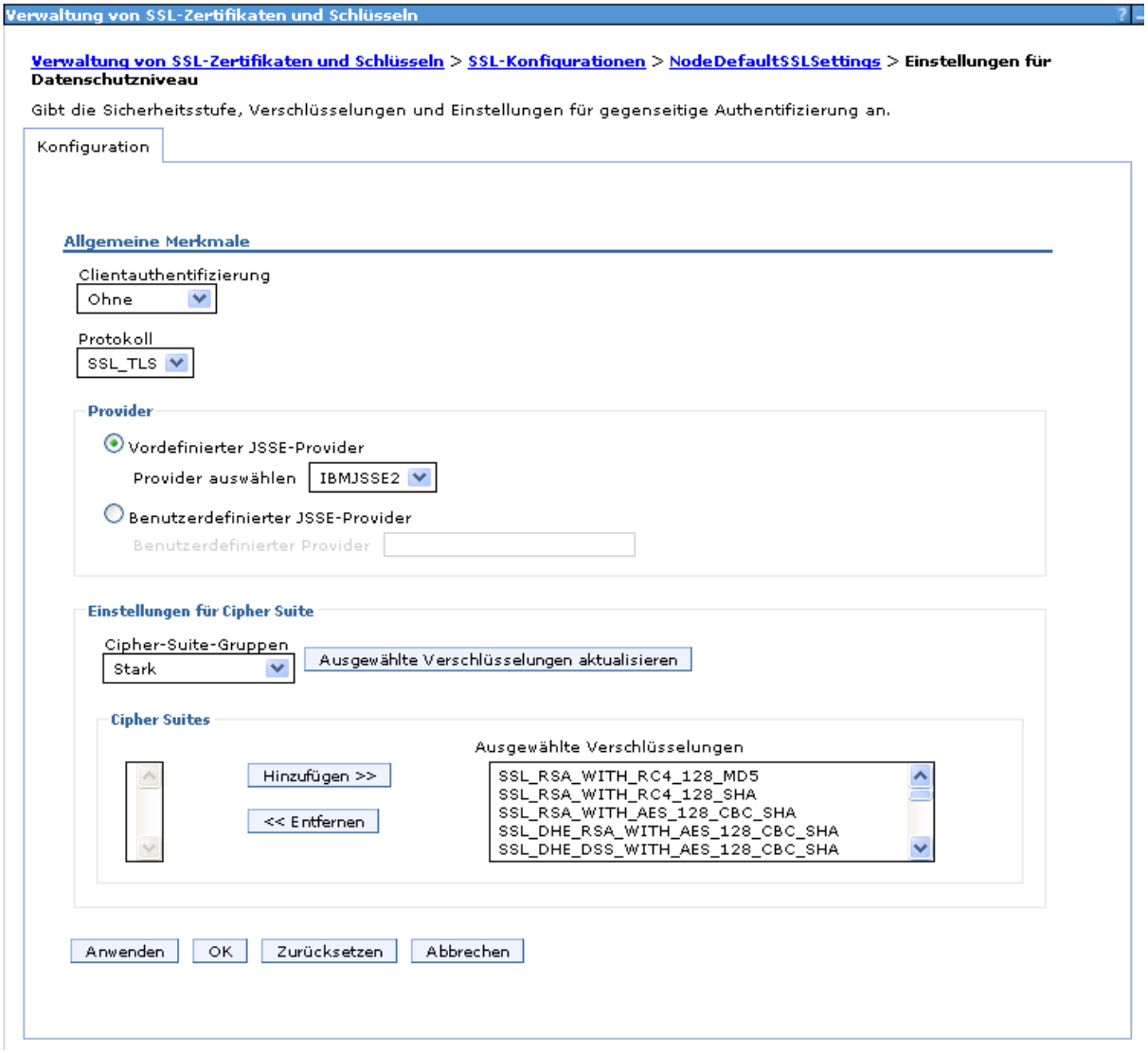

Abbildung 6.10: Einstellungen der Quality of Protection (QoP) in den SSL Konfigurationen

Die in Abbildung 6.10 dargestellten Felder haben folgende Bedeutung:

- **Client Authentifizierung**: Wird gebraucht, wenn WebSphere als SSL Server fungiert. Mögliche Einstellungen sind „None“, „Supported“ oder „Required“ und beziehen sich darauf, ob ein Client Zertifikat benötigt wird, um mit WebSphere zu kommunizieren.
- **Protocol**: Beschreibt das SSL Handshake Protokoll. Standardeinstellung ist „SSL_TLS“, was alle Handshake Protokolle außer SSLv2 ermöglicht. Weitere mögliche Einstellungen, die in WebSphere gemacht werden können sind:
  – SSL
  – SSLv2
  – SSLv3
  – TLS

– SSL_TLS

– TLSv1

In diesem Szenario wird es bei der Standardeinstellung SSL_TLS belassen.

- **Provider**: Gibt einen der vordefinierten JSSE-Provider an. IBMJSSE ist der einzige vordefinierte JSSE-Provider, der auf der Plattform i5/OS unterstützt wird. IBMJSSEFIPS ist die FIPS-konforme (Federal Information Processing Standard) Version des IBMJSSE-Providers.
  In diesem Szenario wird die Einstellung bei IBMJSSE2 belassen.

- **Einstellungen für die Cipher Suite**: Gibt die verschiedenen Cipher-Suite-Gruppen an, die je nach Sicherheitsanforderungen ausgewählt werden können. Je stärker der Verschlüsselungsgrad der Cipher Suite ist, desto höher ist die Sicherheit. Ein starker Verschlüsselungsgrad kann jedoch Leistungseinbußen zur Folge haben. WebSphere Application Server hat drei Standardeinstellungen, ermöglicht aber auch Benutzerdefinierte Einstellungen, in der die Cipher Suite aus einer Liste ausgewählt werden können.

  Die drei Standard Einstellungen sind:

  - **Stark** – unterstützt folgende Chiffren:
    • SSL_RSA_WITH_RC4_128_MD5
    • SSL_RSA_WITH_RC4_128_SHA
    • SSL_RSA_WITH_AES_128_CBC_SHA
    • SSL_RSA_WITH_AES_256_CBC_SHA
    • SSL_DHE_RSA_WITH_AES_128_CBC_SHA
    • SSL_DHE_RSA_WITH_AES_256_CBC_SHA
    • SSL_DHE_DSS_WITH_AES_128_CBC_SHA
    • SSL_DHE_DSS_WITH_AES_256_CBC_SHA
    • SSL_RSA_WITH_3DES_EDE_CBC_SHA
    • SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
    • SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
    • SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
    • SSL_RSA_WITH_DES_CBC_SHA
    • SSL_RSA_FIPS_WITH_DES_CBC_SHA

- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5 (ebenso in **Mittel**)
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Mittel**)
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Mittel**)
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Mittel**)

- **Mittel** - unterstützt folgende Chiffren:
  - SSL_RSA_EXPORT_WITH_RC4_40_MD5 (ebenso in **Stark**)
  - SSL_RSA_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Stark**)
  - SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Stark**)
  - SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Stark**)
  - SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 (ebenso in **Schwach**)
  - SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Schwach**)

- **Schwach**, unterstützt folgende Chiffren:
  - SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 (ebenso in **Mittel**)
  - SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA (ebenso in **Mittel**)
  - SSL_DH_anon_WITH_DES_CBC_SHA
  - SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
  - SSL_DH_anon_WITH_RC4_128_MD5
  - SSL_DH_anon_WITH_AES_128_CBC_SHA
  - SSL_DH_anon_WITH_AES_256_CBC_SHA
  - SSL_RSA_WITH_NULL_MD5 (keine Verschlüsselung)
  - SSL_RSA_WITH_NULL_SHA (keine Verschlüsselung)

Aufgrund dessen, dass es vier Chiffren in „Stark“ gibt, die auch in „Mittel“ vorkommen, und zwei Chiffren, die in „Mittel“ und in „Schwach“ vorkommen, wird als Standard „Stark“ ausgewählt, um sicherzugehen, dass die Daten immer geschützt werden.

6.5 CICS Konfiguration für SSL Verarbeitung

In diesem Abschnitt wird erläutert wie CICS für serverseitige SSL Verarbeitung konfiguriert werden sollte. Dabei werden folgende Schritte durchlaufen:

- CICS für das Benutzen von SSL aktivieren
- TCPIPSERVICE für HTTPS definieren

Hinweis: An diese Stelle wird keine neue Pipeline für dieses Szenario definiert. Die Sicherheit befindet sich auf Transport Level, nicht auf Nachrichtenebene. Daher wird zwar ein neuer TCPIPSERVICE benötigt, jedoch keine neue PIPELINE.

6.5.1 CICS für die Verarbeitung von SSL aktivieren

CICS Zugriff auf die System SSL Bibliotheken sicherstellen

Die System SSL Bibliothek *hlq*.SIEALNKE muss der CICS Region in jeder der folgenden zugänglich gemacht werden:

- JOBLIB
- STEPLIB
- LINKLIST

Diese Linkliste kann mit folgendem Befehl überprüft werden (nur mit den entsprechenden Zugriffsrechten):

```
/D PROG,LNKLST
```

Sollte nach diesem Aufruf „SIEALNKE“ in der Liste erscheinen, dann bedeutet dies, dass die CICS Region Zugriff darauf hat.

Spezifizieren der SIT Parameter

Sechs SIT Parameter beziehen sich auf die SSL Verarbeitung (Erläuterung in Kapitel 4.3.3)

- CRLPROFILE: (Hierfür gibt es keinen Standardwert)
- ENCRYPTION: Dieser SIT Parameter kann auf WEAK, MEDIUM oder STRONG gesetzt werden. In den folgenden Tabellen wird die dargestellt, welche Cipher Suites in z/OS 1.9 unterstützt werden.

Cipher Suite	Encryption Algorithm	Key length	Hash algorithm	Key exchange	Certificate
01	None	None	MD5	RSA	RSA
02	None	None	SHA-1	RSA	RSA
03	RC4	40 bits	MD5	RSA	RSA
06	RC2	128 bits	MD5	RSA	RSA

Tabelle 6.1 Cipher Suite für Encryption = WEAK

Cipher Suite	Encryption Algorithm	Key length	Hash algorithm	Key exchange	Certificate
09	DES	56 bits	SHA-1	RSA	RSA

Tabelle 6.2: Cipher Suite für Encryption = MEDIUM zusätzlich zu den in Tabelle 5.1 aufgelisteten

Cipher Suite	Encryption Algorithm	Key length	Hash algorithm	Key exchange	Certificate
04	RC4	128 bits	MD5	RSA	RSA
05	RC4	128 bits	SHA-1	RSA	RSA
0A	Triple DES	168 bits	SHA-1	RSA	RSA
0C	DES	56 bits	SHA-1	fixed Diffie-Hellman	DSS
0D	Triple DES	168 bits	SHA-1	fixed Diffie-Hellman	DSS
0F	DES	56 bits	SHA-1	fixed Diffie-Hellman	RSA
10	Triple DES	168 bits	SHA-1	fixed Diffie-Hellman	RSA
12	DES	56 bits	SHA-1	ephemeral Diffie-Hellman	DSS
13	Triple DES	168 bits	SHA-1	ephemeral Diffie-Hellman	DSS
15	DES	56 bits	SHA-1	ephemeral Diffie-Hellman	RSA
16	Triple DES	168 bits	SHA-1	ephemeral Diffie-Hellman	RSA
2F	AES	128 bits	SHA-1	RSA	RSA
30	AES	128 bits	SHA-1	fixed Diffie-Hellman	DSS

Tabelle 6.3: C.Suites für Encryption = STRONG (zusätzlich zu den vorangegangenen Tabellen)

Cipher Suite	Encryption Algorithm	Key length	Hash algorithm	Key exchange	Certificate
31	AES	128 bits	SHA-1	fixed Diffie-Hellman	RSA
32	AES	128 bits	SHA-1	ephemeral Diffie-Hellman	DSS
33	AES	128 bits	SHA-1	ephemeral Diffie-Hellman	RSA
35	AES	256 bits	SHA-1	RSA	RSA
36	AES	256 bits	SHA-1	ephemeral Diffie-Hellman	DSS
37	AES	256 bits	SHA-1	ephemeral Diffie-Hellman	RSA
38	AES	256 bits	SHA-1	ephemeral Diffie-Hellman	DSS
39	AES	256 bits	SHA-1	ephemeral Diffie-Hellman	RSA

Tabelle 6.3(Fortsetzung)**:** C.Suites für Encryption = STRONG (zusätzlich zu den vorangegangenen Tabellen)

(Die Tabellen 6.1 bis 6.3 sind aus [IBM05] entnommen)

Für dieses Szenario wird `ENCRYPTPTION=STRONG` gesetzt.

- KEYRING: In Kapitel 6.3.2 wurde ein Key Ring erstellt. Daher wird für die CICS Region dieser Key Ring mit der Angabe KEYRING=CICS.CICSSERV1 spezifiziert.
- MAXSSLTCBS: Für dieses Szenario wird für die CICS Region MAXSSLTCBS=8 spezifiziert (default)
- SSLCACHE: Aufgrund dessen, dass nur mit einer CICS Region gearbeitet wird, wird der SIT Parameter SSLCACHE auf den Wert „CICS“ gesetzt.
- SSLDELAY: Der Wert für diesen SIT Parameter wird auf 600 gesetzt (default).

Die CICS Region wird nun neu gestartet. Die SIT Parameter sind nun wie folgt gesetzt (CRLPROFILE nicht spezifiziert):

```
ENCRYPTION=STRONG,
KEYRING=CICS.CICSSERV1,
MAXSSLTCBS=8,
SSLCACHE=CICS,
SSLDELAY=600,
```

6.5.2 Konfigurieren des TCPIPSERVICES

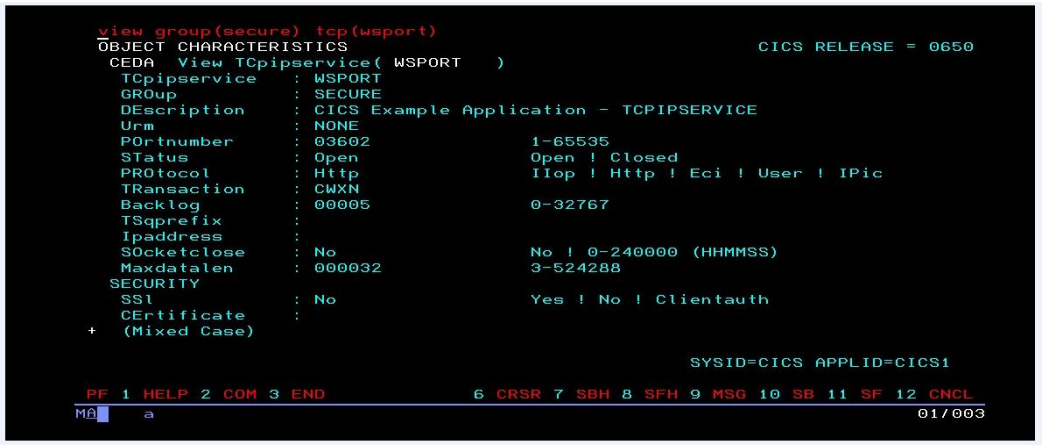

Abbildung 6.11: TCPIPSERVICE(WSPORT)

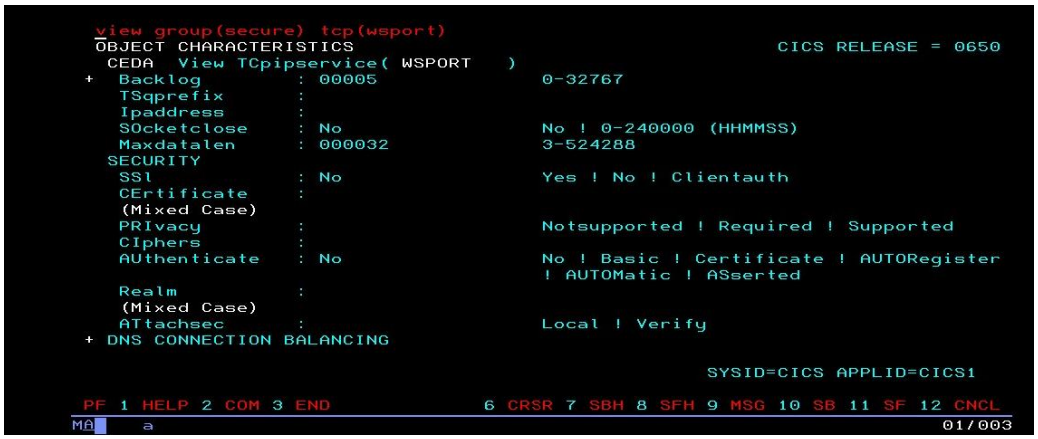

Abbildung 6.11: TCPIPSERVICE(WSPORT) (Fortsetzung)

In diesem Szenario ist die Sicherheit des TCPIPSERVICE namens WSPORT und der auf den Port 3602 horcht, noch nicht aktiviert. Da in unserem Szenario die Web Service

Kommunikation nur noch gesichert, also über HTTPS, laufen soll, modifizieren wir den momentan laufenden TCPIPSERVICE so, dass dieser SSL unterstützt. Eine andere Möglichkeit wäre es einfach einen neuen TCPIPSERVICE zu konfigurieren, der auf einen anderen Port lauscht und nur mit HTTPS kommuniziert. Dies wäre dann sinnvoll, wenn weitere Anfrage auf Services existieren, die nicht mit SSL verarbeitet werden müssen, weil die Informationen, die übertragen werden sollen, nicht kritisch sind. Dies ist jedoch in diesem Fall nicht nötig, da nur auf den getNameLength Service zugegriffen wird und hier nur *demonstriert* werden soll, wie mit kritischen Daten umgegangen wird.

Die Modifizierung des TCPIPSERVICES läuft folgendermaßen ab:

1.) Da der TCPIPSERVICE namens WSPORT momentan horcht, also „offen“ ist, muss diese zunächst „geschlossen“ werden

```
CEMT SET TCPIPS(WSPORT) CLOSED
```

In Abbildung 6.13 wird dargestellt, was passiert wenn dieser Befehl abgesetzt wurde.

```
SET TCPIPS(WSPORT) CLOSED
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tcpips(WSPORT  ) Ope Por(03602) Http Nos Tra(CWXN) Bas       BEING CLOSED
    Con(00002) Bac( 00005 ) Max( 000032 ) Urm( NONE     )
```

Abbildung 6.12: Schließen des TCPIPSERVICE(WSPORT)

2.) Als nächstes wird dieser TCPIPSERVICE für SSL Verarbeitung konfiguriert:

```
CEDA ALTER TCPIPS(WSPORT) GROUP(SECURE) SSL(YES)
CEDA ALTER TCPIPS(WSPORT) GROUP(SECURE)
CERTIFICATE(CICSSERV1)
CEDA ALTER TCPIPS(WSPORT) GROUP(SECURE)
CIPHERS(05042F303132330A1613100D)
```

3.) Der TCPIPSERVICE muss nun wieder installiert werden:

```
CEDA I TCPIPS(WSPORT) GROUP(SECURE)
```

4.) Und abschließend:

```
CEMT I TCPIPS(WSPORT)
```

In Abbildung 6.11 und Abbildung 6.12 wurde der TCPIPSERVICE namens WSPORT abgebildet. Im Folgenden werden die für dieses Szenario relevanten Felder des TCPIPSERVICE Ressource aufgezählt und deren Bedeutung erläutert.

Protocol: Wird auf `Http` gesetzt, was soviel bedeutet wie, dass diese TCPIPSERVICE Nachrichten in HTTP Format erwartet werden. TCPIPSERVICEs können ebenso für Internet Inter-ORB Protocol (IIOP), External Call Interface (ECI), IP Connections (IPIC) und Benutzer-definierte Protokolle verwendet werden.

Socketclose: Wird auf `No` gesetzt. CICS wird nie ein Socket schließen, jedoch warten bis der Client diesen Socket schließt. Die ermöglicht persistente Sockets und verringert den Overhead der beim Öffnen und Schließen für jede Anfrage entsteht. Das Ziel ist es SSL Handshaking auf ein Minimum zu reduzieren.

SSL: wird auf `Yes` gesetzt. Das bedeutet, dass dieser TCPIPSERVICE serverseitiges SSL verwenden wird, um seine Server Zertifikate, als einen Teil des Handshakes, zu übermitteln.

Certificate: wird auf `CICSSERV1` gesetzt, welches das Label des X.509 Zertifikates ist, das während dem SSL Handshake benutzt wird. Wird dieses Attribut ausgelassen, dann wird das Default Zertifikat, das in der dem Key Ring der CICS Region definiert ist, benutzt.

Ciphers: wird auf `0A1613100D05042F30313233` gesetzt. Darauf wird im Folgenden genauer eingegangen.

Authenticate: wird auf No gesetzt. Momentan benötigen wir keine Client Authentifizierung gegenüber CICS.

Ciphers in TCIPSERVICEs

Wie in 5.5.1 angesprochen, bestimmt der ENCRYPTION Parameter der SIT, welche Chiffren CICS für den Gebrauch zur Verfügung stehen.

Wenn ein TCPIPSERVICE SSL verwendet wird, dann werden standardmäßig die CIPHERS Attribute mit dem vollen Satz der von CICS Region unterstützten Chiffren initialisiert. Die CIPHERS Attribute können auch wieder auf diesen Wert re-initialisiert werden, in dem alle Chiffren innerhalb dessen gelöscht und dies mit Enter bestätigt.

Die Chiffren werden als Codes mit zwei Zeichen repräsentiert, welche sich auf die Tabellen 6.1 bis 6.3 beziehen. Aufgrund der Schwäche der Cipher Suites 01 und 02 und der Tatsache, dass diese keinerlei Verschlüsselung bieten, wird hier kurz auf diese eingegangen. Es ist nämlich nicht üblich, dass ein SSL Client nur diese Algorithmen benutzt, und um die Sicherheit aufrecht zu erhalten, werden diese aus der Liste der (verketteten) Cipher Suites entfernt.
Im Folgenden ist:

a) Die standardmäßig verwendete Chiffren für `Encryption = Strong`
b) Die modifizierte Liste (ohne Cipher Suite `01` und `02`)

a) **Chiffren:**
`050435363738392F303132330A1613100D0915120F0C0306`**`0201`**
b) **Chiffren:** `050435363738392F303132330A1613100D0915120F0C0306`

Einige Chiffren haben kurze Schlüssel, wodurch sie einfach geknackt werden könnten. Daher werden alle Chiffren aus der Liste entfernt, deren Schlüssel kleiner als 128 Bits lang sind.

Im Folgenden ist:

c) Die Liste ohne die Chiffren 01 und 02
d) Die Liste ohne die Chiffren 01 und 02 und ohne kleine Schlüssel

c) **Chiffren:** `050435363738392F303132330A1613100D0915120F0C0306`
d) **Chiffren:** `050435363738392F303132330A1613100D`

Andere Chiffren wiederum bieten eine großartige Kryptografische Verschlüsselung an, sind jedoch sehr rechenaufwendig, vor allem, wenn die von dem System verwendete Hardware dies nicht unterstützt. Wenn davon ausgeht, dass das System keine Hardware benutzt, die

beispielsweise AES mit 256-bit Schlüssel unterstützt, könnten diese ebenfalls aus der Liste entfernt werden.

Im Folgenden ist:

e) Die Liste ohne die Chiffren 01 und 02 und ohne kleine Schlüssel
f) Die Liste aus e) nur ohne AES mit 256 Schlüssel

e) **Chiffren:** `050435363738392F303132330A1613100D`
f) **Chiffren:** `05042F303132330A1613100D`

Die Schritte die in a) bis f) sind natürlich nicht nötig. Vielmehr soll hierbei demonstriert werden, wie die Chiffren angepasst werden können. CICS erlaubt es auch, dass die Reihenfolge der auftretenden Cipher Suites in den Chiffren verändert wird. CICS wird die Liste von links nach rechts durcharbeiten, um die erste Cipher Suite zu finden, die mit der des Client übereinstimmt.
Hinweis: Im Fall, dass CICS keine übereinstimmende Cipher Suites mit dem Client findet, wird die Verbindung geschlossen.

Mit den Chiffren die in f) aufgeführt sind, würde der TCPIPSERVICE zuerst alle möglichen 128-Bit Schlüssel Cipher Suites ausprobieren, anschließend mit den 168-Bit Schlüssel Cipher Suites fortsetzen. Dies ist sinnvoll, da die 128 Bit Verschlüsselung sicher ist und nicht so rechenaufwendig wie die 168-bit Schlüssel Verschlüsslung. In unserem Szenario sollen jedoch zuerst die 168-Bit Schlüssel ausprobiert werden und wenn es keine Übereinstimmung mit dem Client gibt, sollten erst dann die mit 128-Bit langen Schlüssel ausprobiert werden. Daher wird die Reihenfolge wie folgt geändert.

g) Die Liste der Cipher Suites aus f)
h) List aus g), jedoch beginnend mit den Schlüssel der Länge 168 Bit

g) **Chiffren:** `05042F30313233`**`0A1613100D`**
h) **Chiffren:** **`0A1613100D`**`05042F30313233`

Anschließend wird dies mit dem CEDA ALTER Befehl im TCIPSERVICE namens WSPORT abgeändert:

```
CEDA ALTER TCPIPS(WSPORT) GROUP(SECURE)
CIPHER(0A1613100D05042F30313233)
```

6.5.3 Definieren einer neuen WEBSERVICE Ressource

Wenn CICS den automatischen Pipeline Scan Mechanismus durchführt, um die WSBIND Datei aufzufinden, dann wird auch automatisch eine Standard URIMAP erstellt. Die Standard URIMAP ermöglicht es Anfragen auf den Web Service über HTTP zugreifen zu lassen.

Um ein automatisches Erstellen dieser Ressourcen zu verhindern, wird die wsbind-Datei (hier: getNameLength.wsbind) einfach in einen neu erzeugten Ordner im HFS verschoben. (Zum Beispiel /u/cicsts/cics1/provider/secured/). Dies ist nicht dasselbe Verzeichnis, den auch die PIPELINE(SECPIPRO) als Pickup-Verzeichnis benutzt.

Um eine neue WEBSERVICE Ressource zu erstellen, werden folgende Befehle ausgeführt:

```
CEOT TRANIDONLY
CEDA DEFINE GROUP(SECURE) WEBSERICE(getNameL)
PIPELINE(SECPIPRO)
WSBIND(/u/cicsts/cics1/provider/secured/getNameLength.wsbind)
```

Kleiner Hinweis: Der Befehl `CEOT TRANIDONLY` bewirkt, dass das Case-Sensitives Schreiben im Terminal möglich ist. Eine der Standardeinstellungen im Terminal ist, dass alles in Großbuchstaben geschrieben wird und da der UNIX System Service Case-Sensitive ist, würde die wsbind-Datei eventuell nicht gefunden werden.

Der WEBSERVICE wurde `getNameL` genannt. Wenn CICS einen Web Service installiert, ist eines der Ergebnisse des Pipeline Scans der, dass die automatisch erstellte WEBSERVICE Ressource, den gleichen Namen hat, wie die wsbind Datei. Wenn jedoch eine Ressource aus CSD (CICS System Definition) definiert wird, dann ist die Länge des Namens auf 8 Stellen beschränkt.

6.5.4 Definieren einer neuen URIMAP Ressource

Die durch ein Pipeline Scan automatisch erstellte URIMAP Ressource, besitzt folgende Standardwerte für die Attribute:

- `Scheme(Http).` Das bedeutet, dass diese URIMAP HTTP, HTTPS und MQ/JMS unterstützt.
- `Tcpipservice()` Das bedeutet, dass diese URIMAP der Anfragen, die von allen TCPIPSERVICE herrühren, zur Verfügung steht.
- `Host(*).` Das bedeutet, dass diese URIMAP für alle Anfragen, die von irgendeinem Host, oder jegliche MQ Queue stammen, zur Verfügung steht.

Der Grund warum eine URIMAP Ressource manuell definiert wird ist der, dass zum einen der Zugriff auf ein Web Service NUR über HTTPS ermöglicht werden kann, zum anderen, dass der Web Service einer bestimmten User ID zugeordnet ist, die mit bestimmten Zugriffsrechten bespikt ist. Unter dieser User ID wird der Web Service dann laufen.

Als nächstes muss der TCPIPSERVICE geschlossen und dann Installiert werden(siehe 5.5.2).

Bisher wurde der Server für serverseitiges SSL konfiguriert. Im folgenden Unterabschnitt wird die SSL Client Authentifizierung konfiguriert, damit sich der Web Service Requester authentifizieren und identifizieren kann.

6.6 Erstellen des WebSphere Client Zertifikates

In diesem Abschnitt wird erläutert wie:

- Das WebSphere Zertifikat erstellt,
- das WebSphere Zertifikat in eine Datei exportiert,
- und wie WebSphere so konfiguriert wird, dass das Zertifikat als Teil des SSL Handshakes versendet werden kann.

6.6.1 Erstellen des WebSphere Zertifikates

Um Clientseite SSL Verarbeitung zu aktivieren, wird ein Zertifikat und eine Schlüsselpaar erstellt, dass in WebSphere benutzt wird. Das Zertifikat und das Schlüsselpaar werden dann in einem Key Store abgelegt.

In diesem Abschnitt wird ein Selbstsignierendes Zertifikat für WebSphere erstellt, das als Client Zertifikat verwendet wird. Folgende Schritte werden dafür in der WebSphere Admin Konsole durchgeführt:

1.) **Sicherheit → Verwaltung von SSL Zertifikaten und Schlüsseln** anwählen
2.) Unter **Zugehörige Elemente**, wird **Keystores und Zertifikate** ausgewählt
3.) **NodeDefaultKeyStore** wird ausgewählt
4.) Unter **Weitere Merkmale**, wird **Persönliche Zertifikate** ausgewählt.

Es erscheint das Panel, das in Abbildung 5.14 dargestellt ist.

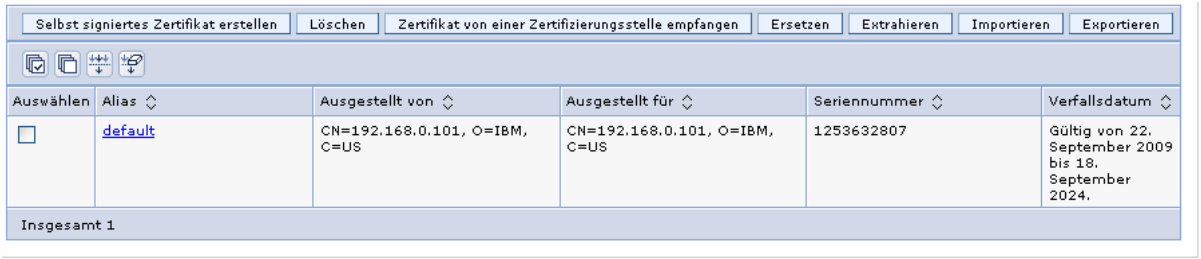

Abbildung 6.13: Persönliche Zertifikate im NodeDefaultKeyStore

5) In diesem Panel wird **Selbst signiertes Zertifikat erstellen** ausgewählt und folgende Zertifikat Informationen eingegeben:

Alias: **was61cert**
Key size: **1024**
Common name: **WAS61Cert02**
Validity Period: **365** (days)
Organization: **University**
Organization unit: **IT**
Locality: **Tuebingen**
State/Province: **BW**
Country: **DE**

Nach dem Bestätigen der eingegebenen Informationen ergibt sich folgendes Panel:

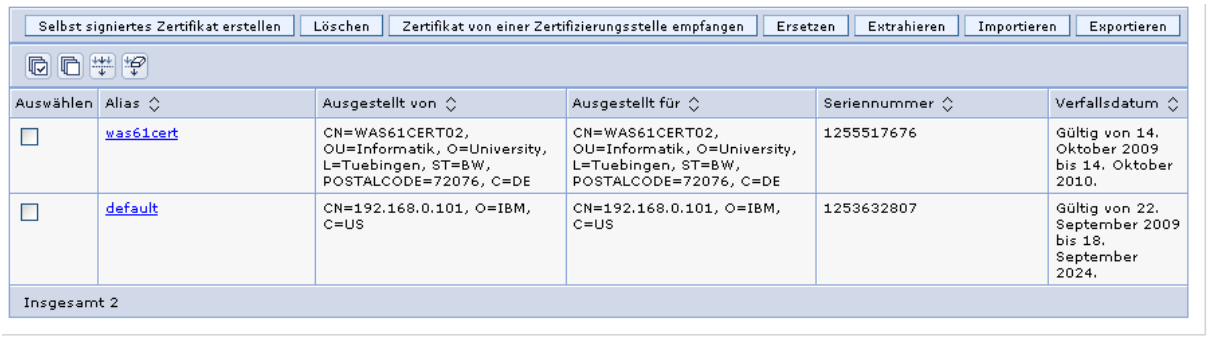

Abbildung 6.14: Selbst signiertes was61cert Zertifikat im NodeDefaultKeyStore

6.6.2 Exportieren des WebSphere Zertifikat in eine Datei

Als nächstes muss das Zertifikat und dessen öffentlicher Schlüssel in eine Datei exportiert werden. Der private (geheime) Schlüssel bleibt in dem Key Store.

1) In dem in Abbildung 6.14 dargestellten Panel, wird das Zertifikat mit dem Alias **was61cert** durch (das Setzen eines Häkchens) markiert und anschließend wird der **Extrahieren** Button angeklickt.

2) Anschließend erscheint das in Abbildung 6.15 dargestellte Panel:

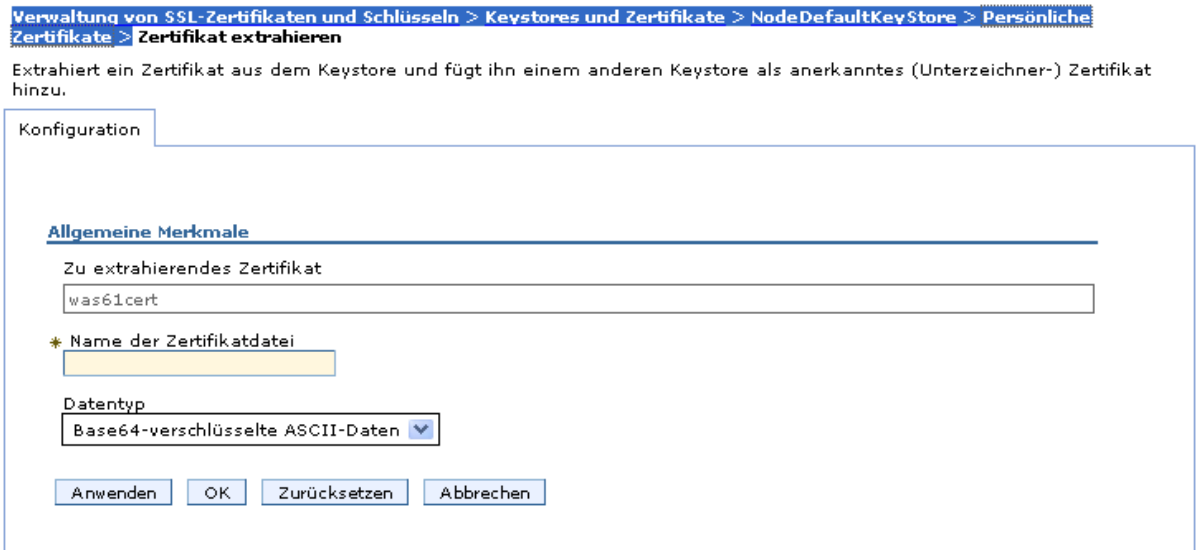

Abbildung 6.15: Exportieren des selbst signierten Zertifikat „was61cert“

2.) In das Feld Name der Zertifikatdatei wird der Name und der Ort der Datei angegeben, in das diese Datei exportiert werden soll: **C:\was61cert.cer**.
Den Datentyp **Base64-verschlüsselte ASCII-Daten** wird übernommen.
Schließlich wird dieser Vorgang mit einem Klick auf den OK Button beendet.

Wenn dieses Exportieren in die Datei erfolgreich war, befindet sich unter dem angegebenen Speicherort die Datei im Base64-Format.

6.6.3 Konfiguration des WebSphere, um das Client Zertifikat mittels SSL zu senden

In diesem Abschnitt wird WebSphere so konfiguriert, dass das selbst signierte Zertifikat als Client Zertifikat während dem SSL Handshake gesendet wird.

Hierzu sind folgende Schritte in der WebSphere Admin Konsole nötig:

1.) Unter **Sicherheit >Verwaltung von SSL-Zertifikaten und Schlüsseln > SSL-Konfigurationen** wird **NodeDefaulSSLSettings** angeklickt.
2.) Unter dem Drop-Down Box **Standardalias für Clientzertifikat** wird **was61cert** ausgewählt. Die Abbildung 6.16 zeigt das ausgewählt Panel.

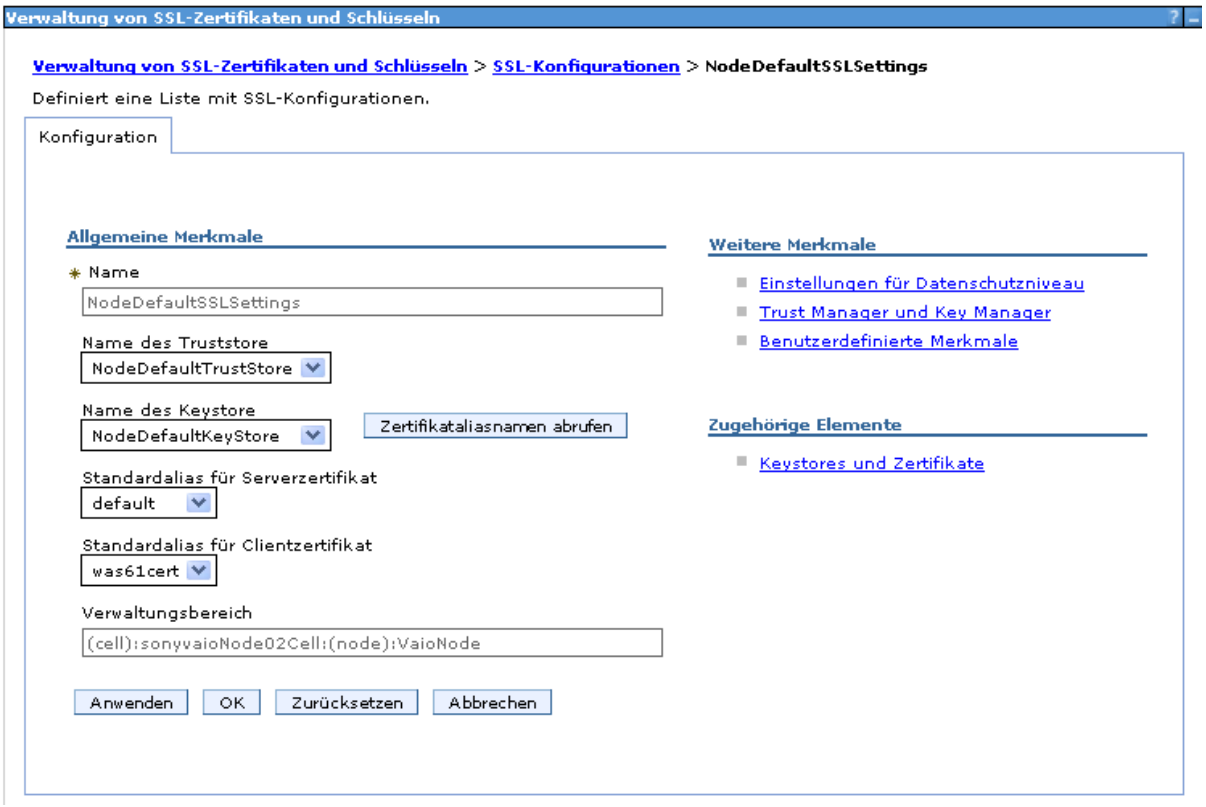

Abbildung 6.16: was61cert Zertifikat als Standardalias für Clientzertifikat

6.7 Konfiguration des CICS Service Provider für SSL Client Authentifizierung

In diesem Abschnitt wird gezeigt, wie CICS konfiguriert werden muss, damit während dem SSL Handshake nach einem Client Zertifikat gefragt wird. Dieses Zertifikat wird dafür verwendet, den Client zu authentifizieren.

Folgende Schritte sind hierfür nötig:

- TCPIPSERVICE für SSL Client Authentifizierung konfigurieren
- Client Zertifikat dem RACF hinzufügen
- Autorisierung des Service Requester, damit dieser den CICS Web Service aufrufen kann

6.7.1 Konfiguration des TCPIPSERVICE für SSL Client Authentifizierung

Es gibt zwei Parameter, die in dem TCPIPSERVICE namens WSPORT geändert werden müssen, damit die Client Authentifizierung aktiviert werden kann.

Zum einen muss das Attribut `SSL` auf `CLIENTAUTH` gesetzt werden und zum anderen muss das Attribut `AUTHENTICATE` auf `CERTIFICATE` gesetzt werden. Diese kann mit folgendem Befehl ausgeführt werden:

```
CEDA ALTER GROUP(SECURE) TCPIPSERVICE(WSPORT) SSL(CLIENTAUTH)
AUTHENTICATE(CERTIFICATE)
```

Nach diesem Befehl wird das TCPIPSERVICE Attribut `SSL` auf `CLIENTAUTH` gesetzt worden, was dazu führt, dass CICS für eine Verbindung zu diesem TCPIPSERVICE zusätzlich ein Client Zertifikat benötigt.

Das `AUTHENTICATE` Attribut wurde auf `CERTIFICATE` gesetzt. CICS benutzt das Client Zertifikat, um die User ID zu ermitteln, unter welcher die CICS Transaktionen laufen werden.

6.7.2 RACF ein Client Zertifikat hinzufügen

Im Folgenden Abschnitt wird RACF das Client Zertifikat hinzugefügt und die RACF ID, welche in Verbindung mit dem Zertifikat steht, wird spezifiziert. Hierzu sind folgende Schritte notwendig:

1.) Die Datei **was61cert.cert** in das z/OS System kopieren
2.) Das Zertifikat in RACF importieren und es einer gültigen User ID zuweisen.

Kopieren der Datei in das z/OS System

Folgendes sollte beim Übertragen der Datei in z/OS System, sei es nun mittels FTP oder per DRAG and DROP, beachtet werden:

- Base64 Kodierung ist ein Textformat, kein binäres Format, daher sollte dies vor der Übertragung eingestellt werden. Am Mainframe angekommen werden diese in das benötigte EBCDIC Format umgewandelt
- RACF erwartet Data Sets für Zertifikate mit einer variablen Blocklänge und 84 Byte Länge.
- Der Data Set Name der Datei sollte nicht mehr als 8 Zeichen betragen. Daher müsste was61cert.cer beispielsweise in was6cert.cer umbenannt werden

Ein Beispiel für das Kopieren von Zertifikaten mittels FTP befindet sich auf Seite 216 von [IBM05].

Importieren des Zertifikates in RACF und Abbildung auf eine gültige User ID

Das Zertifikat muss jetzt in RACF importiert werden und in Verbindung mit einer RACF User ID gebracht werden. Die folgenden Befehle werden hierzu ausgeführt:

```
RACDCERT ID(URIUSRID) ADD(CICS.WAS6CERT.CER) WITHLABEL ('was6cert')
TRUST
```

Dieser Befehl macht folgendes:

- Fügt das Zertifikat RACF hinzu
- Verbindet das Zertifikat mit der RACF User ID `URIUSRID`
- Erstellt das Zertifikat Label von `was6cert`
- Signalisiert, dass das Zertifikat dazu benutzt werden kann, ein User ID mit der `TRUST` Option zu authentifizieren.

Die ID `URIUSRID` ist die ID, die in der URIMAP Ressource manuell definiert wurde und `CICS.WAS6CERT.CER` das Data Set, in dieses das Zertifikat kopiert wurde.

6.7.3 Autorisierung des Service Requester

In diesem Abschnitt werden die RACF Befehle gezeigt, die benutzt werden um den Service Requester zu autorisieren, den Web Service getNameLength aufzurufen.

Zugriffserlaubnis erteilen

Folgender Befehl ermöglicht es einem Web Service Requester mit der User ID URIUSRID die Transaktion CPIH zu starten.

```
PERMIT CICS.CPIH CLASS(TCICSTRN) ID(URIUSRID) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

6.8 Umsetzung - Signieren von SOAP Nachrichten [Modell 2]

Um Integrität auf den Request und Response Nachrichten zu gewährleisten, kann eine digitale XML Signatur den Nachrichten hinzugefügt werden. CICS unterstützt die digitale XML Signatur sowohl im Service Provider als auch im Service Requester Modus.

In diesem Abschnitt wird dargestellt, wie eine SOAP Nachricht von dem WebSphere Application Server signiert werden kann und wie diese Signatur dann von CICS überprüft werden kann.
Dabei kann für die Konfiguration des Service Requester Application der *WebSphere Application Server Toolkit 6.1* oder der *Rational Application Developer 7.0* angewandt werden.

6.8.1 Szenario: SOAP Security

Folgendes Szenario soll hierbei umgesetzt werden:

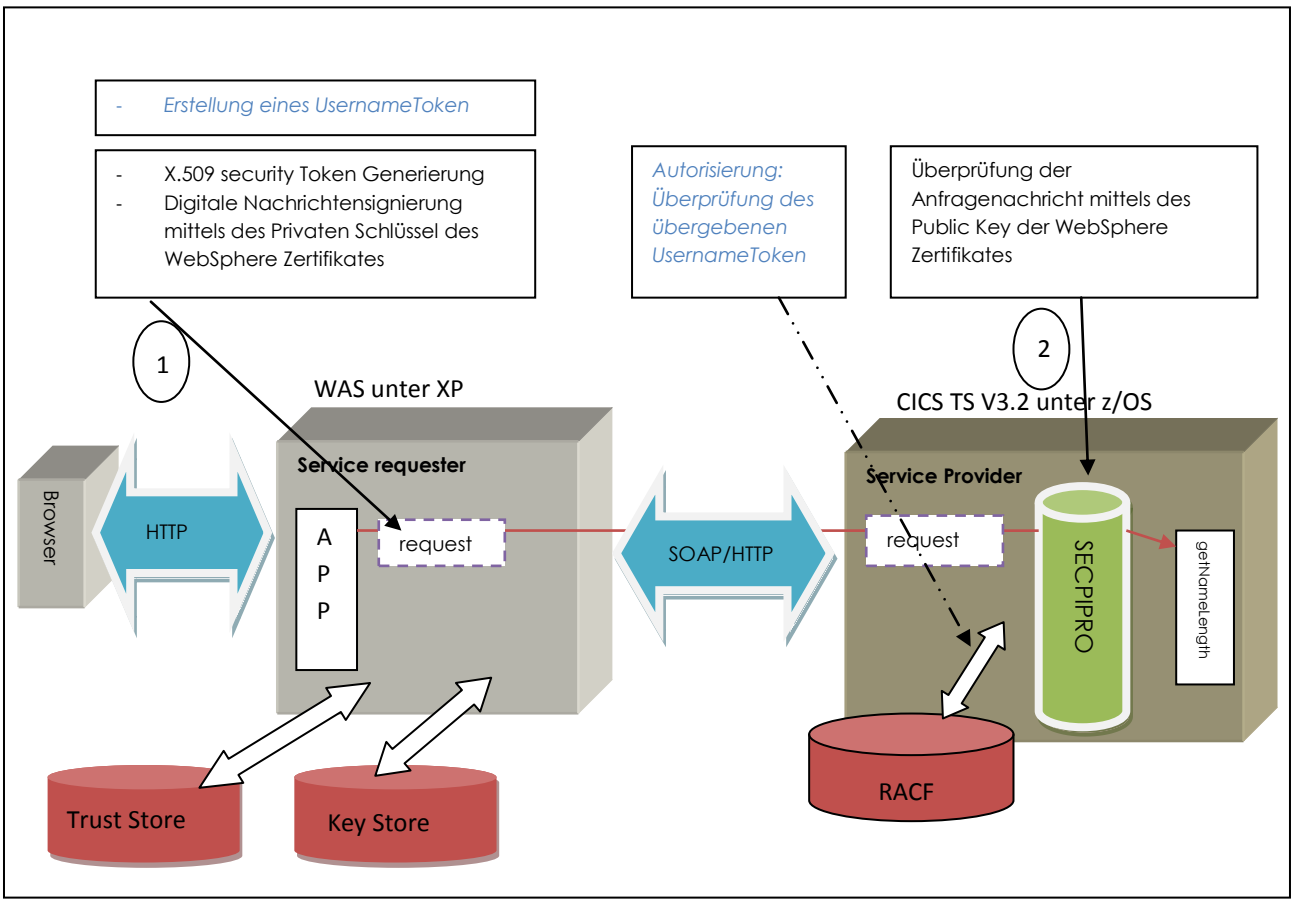

Abbildung 6.17: Szenario: Sicherheit auf Nachrichtenebene. (Blau beschriftetes Szenario ohne digitale Signierung, sondern ausschließlich über UsernameToken)

Erläuterung von Abbildung 6.17:

1.) Der Service Requester generiert aus dem X.509 Zertifikat ein BinarySecurityToken Element. Anschließend signiert es die Nachricht mit seinem privaten Schlüssel und schickt diese Nachricht an CICS. CICS kann nun mittels dem öffentlichen Schlüssel diese Signierung überprüfen und hat folglich eine Bestätigung, dass diese Nachricht nur von dem Service Requester stammt (also dem WAS).

2.) Wenn die Nachricht in CICS ankommt wird eine Reihe von Vorgängen ausgelöst. Eine Vorgang wird sein, dass die Pipeline namens SECPIPRO die Gültigkeit der Signatur der SOAP Nachricht überprüft, da in der Konfigurationsdatei dieser Pipeline einen Message Handler namens DFHWSSE1 hinzugefügt worden ist, der diese Aufgabe übernimmt. Anschließend ruft CICS RACF auf, um das WebSphere X.509 Zertifikat auf eine RACF

Benutzer ID abzubilden und legt diese abgebildete User ID in den DFHWS-USERID Container.

Alternative zu Schritt 2 (blau):
Im Service Requester wird der SOAP Nachricht erstmal nur eine Security Token in Form eines UsernameToken hinzugefügt. Dieser beinhaltet nur den Benutzernamen und das Passwort des Benutzers mit dem dieser sich normalerweise unter CICS anmelden würde. Dieser Schritt ohne weitere Sicherheitsmaßnahmen würde alleine nichts bringen, da diese Informationen im Klartext über das Netz transportiert werden und somit sehr einfach abzufangen wären. Warum diese Alternative trotzdem hier erwähnt wird, wird später erläutert.

3.) Eigentlich müsste hier noch einen dritten Punkt geben, in dem CICS nämlich die Response Message signiert und diese dem Service Requester zurückschickt, der dann in einem weiteren Schritt diese Signierung überprüfen wurde.

Im Vergleich zu der vorgestellten Methode mittels Sicherheit auf Transportebene ist es bei dem Signieren von Nachrichten nicht auf beiden Seiten zwingend nötig. Es könnte nur der Service Provider eine Antwort Nachricht signieren oder wie in diesem Fall, nur der Service Requester.

Um die Unterstützung für XML Signatur zu aktivieren müssen:

- Zertifikate und Schlüsselpaare bereitstellen,
- den WebSphere Requester für die digitale XML Signatur Verarbeitung und
- CICS für digitale XML Signatur Verarbeitung angeordnet werden.

6.8.2 Zertifikate und Schlüsselpaare vorbereiten

In Abbildung 6.17 wird auf der Server Seite *RACF* und auf der Client Seite der *Trust Store* und der *Key Store* dargestellt.

RACF:

- RACF beinhaltet die CICS Zertifikate und den öffentlichen Schlüssel von CICS. Der damit in Zusammenhang stehende private Schlüssel ist einem Private Key Data Set (PKDS) gespeichert.

  Wenn dieser Schlüssel nicht richtig gesichert ist, sind die kompletten Sicherheitsmaßnahmen seitens CICS unbrauchbar. Daher hat diese Geheimhaltung höchste Priorität und nur der dafür vorgesehen Benutzer oder die Anwendung sollte Zugriff auf diesen privaten Schlüssel haben.
  Das CICS Zertifikat wird in diesem Szenario mit dem Key Ring CICS.CICSSERV2 verbunden.
- RACF beinhaltet das WebSphere Zertifikat, welches benutzt wird um den Service Requester zu identifizieren.
- RACF beinhaltet ebenso die CICS und WebSphere CA Zertifikate.

WebSphere Key Store:

- Es beinhaltet die WebSphere Zertifikate und verwaltet auch den öffentlichen und privaten Schlüssel von WebSphere. Das WebSphere Zertifikat wird benutzt um die SOAP Request Nachrichten zu signieren.

WebSphere Trust Store:

- Es beinhaltet das CICS CA Zertifikat, dass für die Überprüfung der Gültigkeit eines von CICS signierten SOAP (Response) Nachricht zuständig ist.

Die nächsten Unterabschnitte erläutern die notwendigen Schritte, damit das Szenario aus Abbildung 6.17 realisiert werden kann.

1.) Da in diese Diplomarbeit nicht realisiert wird, wie CICS Nachrichten unter Benutzung von ICSF signiert, verweise ich hier nur die Quellen, die diese erläutern:
   - [IBM03] Kapitel 4 – Überblick und Einführung in ICSF
   - [IBM03] Kapitel 7.3.1
2.) Erstellung des CICS Zertifikates und Verbindung dieses Zertifikates mit dem Key Ring, dass von der CICS Region verwendet wird. Wird hier wie der vorherige Punkt nicht weiter realisiert, sondern nur auf folgende Quellen verwiesen:
   - [IBM03] Kapitel 7.3.2
3.) Erstellen des WebSphere Zertifikates
4.) Importieren des WebSphere Zertifikates in den Key Store
5.) Das CICS CA Zertifikat dem WebSphere Trust Store hinzufügen.

6.8.3 Erstellen des WebSphere Zertifikates

Um die Verarbeitung von XML Signaturen zu ermöglichen, werden ein Zertifikat und ein Schlüsselpaar, das von WebSphere angewendet wird, erstellt. Das Zertifikat und das Schlüsselpaar werden dann in einem Key Store abgelegt.

Hierzu wird wieder RACDCERT Befehl benutzt:

```
RACDCERT ID(CICSRI) GENCERT
     SUBJECTSDN(CN('WAS61CERT')
          T('WAS61CERT01-Zertifikat’)
          OU('Universitaet')
          O('IT’)
          L('BW')
          SP(‘Tuebingen’)
          C('DE'))
WITHLABEL('WAS61Serv01')
SIGNWITH(CERTAUTH LABEL('WAS6ROOT'))
NOTAFTER(DATE(2010/09/30))
SIZE(1024)
```

Abbildung 6.18: RACDCERT – Generierung eines Zertifikates und Public-Key Schlüsselpaares

Anschließend wird dieses Zertifikat noch exportiert und dann mittels des OPUT Befehles in eine HFS Datei kopiert. Von dort aus kann diese dann von dem Client importiert werden:

```
RACDCERT ID(CICSRI) EXPORT(LABEL('WAS61ROOT'))              +
DSN('CICS.WASSERV01.P12') FORMAT(PKCS12DER)
OPUT 'CICS.WASSERV01.P12' '/CICS/CERTIFICATES/ WASSERV01.P12'
+ BINARY CONVERT(NO)
```

Abbildung 6.19: Exportieren und Kopieren der Zertifikate in eine HFS Datei

6.8.3.1 Importieren des WebSphere Zertifikates in den Key Store

Als nächstes wird das WebSphere Zertifikat und Schlüsselpaar in den WebSphere Key Store eingefügt. Hierzu muss die Datei von dem Host heruntergeladen werden.

Dies kann zum einen über FTP geschehen oder ganz bequem mit der Drag-And-Drop Funktion des Rational Developer for System z. Nachdem dieses Zertifikat heruntergeladen wurde, muss dieses nun dem WebSphere Key Store hinzugefügt werden.
Im Folgenden wird das Zertifikat in dem Verzeichnis: C:\WASSERV01.P12 abgelegt.

Das Hinzufügen in den Key Store erfolgt mittels der Administrationskonsole. Hierzu werden folgende Schritte ausgeführt:

1.) Wähle **Security → „Verwaltung von SSL-Zertifikaten und Schlüsseln“**
2.) Unter „**Zugehörige Elemente**“ wird „**Keystores und Zertifikate**“ ausgewählt
3.) Das sich daraufhin öffnende Panel zeigt die Key Stores und Trust Stores an. Von den Angezeigten wird nun auf „**NodeDefaultKeyStore**“ geklickt.
4.) Unter „**Weitere Merkmale**“ wird auf „**Persönliche Zertifikate“** geklickt und im sich öffnenden Panels „Importieren“ ausgewählt. Dort werden dann folgende Werte eingetragen:

Name der Schlüsseldatei:	`C:\WASSERV01.P12`
Typ:	`PKCS12`
Kennwort der Schlüsseldatei:	`PASSWORT`

5.) Anschließend wird auf den Button „**Aliasnamen für Schlüsseldatei aufrufen“** geklickt. WebSphere zeigt jetzt den Aliasnamen in der Drop-Down Box „**Zu importierender Zertifikatalias“** anzeigt (in diesem Szenario: `WAS61CERT01)`. Dieser wird ausgewählt und dann auf OK geklickt.

6.) Änderungen speichern.

6.8.4 CICS Zertifikat dem Trust Store hinzufügen

Um jetzt das Setup der Zertifikate zu vervollständigen muss das CICS Zertifikat dem Trust Store hinzugefügt werden. Dies wurde bereits in 5.4.1 erläutert und erfolg analog dazu.

6.9. Konfiguration des Service Requester

Im folgenden Abschnitt wird dargelegt, wie die Client J2EE Anwendung konfiguriert werden muss, damit das Senden und Empfangen von SOAP Nachrichten ermöglicht werden kann. Die notwendigen Schritte sind hierbei:

1.) Importieren des EAR Projektes “getNameLengthClient” in den Application Server Toolkit(AST) oder alternativ in den Rational Application Developer (RAD).
2.) Den Service Request Generator für das Signieren konfigurieren.
3.) Den Response Consumer Generator für das Signieren konfigurieren.
4.) Die Web Service Client Anwendung erneut deployen.

Importieren der Client Anwendung

In einer neuen Workspace des AST oder RAD wird die Datei „getNameLengthClient.ear“ importiert.

Request Generators für das Signieren konfigurieren

Im AST sowie im RAD gibt es zwei Möglichkeiten wie das Signieren von Nachrichten konfiguriert werden kann. Entweder mittels des Web Service Wizard oder manuell durch den Web Service Editor.

Der Web Service Wizard ist ein Feature des AST 6.1, der folgendes ermöglicht:

- Hinzufügen einer digitalen XML Signatur,
- eines XML Verschlüsselung und

- Hinzufügen eines Stand-Alone Security Tokens.

Das Hinzusetzen einer digitalen Signatur wird anhand des WS-Security Wizards des WebSphere Application Server Toolkits v6.1demonstriert:

1.) Erweitern des JSR-109 Web Services in dem Projetexplorer und auswahl des Services „getNamelLengthService“. Dies ist der zugehörige Web Service Client der getNameLength Anwendung, die auf CICS als Service Provider fungiert. Dieser Service wird mittels der rechten Mausstaste angeklickt und die Option „Gesicherter Web-Service-Client“ →„Digitale XML-Signatur hinzufügen“, wie in Abbildung 5.20 dargestellt, ausgewählt.

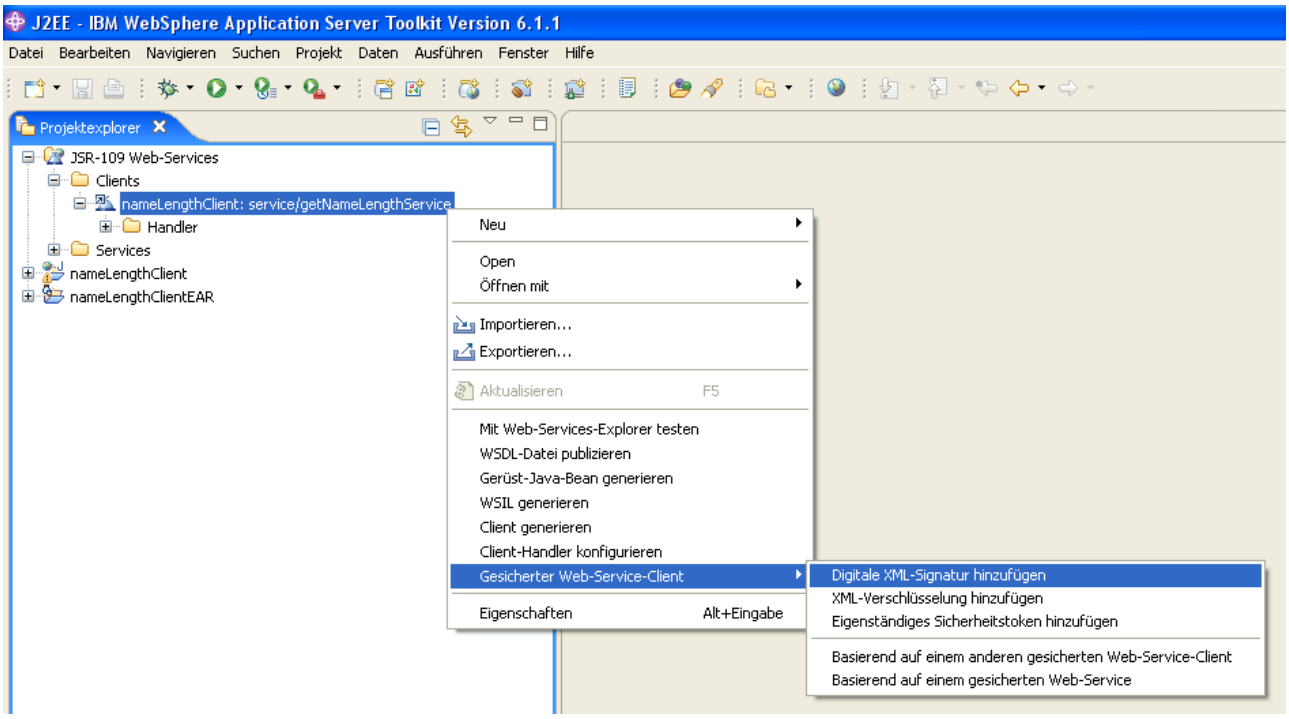

Abbildung 6.20: Application Server Toolkit 6.1 WS-Security wizard

2.) In dem Panel, das anschließend erscheint, werden die Standardeinstellungen übernommen. Dadurch fällt die Auswahl auf:

- RSA als Signaturverfahren
- Signierung des Body Elements der SOAP Nachricht und
- die Schlüsselinformation vom Typ STRREF.

STRREF (Direkte Referenz) wird hier als Schlüsselinformation ausgewählt, weil WebSphere dazu gebracht werden soll den öffentlichen Schlüssel als Teil des ganzen Zertifikates mit zusenden, da dieses Zertifikat zum einen für die Authentifizierung und zum anderen für das Entschlüsseln der signierten Nachricht, gebraucht wird.

Ein Klick auf den Next Button lässt das Panel, das in Abbildung 5.21 dargestellt ist, erscheinen.

3.) In dem Panel des Tokengenerators wird das WebSphere Zertifikat und die Informationen für den Schlüsselspeicher angegeben. (Aus den eingegebenen Daten der Abbildung 5.21 geht hervor, was für eine Art von Token in der Anfrage Nachricht gesendet werden soll und die Schlüsselinformation, die es WebSphere ermöglicht die Anfragen zu signieren.)

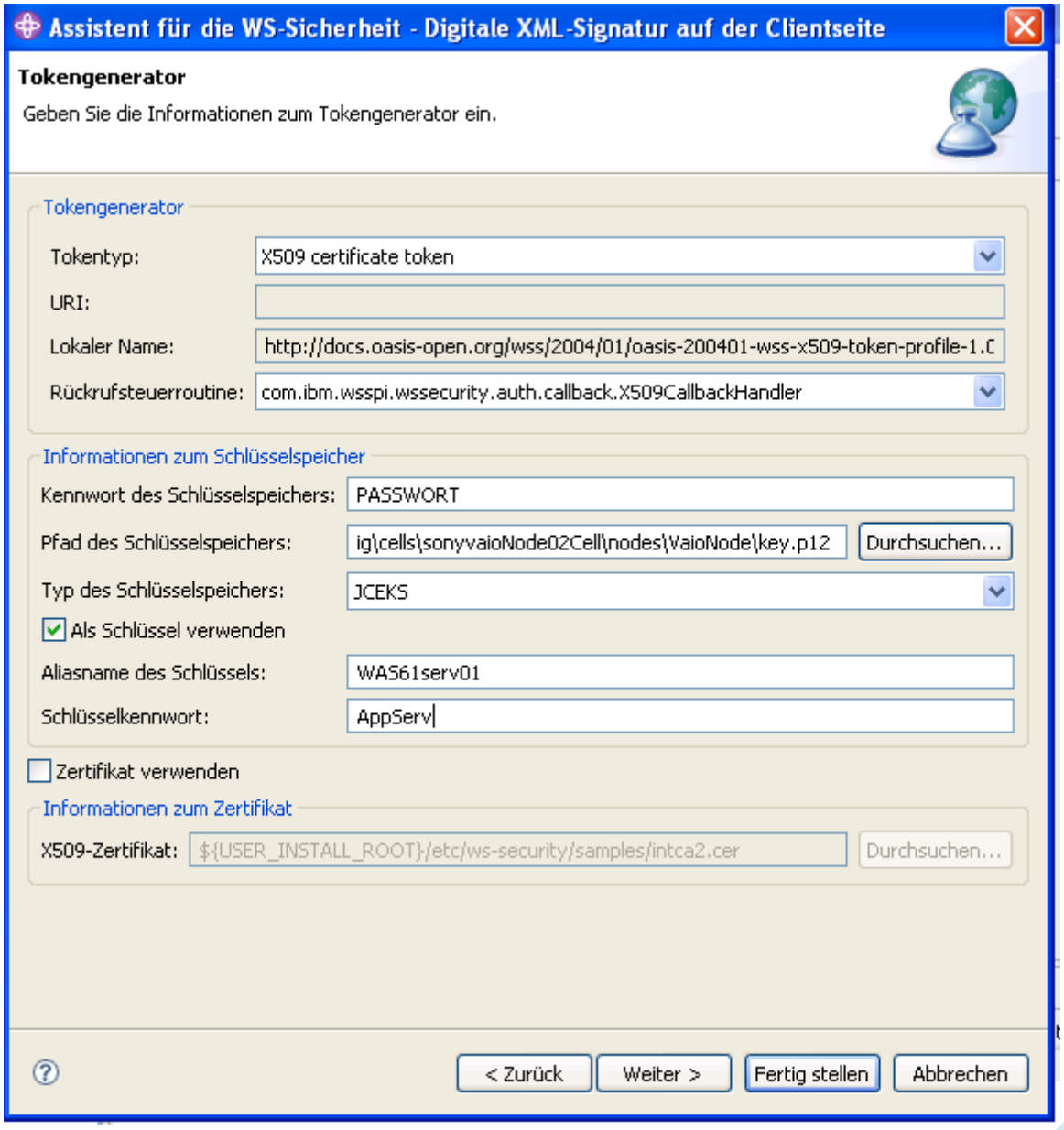

Abbildung 6.21: Application Server Toolkit WS-Security wizard - Tokengenerator

Die Rückrufsteuerroutine wird bei der Standardeinstellung belassen und als Token Typ wird **X509 certificate token v3** ausgewählt. Für die Informationen über den Schlüsselspeicher wird **Use a key** durch ein Häkchen markiert, als Kennwort für den Schlüsselspeicher **PASSWORT** eingetragen und als Pfad zu dem Schlüsselspeicher folgender Pfad angegeben: **E:\Programme\IBM\WebSphere\AppServer\profiles\AppSrv02\config\cells\sonyvaioNode02Cell\nodes\VaioNode\key.p12.**

Hierbei handelt es sich um den Standard WebSphere Schlüsselspeicher, dessen Speicherort von dem Installationspfad abhängt.

Desweiteren wurde als Typ des Schlüsselspeichers **PKCS12**, als Alias für den Schlüssel **was61serv01** und **AppServ** als Passwort angegeben.

Mit **Next** werden die Eingaben dieses Panels bestätigt und es erscheint das Panel:
Digitale Signatur für Antwortkonsumenten auf der Clientseite.

4.) In diesem Panel wird es bei allen Standardeinstellungen belassen. Mit einem betätigen des **Next** Buttons gelangt man schließlich zum Panel **Tokenkonsument**.

5.) In diesem Panel werden die Token Informationen für den Antwortkonsumenten spezifiziert. Hier wird spezifiziert was für ein Typ von Token in der Antwort Nachricht erwartet wird und Informationen über den Trust Store, welches WebSphere benötigt, um das Zertifikat, das für das Signieren der Antwort benutzt wurde, auf Gültigkeit zu Überprüfen.

   In dem Panel wird es bei den Standard JAAS Name belassen, als Token Typ **X509 Zertifikat Token v3** ausgewählt und **Zertifikate mit folgender Referenz anerkennen**, ausgewählt, wie es in Abbildung 5.22 dargestellt ist.

   In das Textfeld, das nach dem Kennwort für den Schlüsselspeicher frägt wird **AppServ** eingegeben, als Schlüsselspeichertyp **PKCS12** und als Pfad zum Schlüsselspeicher **E:\Programme\IBM\WebSphere\AppServer\profiles\AppSrv02\config\cells\sonyvaioNode02Cell\nodes\VaioNode\key.p12** angegeben.

   Durch das Betätigen des “Fertig stellen” Buttons erstellt dieser Wizard die entsprechenden Modifikationen im Deployment Deskriptor Dateien. Diese Einstellungen können durch das ein Doppelklick auf den getNameLengthService und anschließender Anwahl der Registerkarten **WS Extension** und **WS Binding** eingesehen werden.

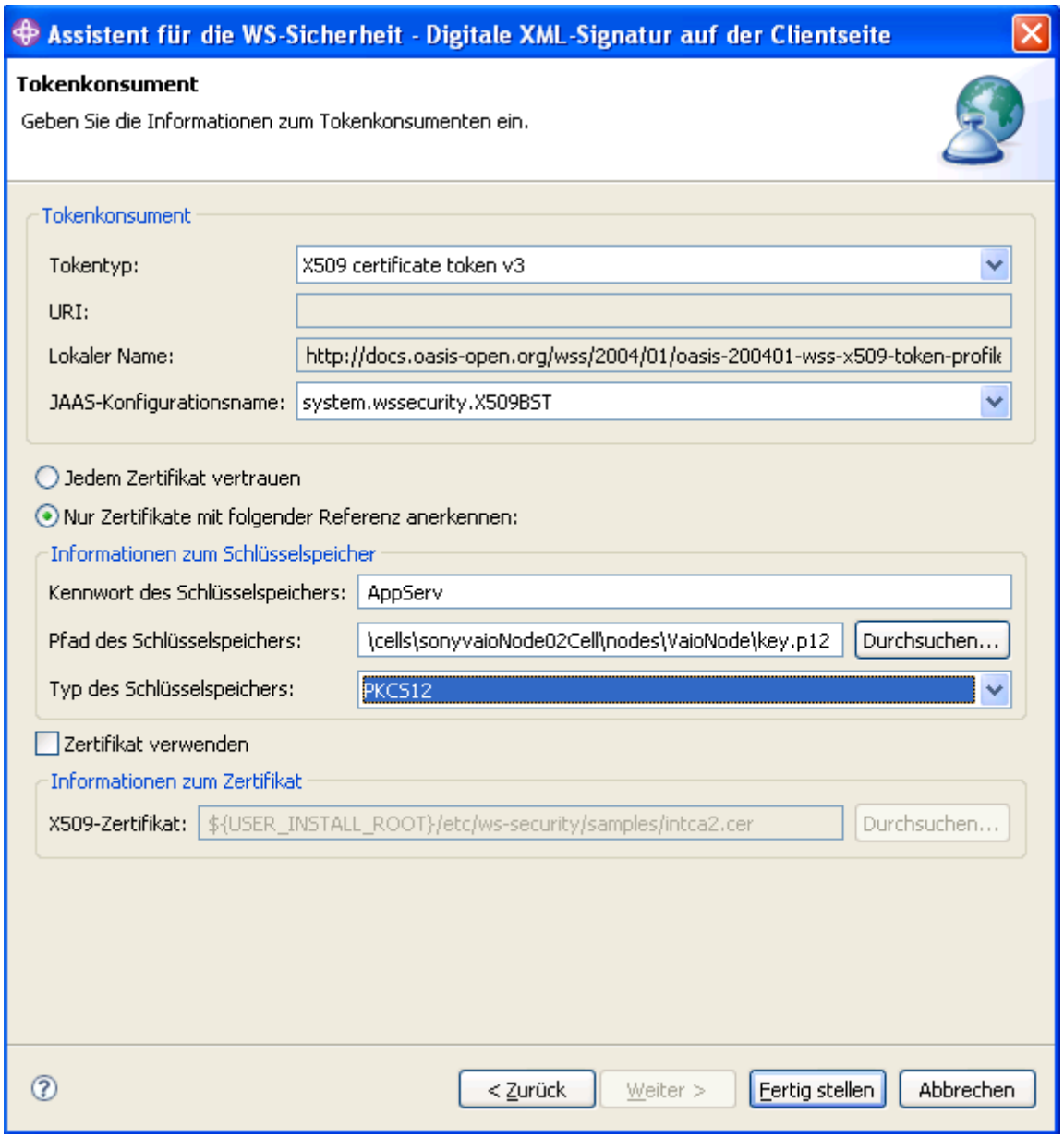

Abbildung 6.22: Application Server Toolkit WS-Security wizard - Tokenkonsument

6.10 Konfiguration von CICS für Signaturverarbeitung

In diesem Unterabschnitt wird erläutert wie die CICS Pipeline konfiguriert werden muss, um Signierte SOAP Nachrichten empfangen und versenden zu können.

Ein Beispiel für eine Pipeline Konfiguration Datei, die die einfachste Form eines Security Tokens erwartet, nämlich nur ein UsernameToken, ist in Abbildung 5.23 dargestellt:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
                                xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
provider.xsd ">
  <service>
      <service_handler_list>
            <wsse_handler>
                <dfhwsse_configuration version="1">
                  <authentication trust="blind" mode="basic">
                  </authentication>
                  </dfhwsse_configuration>
            </wsse_handler>
      </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Abbildung 6.23: Pipeline Konfigurationsdatei mit Security Message Handler

In diesem Beispiel wurden die Attribute des <authentication> Elements auf `trust="blind"` und `mode="basic"` gesetzt. Diese bewirkt das einkommende Nachrichten ein Identifizierungstoken besitzen müssen. Dieser Token muss aus einem Usernamen und optional einem Passwort bestehen. CICS legt diese User ID in den DFHWS-USERID Container. Wenn sich kein Passwort in dem Token befindet benutzt CICS diese User ID ohne sie zu überprüfen. Wenn ein Passwort dabei ist, dann wird er Security Handler DFHWSSE1 dieses überprüfen.

Wie in 3.4.4 bereits angesprochen können für die Attribute trust die Wert „blind“, „basic“, „signature“ und für das Attribut mode die Werte „none“, „basic“ und „signature“ eingesetzt werden. Fast alle Kombinationen sind möglich und ergeben ein bestimmtes Sicherheitsszenario. Ein Überblick über diese Tabelle und deren Kombinationen mit einer ausführlichen Erläuterung findet man auf den Seiten 80 in [IBM03].

Kapitel 7 – Zusammenfassung und Ausblick

Zusammenfassung:

Gesicherter Zugriff auf Informationen ist für jede Art von Business sehr wichtig. Vor allem für geschäftskritische Systeme, die vertrauliche Daten verwalten, wie es oft bei Systemen der Fall ist, die auf den IBM Customer Information Control System (CICS) basieren.
Die Sicherheit wird dann noch kritischer, wenn es sich um Implementierungen handelt, die auf der Service Orientierten Architektur (SOA) basieren. Der Grund ist, dass diese Art von Implementierungen zu Ihren Diensten und Anwendungen lose gekoppelt sind und deren möglichen Operationen auch über vertrauenswürdige Grenzen hinweg agieren.

Kapitel 2 gibt eine kurze Einführung in CICS Web Services. Es wird ein Überblick über die externen Standards gegeben, die von CICS unterstützt werden. Da die Web Service Description Language (WSDL) und in das Simple Object Access Protocol (SOAP) fundamental für das Benutzen von Web Services sind, werden diese ebenfalls angesprochen.

In Kapitel 3 werden die Sicherheitsmechanismen vorgestellt, die für das Sichern von CICS Web Service nötig sind. In einem Szenario wird gezeigt, wie ein Web Service Client ungesichert auf einen Web Service Provider zugreift, der unter CICS läuft. Hierbei wird verdeutlicht, wie ein Angreifer in das Geschehen eingreifen kann und um dem entgegenzuwirken werden die entsprechenden Sicherheitsmöglichkeiten vorgestellt. Da der Service Provider bestimmte CICS Ressourcen benutzt, die für das Sichern von Web Services eine wichtige Rolle spielen, wird auf diese Ressourcen detailliert eingegangen.

In Kapitel 4 wird die *Sicherheit auf Nachrichtenebene* behandelt. Nach einem Überblick über die Sicherheitsspezifikationen (*WS-Security*), die von CICS unterstützt werden, wird auf die Konfiguration des den von CICS bereitgestellten Security Handlers eingegangen. Ein Vergleich der bisher vorgestellten Sicherheitsmechanismen schließt diese Kapitel ab.

In Kapitel 5 wird die *Sicherheit auf Transportebene* behandelt. Ausgehend von der Verwendung von Basisauthentifizierung um einen Web Service Client zu authentifizieren, wird erläutert wie SSL/TLS in CICS nutzbar gemacht werden kann, um zum einen den Web Service Client zu authentifizieren und zum anderen Geheimhaltung und Integrität zu bewahren. Dabei wurden nicht nur die Konfigurationen aufgelistet, die an den Ressourcen

vorgenommen werden müssen vorgestellt, sondern darüber hinaus auch konkret die Einstellungen am System (wie z.B. Systeminitialisierung konfigurieren, Bibliotheken importieren…). Diese Einstellungen werden von den Sicherheitsarchitekten vorgenommen.

In Kapitel 6 werden zwei Lösungen für das Ausgangsproblem (Aufgabenstellung) vorgestellt. Nach Veranschaulichung der Ausgangssituation (Aufgabenstellung) zeigen zwei (Lösungs-) Modelle die Realisierung der Sicherheitsmechanismen, die in den vorangegangen Abschnitten behandelt wurden.

Das erste Modell, dass sich von den Kapitel 6.3 bis einschließlich 6.7 erstreckt, beschreibt ausführlich die Konfigurationen der CICS Region und der WebSphere Umgebung, um Sicherheit auf Transportebene zu benutzen.

Das zweite Modell, dass sich von Kapitel 6.8 bis zum Ende des 6. Kapitels erstreckt, zeigt wie SOAP Nachrichten mittels des WebSphere Application Server signiert werden können und wie diese Signatur von CICS verifiziert werden kann. Die Sicherheitskonfigurationen für den WebSphere Service Requester, die mittels des WebSphere Application Server Toolkit 6.1 durchgeführt werden und die Sicherheitskonfigurationen der CICS Service Provider Pipeline, werden Schritt für Schritt erläutert und werden von Systemprogrammierer durchgeführt.

Für welches Modell letztendlich entschieden wird, hängt von der Infrastruktur der verwendeten Architektur ab. Nachdem CICS einmalig für das Benutzen von SSL konfiguriert wird, ist der zeitliche Aufwand, um das Modell eins zu realisieren, dank der modernen Entwicklungstools minimal. Das Benutzen dieser Sicherheitseinrichtung ist trotzdem sehr teuer. Die Kosten, die im Modell eins anfallen liegen an den SSL Handshakes, die (zeitlich) sehr teuer sind. Diese Kosten können jedoch durch das Benutzen von kryptografischer Hardware und durch eine optimale Konfiguration, in der nur das mit einfließt was auch wirklich benötigt wird, reduziert werden. Ein Beispiel für eine derartige Konfiguration ist in 6.5.2 unter „Cipher in TCPIPSERVICEs“ demonstriert.

Die Kosten des ersten Modells fallen im zweiten Modell nicht an, jedoch entsteht hier durch Verschlüsselung/ Signierung ein enormer Overhead an Daten. Zusätzlich entsteht für die Berechnung der Verschlüsselung/ Signierung und für die Validierung dieser Verschlüsselung eine zeitliche Verzögerung, die auf die zusätzliche CPU-Benutzung zurückzuführen ist. Diese Validierung kann im Worst Case an jedem Knoten, der sich auf dem Weg zwischen WS Requester und WS Provider befindet, durchgeführt werden.

Ein Vergleich von Beispiel 4.1 auf Seite 54, in der eine einfache SOAP Nachricht ohne jegliche Sicherheitsaspekte abgebildet ist, mit der Abbildung, die sich in Appendix A abgebildet ist, verdeutlicht den Overhead an Informationen, der durch das Verschlüsseln und Signieren entsteht.

Existieren zwischen dem Service Provider und dem Service Requester verschiedene Dienste (sogenannte Intermediaries), dann fällt die Wahl nur noch auf das Modell zwei. SSL, das im Modell eins benutzt wird, sichert nur *Punkt zu Punkt*. Durch die WS-Security Spezifikation kann jedoch bei der Sicherheit auf Nachrichtenebene eine *Ende-zu-Ende* Sicherheit gewährleistet werden.

Ziel dieser Diplomarbeit ist es alle Informationen zur Verfügung zu stellen, damit ausgehend von einem bereits implementierten CICS Web Service, Anleitungen erstellt werden können. In diese werden alle nötigen Schritte und Konfigurationen aufzeigt, damit die Kommunikation zwischen einem Web Service Client und dem Web Service Provider auf CICS sicher ist. Es wurden dafür zwei Lösungsmodelle vorgestellt, die in Kapitel 6 veranschaulicht werden.

Ausblick:

Der Aufgabe dieser Diplomarbeit lag darin, CICS als Service Provider zu untersuchen. Eine weitere Möglichkeit, die in dieser Arbeit teilweise angesprochen wurde, ist CICS in der Rolle als Service Requester zu untersuchen. Dadurch, dass im Rahmen dieser Arbeit bereits die entsprechenden Konfigurationen an der CICS Region vorgenommen wurden, ist der Aufwand, der betrieben werden muss, um zum einen CICS als Service Requester zur Verfügung zu stellen, und zum anderen diesen dann zu sichern, nicht sehr hoch.

Für diese Diplomarbeit war vorgesehen, dass HTTP als Transportprotokoll verwendet wird. Eine von vielen weiteren Möglichkeiten, die ebenfalls untersucht werden kann, besteht darin WebSphere MQ für den Transport zu benutzen und anschließend zu sichern.

Ein weiterer Punkt, der kurz angesprochen wurde, betrifft die Optimierung. Eine Auflistung von allen zu optimierenden Optionen wäre sicherlich für einen Sicherheitsarchitekten, beim Entwurf einer Sicherheitsinfrastruktur von Bedeutung.

Eine weitere Möglichkeit, wie diese Arbeit fortgesetzt werden könnte, ist es ein Sicherheitsszenario zu erstellen, in dem ein Intermediary Server zwischen dem Client und dem Server geschaltet wird. Der Authentifizierungsmechanismus, dass hierbei in Frage kommen würde, nennt sich *Identity Assertion.*

In diesem Szenario würde sich ein Client gegenüber dem Intermediary Server authentifizieren und dieser würde dann die Identität des Clients, die Anfrage des Client und das Security Token des Intermediary Servers an den Server weiterleiten, der in diesem Fall die Server Provider Applikation ist, die unter CICS läuft. Ein Intermediary Server könnte entweder ein WebSphere Application Server oder die DataPower SOA Appliance sein.

Wie stark CICS Web Services in Zukunft durchsetzen werden, kann derzeit noch nicht gesagt werden. Jedoch werden intensive Forschungen (beispielsweise bei der IBM) im Bereich Service-Oriented Architecture geführt, um diese Technologie noch sicherer und vor allem effizienter zu machen. Diese klingen zum Teil sehr viel versprechend.

Glossary

AST	**A**pplication **S**erver **T**oolkit
CA	**C**ertificate **A**uthority
CICS	**C**ustomer **I**nformation **C**ontrol **S**ystem
CSD	(**C**ICS **S**ystem **D**efinition)
HTTP	**H**ypertext **T**ransfer **P**rotocol
HTTPS	**H**ypertext **T**ransfer **P**rotocol **S**ecure
FTP	**F**ile **T**ransfer **P**rotocol
ICSF	**I**ntegrated **C**ryptographic **S**ervice **F**acitlity
IIOP	**I**nternet **I**nter-**O**rb **P**rotocol
IP	**I**nternet **P**rotocol
IT	**I**nformation **T**echnology
JMS	**J**ava **M**essaging **S**ervice
JSP	**J**ava **S**erver **P**ages
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
RACF	**R**esource **A**ccess **C**ontrol **F**acility
P3P	**P**latform for **P**rivacy **P**references
PKI	**P**ublic **K**ey **I**nfrastructure
RAD	**R**ational **A**pplication **D**eveloper
SHA	**S**ecure **H**ash **A**lgorithm
SIT	**S**ystem **I**nitialisierungs**T**abelle
SOA	**S**ervice-**o**riented **A**rchitecture
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
SSL	**S**ecure **S**ocket **L**ayer
STS	**S**ecurity **T**oken **S**ervice
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TS	**T**ransaction **S**erver
UDDI	**U**niversal **D**escription, **D**iscovery and **I**ntegration
URL	**U**niform **R**esource **L**ocator
URI	**U**niform **R**esource **I**dentifier
W3C	**W**orld **W**ide **W**eb **C**onsortium
WAS	**W**ebSphere **A**pplication **S**erver
WSD	**W**eb**S**phere **D**eveloper
WSDL	**W**eb **S**ervices **D**escription **L**anguage
WS-I	**W**eb **S**ervice **I**nteroperbility **O**rganisation
WSS	**W**eb **S**ervices **S**ecurity
WSSE	**W**eb **S**ervices **S**ecurity **E**xtension
XML	**E**xtensible **M**arkup **L**anguage
XSD	**X**ML **S**chema **D**efinition Language

Literaturverzeichnis

[IBM01] Redbook: *Implementing CICS Web Services*
(WWW Dokument)
Quelle: http://www.redbooks.ibm.com/redbooks/pdfs/sg247206.pdf

[IBM02] Redbook: *Considerations for CICS Web Services Performance*
(WWW Dokument)
Quelle: http://www.redbooks.ibm.com/redbooks/pdfs/sg247687.pdf

[IBM03] Redbook: *CICS TS V3.2 Web Service Guide*
(WWW Dokument)
Quelle:http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/topic/com.ibm.cics.ts.webservices.doc/pdf/dfhwsc00.pdf

[IBM04] IBM: *UDDI Technical White Paper*
(WWW Dokument)
Quelle: http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

[IBM05] Redbook: *Securing CICS Web Services*
(WWW Dokument)
Quelle: http://www.redbooks.ibm.com/redbooks/pdfs/sg247658.pdf

[IBM06] Redbook: *Security Server RACF Security Administrator's Guide*
(WWW Dokument)
Quelle:http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SA22-7683-04#

[IBM07] Redbook: CICS TS V32 RACF Security Guide
(WWW Dokument)
Quelle:http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PB_L=SC34-6835-00

[IBM08] Demo: Web Service Bottom Up Video
(WWW Dokument)
Quelle:http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.wdz/wdz/7.0z/WebServicesDev/EST_BottomUpGen/EST_BottomUpGen_viewlet_swf.html?dmuid=20071011165326872091&noframes=true

[IBM09] Demo: *Web Service Bottom Up Video*
(WWW Dokument)
Quelle:http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.wdz/wdz/7.0z/WebServicesDev/EST_BottomUpDeploy/EST_BottomUpDeploy_viewlet_swf.html?dmuid=20071011165329558872&noframes=true

[IBM10] Redbook: *z/OS Security Server RACF Command Language Reference*
(WWW Dokument)
Quelle:http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SA22-7687-06#

[OASIS01] OASIS: *Web Services Security: SOAP Message Security 1.1*
(WWW Dokument)
Quelle: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[OASIS02] OASIS: *Web Services Security- UsernameToken Profile 1.0*
(WWW Dokument)
Quelle: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

[OASIS03] OASIS: *Web Services Security- X.509 Certificate Token Profile*
(WWW Dokument)
Quelle: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[XML01] W3C: *Extensible Markup Language (XML) 1.0 (Zweite Auflage)*
(WWW Dokument)
Quelle: http://www.edition-w3c.de/TR/2000/REC-xml-20001006/

[XML02] W3C: *Encryption Syntax and Processing*
(WWW Dokument)
Quelle: http://www.w3.org/TR/xmlenc-core/

[XML03] W3C: *XML Signature Syntax and Processing*
(WWW Dokument)
Quelle: http://www.w3.org/TR/xmldsig-core/

[XML04] W3C: *XML-binary Optimized Packaging*
(WWW Dokument)
Quelle: http://www.w3.org/TR/xop10/

[WS-I01] W3C: *WS-I Basic Profile Version 1.0*
(WWW Dokument)
Quelle: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[WS-I02] W3C: *Simple SOAP Binding Profile Version 1.0*
(WWW Dokument)
Quelle: http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html

[WS-I03] W3C: *Basic Profile Version 1.1*
(WWW Dokument)
Quelle: http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[W3C01] W3C: *Web Services Description Language (WSDL) 1.1*
(WWW Dokument)
Quelle: http://www.w3.org/TR/wsdl

[W3C02] W3C: *WSDL 1.1 Binding Extension for SOAP 1.2*
(WWW Dokument)
Quelle: http://www.w3.org/Submission/wsdl11soap12/

[W3C03] W3C: *WSDL Version 2.0 Part2: Web Services Message Exchange Patterns*
(WWW Dokument)
Quelle: http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326/

[W3C04] W3C: *Simple Object Access Protocol (SOAP) 1.1*
(WWW Dokument)
Quelle: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[W3C05] W3C: *SOAP Version 1.2 Part 0: Primer*
(WWW Dokument)
Quelle: http://www.w3.org/TR/soap12-part0/

[W3C06] W3C: *SOAP Version 1.2 Part 1: Messaging Framework (W3C Empfehlung)*
(WWW Dokument)
Quelle: http://www.w3.org/TR/soap12-part1/

[W3C07] W3C: *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*
(WWW Dokument)
Quelle: http://www.w3.org/TR/soap12-part2/

[W3C08] W3C: *SOAP Version 1.2 Teil 0: Einführung*
(WWW Dokument)
Quelle: http://poseidon.home.tlink.de/w3c/REC-soap12-part0-20030624-de_DE/#L1165

[WIKI01] Wikipedia: *UDDI*
(WWW Dokument)
Quelle:http://de.wikipedia.org/wiki/Universal_Description,_Discovery_and_Integration

[RFC01] RFC: *HTTP Authentication: Basic and Digest Access Authentication*
(WWW Dokument)
Quelle: http://www.ietf.org/rfc/rfc2617.txt

[RFC02] RFC: *TLS Protocol*
(WWW Dokument)
Quelle: http://www.ietf.org/rfc/rfc2246.txt

[UniTue] Info: *Hardware und Software des System zRechners der Universität Tübingen*
(WWW Dokument)
Quelle: http://hobbit.cs.uni-tuebingen.de/rechner/rechner.html

Index

T

U

W

X

Appendix A

SOAP Nachricht: Signierung

Das folgende Beispiel zeigt eine signierte SOAP Nachricht. Der Teil der Nachricht, der zusätzlich durch die Signierung generiert wurde, ist hier rot markiert.

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header>
         <wsu:Timestamp
         xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
         security-utility-1.0.xsd">
         <wsu:Created>2004-11-26T09:32:31.759Z</wsu:Created>
         </wsu:Timestamp>
         <wsse:Security soapenv:mustUnderstand="1"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wsswssecurity-
            secext-1.0.xsd">
            <wsse:BinarySecurityToken
               EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                     wss-soap-message-security-1.0#Base64Binary"
               ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x5
                     09-token-profile-1.0#X509"
               wsu:Id="x509bst_1080660497650146620"
               xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
               security-utility-1.0.xsd">
                   MIIBzzCCATigAwIBAgIEQZnQ7DANBgkqhkiG9w0BAQQFADAsMQswCQYDVQQGEwJVUz
                   EMMAoGA1UECxMDSUJNMQ8wDQYDVQQDEwZDbGllbnQwHhcNMDQxMTE2MTAwNTMyWhcN
                   MDUwMjE0MTAwNTMyWjAsMQswCQYDVQQGEwJVUzEMMAoGA1UECxMDSUJNMQ8wDQYDVQ
                   QDEwZDbGlfKTYLOqMAxq6YvfFweo6Z0H7r3+jAZu880Z7Ru6iMFvajpxrRJshesPnp
                   66binS5vQqfmPAUxyhk+IMUAoFZQvG4byEwzDPwmys7M8Q4uZfUK7R/6PocAtwVCss
                   x6zGsNcaaDkGYDC/5qK8+95y4ofAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEArrrFUwS
                   VHvWt05eVBImRu8t3cBbPdlQruBOuCHrNHoG9TwoCJE2JTeZV7+bCjfB17sD8K3mu/
                   WIvS2OFoTZWxEGnbxzHQ5aJb7ZCGQZ+ffclLCxWj+pG2Eg+BbWR2xStH3NJgPUPmpi
                   ei6f5fkht16e+CDN9XXYMlqiYhR9FkdI=
            </wsse:BinarySecurityToken>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
             <ds:SignedInfo>
                       <ds:CanonicalizationMethod
                             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                             <ec:InclusiveNamespaces
                             PrefixList="wsse ds xsi soapenc xsd soapenv "
                             xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                       </ds:CanonicalizationMethod>
             <ds:SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
              <ds:Reference
                    URI="#wssecurity_signature_id_7018050865908551142">
                    <ds:Transforms>
                       <ds:Transform
                            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                       <ec:InclusiveNamespaces
                            PrefixList="xsi soapenc p821 xsd wsu soapenv "
                            xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                       </ds:Transform>
                    </ds:Transforms>
                    <ds:DigestMethod
                          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                     <ds:DigestValue>53Ed8o+4P7XquUGuRVm50AbQ4XY=</ds:DigestValue>
              </ds:Reference>
```

```
            </ds:SignedInfo>
                  <ds:SignatureValue>
                        SbhHeGPrsoyxXbTPdIEcybfqvoEdZ1KiYjjvWZL/dvgqMS6/oi0cdR2Di
                        08VNombjHf9h/EAov+/zvt8i5enw5AziecKr6atLVNG4jKbuNORAhts24
                        2bBfybUks4YzduoWYcDU9EIXUCMTjRiMbWVvwgc1k4VhTUmKb3jN+yeRA
                        =
                  </ds:SignatureValue>
                  <ds:KeyInfo>
                        <wsse:SecurityTokenReference>
                              <wsse:Reference URI="#x509bst_1080660497650146620"
                                    ValueType="http://docs.oasis-
                                open.org/wss/2004/01/oasis-
                                          200401-wss-x509-token-profile-
                                      1.0#X509"/>
                        </wsse:SecurityTokenReference>
                  </ds:KeyInfo>
               </ds:Signature>
            </wsse:Security>
         </soapenv:Header>
         <soapenv:Body
             wsu:Id="wssecurity_signature_id_7018050865908551142"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
             wss-wssecurity-utility-1.0.xsd">
             <p821:getDayForecast xmlns:p821="http://bean.itso">
                   <theDate>2004-11-25T15:00:00.000Z</theDate>
             </p821:getDayForecast>
         </soapenv:Body>
      </soapenv:Envelope>
```

Quelle[IBM05]

Appendix B

SOAP Nachricht: Geheimhaltung

Das folgende Beispiel ist eine SOAP Nachricht, deren Body Teil verschlüsselt wurde. Die Nachricht wurde mit dem öffentlichen Schlüssel eines Empfängers verschlüsselt. Nur der Empfänger dieser Nachricht besitzt den entsprechenden Schlüssel (privaten Schlüssel), um diese Nachricht zu entschlüsseln. Er benötigt jedoch noch zusätzliche Informationen über die Art der Verschlüsselung, die sich im Header befindet. Der Teil der Nachricht, der für die Verschlüsselung generiert wurde, ist hier rot markiert.

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
       <wsu:Timestamp
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
              security-utility-1.0.xsd">
                     <wsu:Created>2004-11-26T09:34:50.838Z</wsu:Created>
       </wsu:Timestamp>
       <wsse:Security soapenv:mustUnderstand="1"
              xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-w
              ssecurity-secext-1.0.xsd">
              <EncryptedKey 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#">
                        <wsse:SecurityTokenReference>
                          <wsse:KeyIdentifier
                            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                            200401-wss-x509-token-profile-1.0#X509v3SubjectKeyIdentifier">
                            Vniy7MUOXBumPoH1MNbDpiIWOPA=
                          </wsse:KeyIdentifier>
                        </wsse:SecurityTokenReference>
                     </ds:KeyInfo>
                     <CipherData>
                       <CipherValue>
                          O+2mTsRjU1iNlwANv1kGdzpkRV1GQc5epAT3p5Eg5UNAJ3H3YAX5VrdgMQmj1wzd
                          SZLDEzBtcHPJq3c8c0AgmAy9EVdcgXIn/ZeV+80jMDn/HN2HfodYjURtIYBg480S
                          Skot0fy+YpBSXNR/MTfs1HT2H/Mjw/CyIbomWdQZHmE=
                       </CipherValue>
                     </CipherData>
                     <ReferenceList>
                        <DataReference
                          URI="#wssecurity_encryption_id_6866950837840688804"/>
                     </ReferenceList>
              </EncryptedKey>
       </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
       <EncryptedData
              Id="wssecurity_encryption_id_6866950837840688804"
              Type="http://www.w3.org/2001/04/xmlenc#Content"
              xmlns="http://www.w3.org/2001/04/xmlenc#">
         <EncryptionMethod
              Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
              <CipherData>
                   <CipherValue>
                     OvLekOlbuZhFBllBNL4Kos195YHwYw0kSbMxkbI2pk7nl17g0prPS2Ba2hyrXHABGQVmosWp
                     gqt+zijCPHUQCMwmm3qgFraK11DPMmwP94HvgxlgBmPw1Unt+WM4aKLNrHDnwwcQX5RO7KT+
                     fhFp4wxFEABwfHqzvTGNK3xRwJE=
              </CipherValue>
              </CipherData>
       </EncryptedData>
    </soapenv:Body>
</soapenv:Envelope>
```

Quelle[IBM05]

Appendix C

Inhalt der CD:

Die beigefügte CD weist folgende Verzeichnisstruktur auf:

In dem Verzeichnis **Anlagen** befinden sich:

- der Web Service Client (getNameLength.ear)
- die wsbind-Datei (getNameLength.wsbind)
- die wsdl-Datei (getNameLength.wsdl)
- die Pipelinekonfigurationsdateien:
  - basic, ohne security (basissoap11provider.xml)
  - mit konfigurierten Security Handler (basicsoap11provider.xml).

In dem Verzeichnis **Diplomarbeit** befindet sich die Diplomarbeit im PDF-Format.

In dem Verzeichnis **PDF** liegen alle Quellen als PDF-Dateien vor, auf die in den Kapiteln dieser Arbeit verwiesen wurde. Die Bezeichnung der einzelnen PDF Dateien ist dabei identisch mit der Bezeichnung der Quellen, auf die verwiesen wurde.

In dem Verzeichnis **IBM Video Demo** befindet sich ein Videotutorial, das demonstriert wie ein Web Service mittels der „Bottum Up“ Methode auf dem WebSphere Developer for System z entwickelt wird.