

# Web Service Security

Alexander Fuchs  
e0106909@student.tuwien.ac.at  
Mat.Nr.: 0106909

Gernot Goluch  
e0026039@student.tuwien.ac.at  
Mat.Nr.: 0026039

Thomas Verhounig  
thomas\_v@gmx.at  
Mat.Nr.: 0126429

11. Januar 2006

## Zusammenfassung

Am Beginn der Arbeit werden die Basiskonzepte von WS-Security, XML-Signature und XML-Encryption vorgestellt. XML-Signature ist ein Sicherheitsstandard der in erster Linie Integrität des Nachrichtenaustausches gewährleisten soll. Der Sender einer XML-Nachricht hängt eine digitale Signatur an die Nachricht an. Diese Signatur gibt dem Empfänger der Nachricht die Möglichkeit, diese auf Veränderung zu prüfen. XML-Encryption ist ein weiterer Sicherheitsstandard der es ermöglicht XML-Nachrichten beziehungsweise Teile von XML-Nachrichten zu verschlüsseln, um so die Inhalte vor unberechtigten Einblicken durch Dritte zu schützen. Im Anschluss daran wird die praktische Anwendung des WS-Security Konzepts auf SOAP (Simple Object Access Protocol) behandelt. Hierfür wird kurz auf Aufbau und Funktionsweise von SAML (Security Assertion Markup Language) eingegangen, um die Authentifikation und Autorisation im WS-Security Prozess zu ermöglichen. Ein weiterer wichtiger Bestandteil des angesprochenen Konzeptes ist WS-Policy, der es uns ermöglicht spezifische Richtlinien beziehungsweise Regelsets (Policies) für Web Services zu entwickeln. Abschließend wird anhand von WSS4J, einer Apache Implementierung, WS-Security in der Anwendung gezeigt.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>WS-Security Basis-Konzepte</b>	<b>4</b>
2.1	XML-Signature . . . . .	4
2.1.1	Komponenten einer XML-Signatur . . . . .	5
2.1.2	Arten von XML-Signaturen . . . . .	7
2.2	XML-Encryption . . . . .	7
2.2.1	XML-Encryption - Verschlüsselungsarten . . . . .	7
2.2.2	Komponenten der XML-Encryption . . . . .	8
2.3	SAML . . . . .	9
2.3.1	Assertions . . . . .	10
2.3.2	Protokoll . . . . .	12
2.3.3	Bindings . . . . .	12
<b>3</b>	<b>WS-Security in SOAP</b>	<b>13</b>
3.1	Security Tokens . . . . .	13
3.1.1	UsernameToken . . . . .	13
3.1.2	BinarySecurityToken . . . . .	14
3.1.3	XMLToken . . . . .	14
3.2	XML Verschlüsselung & Signatur . . . . .	14
3.2.1	'XML Encryption' . . . . .	15
3.2.2	'XML Signature' . . . . .	15
<b>4</b>	<b>WS-Policy</b>	<b>16</b>
<b>5</b>	<b>WS-Security Implementierung</b>	<b>17</b>
5.1	Apache WSS4J . . . . .	17
5.1.1	Allgemeines . . . . .	17
5.1.2	Umsetzung des Beispielszenario . . . . .	20
<b>6</b>	<b>Resüme und Ausblick</b>	<b>22</b>

# 1 Einleitung

Diese Arbeit befasst sich mit Web Service Security. Bevor wir jedoch auf die einzelnen Security-Features eingehen, soll an dieser Stelle eine kurze Einführung zu Web Services gegeben werden. Es soll geklärt werden was ein Web Service eigentlich ist und welche Sicherheitslücken mit Hilfe von Web Service Security abgedeckt werden sollen beziehungsweise abgedeckt werden können.

'A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.' [1]

Das obige Zitat beschreibt sehr gut die grundsätzliche Funktion. Web Services ermöglichen also eine Maschine-zu-Maschine Kommunikation, bei der XML-Nachrichten ausgetauscht werden. Um XML-Nachrichten von einer Maschine auf die andere zu übertragen, bedienen sich Web Services dem sogenannten Simple Object Access Protocol (SOAP) das auf einem Transportprotokoll wie beispielsweise HTTP aufsetzt. Die Tatsache, dass Nachrichten über ein unsicheres Medium übertragen werden birgt natürlich viele Sicherheitslücken. Beispielsweise könnte ein Angreifer eine SOAP-Nachrichten während des Transportes abfangen, verändern und erst im Anschluss an den ursprünglichen Empfänger weiterleiten. Man kann sich vorstellen, welche Schäden in manchen Fällen durch solch eine Attacke verursacht werden können.

Die Aufgabe von WS-Security ist es, für die Sicherheit von SOAP-Nachrichten zu sorgen. Im speziellen soll WS-Security dafür sorgen, dass SOAP-Nachrichten vor unbemerkten Veränderungen geschützt werden, dass Nachrichten nur von den Personen gelesen werden können für die sie auch bestimmt sind und dass einer Nachricht die eindeutige Identität des Senders zugeordnet werden kann.

Für den Schutz vor unbemerkter Veränderung einer Nachricht wird XML-Signature verwendet, dass die Signierung von Nachrichten beziehungsweise Teilen von Nachrichten realisiert. XML-Encryption ermöglicht die Verschlüsselung von Nachrichten, womit ein unbefugtes Lesen verhindert werden kann. Um sicherzustellen, dass der angegebene Sender einer Nachricht auch der wahre Sender dieser ist, setzt WS-Security auf die Security Assertion Markup Language (SAML), die die eindeutige Zuordnung einer Identität zur Nachricht ermöglicht.

## 2 WS-Security Basis-Konzepte

In den folgenden zwei Kapiteln werden zwei Schlüssel-Sicherheitskonzepte für XML-Nachrichten, im Zusammenhang mit Web-Services vorgestellt. Diese beiden Konzepte sollen folgende Sicherheitsanforderungen [6] sicherstellen:

- **Integrität (integrity)**  
XML-Nachrichten sollen vor beabsichtigter, oder aber auch zufälliger Manipulation geschützt werden. Das heißt, es soll gewährleistet werden, dass nur erlaubte und beabsichtigte Veränderungen an gespeicherten, verarbeiteten oder übertragenen Nachrichten zulässig sind.
- **Verbindlichkeit (non-repudiation)**  
Es soll eine Möglichkeit zur Anerkennung der Übermittlung (Empfängernachweis) und zur Anerkennung des Ursprungs (Urhebernachweis) der Information geschaffen werden. Das heißt, es soll ermöglicht werden, dass ein Nachrichtenaustausch nicht verleugnet werden kann.
- **Vertraulichkeit (confidentiality)**  
XML-Nachrichten sollen vor unberechtigtem Einblick durch Dritte geschützt werden. Es soll sichergestellt werden, dass nur der wahre Empfänger die Nachricht lesen kann.

Um die oben genannten Anforderungen zu erfüllen werden im WS-Security Standard zwei Strategien - XML Signature und XML Encryption - zur Anwendung gebracht, die im folgenden näher besprochen werden.

Zum Abschluß werden wir einen Blick auf SAML (Security Assertion Markup Language) [15] werfen. SAML bietet die Möglichkeit 'Vertrauenserklärungen' (trust assertions) eines Individuums (z.B.: ein Person oder ein Programm) zu erstellen und zu senden bzw. darauf zu antworten. Dies ist unumgänglich um eine verlässliche Authentifikation oder Autorisierung eines Web Services sicherzustellen, und somit ist SAML ein integraler Bestandteil von WS-Security.

### 2.1 XML-Signature

XML-Signature ist ein W3C XML-Security-Standard [8] mit dem in erster Linie Integrität des Nachrichtenaustausches gewährleistet werden soll. XML-Signaturen bedienen sich dazu dem Prinzip digitaler Signaturen. Die Grundidee dabei ist das Anbringen einer digitalen Unterschrift an ein XML-Dokument. Das heißt, die XML-Nachricht wird durch die Signatur ergänzt, jedoch nicht verändert. Mit Hilfe dieser digitalen Signatur, kann eine XML-Nachricht auf Veränderung überprüft werden.

Bevor XML-Signaturen im Detail besprochen werden, sollen an dieser Stelle noch einige Grund-Anforderungen [10], die die Entwicklung von XML-Signaturen geprägt haben, vorgestellt werden:

- XML-Signaturen müssen auf alle Ressourcen, die mit einer URI adressiert werden können, anwendbar sein. Dies inkludiert auch nicht XML-Inhalte wie Bilder, HTML und der gleichen.

- XML-Signaturen müssen auf Teile eines Dokumentes oder auf das gesamte Dokument anwendbar sein. Dies ist besonders wichtig, wenn ein XML-Dokument von mehreren Personen bearbeitet wird. Ist dies der Fall, wird ermöglicht, dass jede Partei nur den Teil, der von ihnen bearbeitet wurde, digital signiert. Weiters ermöglicht dieses Feature, dass Teile eines Dokumentes nicht signiert werden und somit die Möglichkeit offen gelassen wird, diese Teile weiter zu bearbeiten.
- XML-Signaturen müssen beliebige kryptographische Signaturmethoden unterstützen

Natürlich existieren noch jede Menge weitere Anforderungen an die Spezifikation von XML-Signaturen. Die oben angeführten sollen nur kurz verdeutlichen, dass es sich bei XML-Signaturen um weit mehr als „nur“ digitale Signaturen handelt.

### 2.1.1 Komponenten einer XML-Signatur

In diesem Kapitel soll der Aufbau von XML-Signaturen erläutert werden. Listing 1 zeigt die Hauptkomponenten einer XML-Signatur.

```

1 <Signature>
2   <SignedInfo>
3     ( CanonicalizationMethod )
4     ( SignatureMethod )
5     (<Reference URI=? >
6       ( Transforms )?
7       ( DigestMethod )
8       ( DigestValue )
9     </Reference >)+
10  </SignedInfo>
11  ( SignatureValue )
12  ( KeyInfo )?
13  ( Object )*
14 </Signature>

```

Listing 1: Aufbau einer XML-Signatur [8]

Wie die obige Abbildung zeigt, besteht eine XML-Signatur im Wesentlichen aus vier Elementen, wobei die Elemente <KeyInfo> und <Object> optional sind. Im folgenden werden die einzelnen Elemente näher besprochen [8] [5]:

1. **<SignedInfo>**: Dieses Element beinhaltet alle Informationen über das zu Signierende. Es beinhaltet drei Kind-Elemente:
  - **CanonicalizationMethod**: Wenn man zwei Dokumente digital signiert und in einem Dokument nur ein einziges Bit verändert, wird der Digest (Hashwert) unterschiedlich sein. Bei der Verwendung unterschiedlicher Betriebssysteme oder XML-Parser kann es vorkommen, dass es kleine Unterschiede in der Verarbeitung von XML-Dokumenten gibt und dies würde unterschiedliche Hashwerte mit sich bringen. Beispielsweise unterscheidet sich, eine auf einem Windows System erstellte Datei, mit einer selben auf einem Unix System erstellten, da beide Betriebssysteme verschiedene ASCII-Zeichen für ein Zeilenende verwenden. Das

CanonicalizationMethod-Element ist dafür verantwortlich, XML-Dokumente zu normalisieren, so dass zwei äquivalente XML-Dokumente, ungeachtet ihrer physikalischen Unterschiede, Bit für Bit gleich sind.

- **SignatureMethod:** Dieses Element benennt den Algorithmus, der für die Signierung und Überprüfung der digitalen Signatur verwendet wird. Es werden zwei Algorithmen benötigt: Einerseits der DSA mit SHA1. Dies ist ein public key Verfahren, das heißt es wird mit einem privaten Schlüssel signiert und im Anschluss mit dem dazugehörigen öffentlichen Schlüssel validiert. Zusätzlich zum DSA mit SHA1 kommt ein Private-key Hashverfahren, das sogenannte HMAC mit SHA1 zum Einsatz.

Wie bereits erwähnt verfolgen digitale Signaturen das Ziel Integrität und Identität zu gewährleisten. Private-key-Verfahren eignen sich jedoch nicht zum Validieren der Identität. Das Ziel der Entwickler war es jedoch, einen schnellen Algorithmus, für das Überprüfen der Integrität zur Verfügung zu stellen. Optional kann anstatt des HMAC-SHA1 der public-key Algorithmus RSA mit SHA1 verwendet werden, sollte die Überprüfung der Identität ebenfalls erwünscht sein.

- **Reference:** Dieses Element beinhaltet in Form einer URI (Uniform Resource Identifier) die Ressource beziehungsweise Ressourcen die signiert werden. Das Referenz-Element muss zumindest eine Ressource beinhalten (da zumindest etwas signiert werden muss). An dieser Stelle sei angemerkt, dass URIs im Zusammenhang mit XML weit Ausdrucksstärker, als im Zusammenhang mit HTML sind. Eine URI kann beispielsweise auf ein externes XML-Dokument, auf das Root-Element im aktuellen Dokument, auf ein beliebes Element in einem externen Element, oder ähnliches zeigen.
  - **Transform:** Ist ein optionales Kindelement von Reference, das eine geordnete Liste von Transform-Elementen beinhaltet. Beispielsweise könnte ein XML-Dokument schon mehrmals signiert worden sein und daraus folgt, dass der Signierer nicht das ursprüngliche Dokument signiert, sondern ein Dokument, das bereits eine beziehungsweise mehrere Signierungen durchlaufen hat.
  - **DigestMethod:** Ist ebenfalls ein Kindelement von Reference und benennt den Algorithmus, der für die Berechnung des Digest der Reference URI beziehungsweise URIs sowie allen Transforms verwendet wurde.
  - **DigestValue:** Beinhaltet den eigentlichen Wert des Digest.

2. **<SignatureValue>:** dieses Element repräsentiert die Signatur an sich.

3. **<Object>:** Wie bereits erwähnt handelt es sich beim Object-Element um ein optionales Element. Es wird meist verwendet wenn es sich um eine Enveloping-Signature handelt, das heißt, dass der zu signierende Teil selbst Teil der Signatur ist (Details zu Enveloping-Signatures werden im nächsten Kapitel behandelt). Das Object-Element kann beispielsweise XML-Fragmente, Bilder oder ähnliches enthalten.

4. **<KeyInfo>:** Auch dieses Element ist ein optionales Element. Es kann dazu verwendet werden, um Informationen über den Schlüssel zur Validierung der XML-Signatur zur Verfügung zu stellen. Beispielsweise kann es den Schlüssel selbst, den Schlüssel inner-

halb eines digitalen Zertifikates, Informationen die zum Schlüssel führen oder ähnliches beinhalten.

### 2.1.2 Arten von XML-Signaturen

Wie bereits erwähnt können XML-Signaturen auf XML-Dokumente, auf Teile von XML-Dokumenten oder aber auch auf externe Ressourcen angewandt werden. Die zu signierenden Teile werden mit Hilfe von URIs referenziert. XML-Signaturen können in drei verschiedene Arten [5] unterteilt werden: Enveloping-, Enveloped- und Detached-Signatures. Im folgenden werden die Unterschiede dieser erläutert:

- **Enveloping-Signatures**

Bei einer Enveloping-Signature, ist der zu signierende Teil selbst Teil der Signatur. Eine Enveloping-Signature ist daran zu erkennen, dass die URI im Reference-Element auf das Object-Element innerhalb der Signatur zeigt.

- **Enveloped-Signatures**

Bei einer Enveloped-Signature, ist die Signatur selbst, ein Kindelement eines anderen Elementes. Diese Art der Signatur ist daran zu erkennen, dass die URI im Reference-Element, auf ein Element, dass das Vater-Element des Signature-Elementes ist, zeigt.

- **Detached-Signatures**

Eine Detached-Signatur referenziert eine XML-Nachricht, einen Teil einer XML-Nachricht beziehungsweise ein beliebiges Objekt, dass über eine URI referenziert werden kann, außerhalb der Signatur. Diese Art der Signatur erkennt man daran, dass die URI weder ein Kind- noch ein Vater-Element des Signatur-Elementes referenziert.

## 2.2 XML-Encryption

Wie bereits erwähnt, setzen Web-Services auf SOAP um Nachrichten zu transportieren. Das Ziel von XML-Encryption [13] ist, alles beziehungsweise Teile einer SOAP-Nachricht geheim zu halten. XML-Encryption ist ein weiterer XML-Security-Standard des W3C, der der Entwicklung von XML-Signature folgte. Es ermöglicht gesamte XML-Dokumente, beziehungsweise Teile von XML-Dokumenten zu verschlüsseln und dies, wenn gewünscht, mit mehreren verschiedenen Schlüsseln. Die Verwendung mehrerer verschiedener Schlüssel ermöglicht, dass unterschiedliche Empfänger einer Nachricht, jeweils nur den für ihn bestimmten Teil einer Nachricht entschlüsseln können und Teile, die für diesen Empfänger nicht bestimmt sind, verborgen bleiben.

### 2.2.1 XML-Encryption - Verschlüsselungsarten

XML-Encryption bietet drei verschiedene Möglichkeiten XML-Dokument beziehungsweise Teile von XML-Dokumenten zu verschlüsseln [6]:

- **symmetrische Verschlüsselung:** Bei der symmetrischen Verschlüsselung sind Sender und Empfänger der Nachricht im Besitz des gleichen, für alle anderen geheimen, Schlüssels. Der Sender verschlüsselt die gewünschten Teile oder die gesamte Nachricht mit dem geheimen Schlüssel, sendet die Nachricht an den Empfänger und dieser kann

daraufhin die Nachricht unter Verwendung des gleichen Schlüssels entschlüsseln. Beispiel für solch einen Algorithmus: Data Encryption Standard (DES), 3DES und weitere.

- **asymmetrische Verschlüsselung:** Bei asymmetrischen Verfahren existieren zwei, mathematisch miteinander in Bezug stehende, Schlüssel (öffentlicher und privater Schlüssel). Der Sender verschlüsselt die gewünschten Teile oder die gesamte Nachricht mit dem öffentlichen Schlüssel des Empfängers und nur dieser kann mit Hilfe seines privaten Schlüssels die Nachricht entschlüsseln. Beispiel für solch einen Algorithmus ist der Advanced Encryption Standard (AES).
- **hybride Verschlüsselungsverfahren:** Sind Verfahren, die symmetrische und asymmetrische Verfahren kombinieren. Symmetrische Verfahren werden zur Verschlüsselung der Daten verwendet, jedoch geschieht der Austausch des dazu benötigten Schlüssels mit asymmetrischen Verfahren.

Details über die Wahl des Verschlüsselungsverfahrens in XML-Encryption folgen im nächsten Kapitel.

## 2.2.2 Komponenten der XML-Encryption

In diesem Kapitel soll der Aufbau von XML-Encryption [13] [10] [12] erläutert werden. Listing 2 zeigt die Hauptkomponenten der XML-Encryption.

```
1 <EncryptedData Id? Type? MimeType? Encoding?>
2   <EncryptionMethod/>?
3   <ds:KeyInfo>
4     <EncryptedKey>?
5     <AgreementMethod>?
6     <ds:KeyName>?
7     <ds:RetrievalMethod>?
8     <ds:*>?
9   </ds:KeyInfo>?
10  <CipherData>
11    <CipherValue>?
12    <CipherReference URI?>?
13  </CipherData>
14  <EncryptionProperties>?
15 </EncryptedData>
```

Listing 2: Aufbau der XML-Encryption [13]

Das EncryptionData-Element beinhaltet die Daten die verschlüsselt sind beziehungsweise eine Referenz auf eine externe Ressource die verschlüsselt ist. EncryptedData besitzt vier Kind-Elemente: EncryptionMethod, CipherData und die zwei optionalen Elemente KeyInfo und EncryptionProperties) die nun näher betrachtet werden:

- **<EncryptionMethod>:** Dieses Element beinhaltet eine Referenz auf den Algorithmus der für die Verschlüsselung verwendet wird. Die Spezifikation von XML-Encryption stellt vier Verschlüsselungsalgorithmen zur Auswahl: den Triple-DES, den Advanced Encryption Standard (AES) mit einer Schlüssellänge von 128 Bit, den AES 256, sowie

den AES 192. Das heißt, das EncryptionMethod-Element beinhaltet beispielsweise die URI: „http://www.w3.org/2001/04/xmlenc#256-cbc“ (bei Wahl des AES 256).

- **<KeyInfo>**: Wie bereits in XML-Signature bietet das KeyInfo-Element die Möglichkeit, einen Schlüssel, einen Hinweis auf den Schlüssel oder ähnliches zu hinterlegen. Es ist davon abzuraten den symmetrischen Schlüssel zu hinterlegen, da mit Hilfe dessen, die Nachricht von unberechtigten Personen entschlüsselt werden kann. Eine Alternative dazu ist es, einen mit dem öffentlichen Schlüssel des Empfängers verschlüsselten symmetrischen Schlüssel bereitzustellen oder aber auch das Feld leer zu lassen (was auch Möglich ist, da es sich bei KeyInfo um ein optionales Element handelt).
- **<CipherData>**: Dieses Element besitzt zwei Kind-Elemente und beinhaltet entweder die verschlüsselte Information oder eine URI, die eine verschlüsselte Information referenziert. Listing 3 zeigt beispielsweise ein CipherData-Element mit verschlüsselter Information:

```
1 ...
2 <CipherData>
3   <CipherValue>B4567890567CD546</CipherValue>
4 </CipherData>
5 ...
```

Listing 3: CipherData mit verschlüsselter Information

- **<EncryptionProperties>**: Dieses Element kann zusätzliche Informationen über die Generierung der verschlüsselten Daten beziehungsweise über den verwendeten Schlüssel enthalten. Beispielsweise kann in EncryptionProperties ein TimeStamp, die Hardware die während der Verschlüsselung benutzt wurde oder ähnliches beinhalten. Dieses Element ist ein zusätzlicher, optionaler Sicherheitsfaktor.

## 2.3 SAML

‘SAML is an XML-based framework for request/response exchanges of authentication and authorization.’ [9]

Wie das Zitat oberhalb bereits verdeutlicht besteht die Hauptaufgabe von SAML portable Identitäten bzw. portables Vertrauen mit Hilfe von Web Services einzusetzen und somit die Authentifikation und Autorisation von Entitäten aus anderen Vertrauensdomänen (‘trust domains’) (z.B.: einer anderen Organisation oder einer anderen Applikation) zu ermöglichen. Speziell der Bereich des B2B (Business to Business) verlangt, durch die immer stärkere Vernetzung verschiedener Organisationen und deren korrespondierenden Vertrauensdomänen, solch eine Methode der portablen Identität.

Gerade die Web Service Technologie ist von ihrer Grundidee darauf ausgelegt verschiedenste Vertrauensdomänen zu durchqueren. Eine portable Möglichkeit der Identifikation, Authentifikation und Autorisation ist somit unerlässlich. Abbildung 1 illustriert das Beispiel eines Buchhändlers, der mit Hilfe von Web Services mit anderen solcher Domänen miteinander kommunizieren.

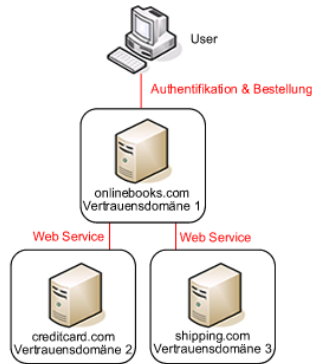


Abbildung 1: Beispiel Vertrauensdomänen

Die Relevanz von SAML lässt sich auch an der Tatsache festmachen dass nach [7] SAML der erste und bisweilen einzige Industriestandard für sichere E-Commerce Transaktionen ist. Dieser Einschätzung folgend werden sich die weiteren Entwicklungen im Bereich der Web Service Sicherheit sehr stark an diesem Standard orientieren bzw. diesen benutzen.

Die folgenden Kapitel gehen näher auf die drei XML basierten Hauptmechanismen von SAML ein: Assertions, Protokolle, Bindings.

Bevor wir uns weiter mit dem Konzept von SAML auseinander setzen, möchte ich zuerst einige Definitionen rund um den Begriff Identität erläutern. Diese wurden großteils aus [5] übernommen bzw. übersetzt:

- Der Begriff **Identität** bezieht sich auf eine Entität einer Organisation.
- Der Begriff **Subjekt** bezieht sich auf eine Entität die vorgibt eine gewisse Identität zu besitzen.
- Die bei Erstellung der Identität generierten Rechte werden unter dem Begriff **Legitimation** zusammengefasst.
- Rechte die in Anspruch genommen werden um eine spezifische Aktion durchzuführen werden unter dem Begriff **Geltendmachung** (im Weiteren wird der Begriff 'Assertions' benutzt) erfasst.
- Der Begriff **Authentifikation** ist eine Geltendmachung bezüglich der Identität der Entität.
- Der Begriff **Authorisation** ist eine Geltendmachung einer Identität bezüglich des Rechts eine bestimmte Aktion durchzuführen.

### 2.3.1 Assertions

SAML Assertions folgen einem vordefinierten XML Schema, das Standardelemente wie Name, Ausstellungsdatum und Gültigkeitszeitraum sowie optionale Elemente wie zusätzliche Bedingungen spezifiziert. Tabelle 1 und Tabelle 2 listen alle definierten Elemente und Attribute des SAML Assertions XML Elements auf.

Element	Beschreibung
<Conditions>	Bedingungen die auf die Assertion zutreffen.
<Audience>	URI der gewünschten Empfänger
<Advice>	zusätzliche Informationen
<ds:Signature>	XML Signatur zur Authentifikation

Tabelle 1: Assertion XML Elemente

Attribut	Beschreibung
Issuer	Name des Antragstellers
IssuerInstant	Zeit der Antragstellung
AssertionID	UID der Assertion
Subject \& Name	Vertrauensdomäne der Assertion
SubjectConfirmation	Authentifikationsprotokoll und zugehörige Daten

Tabelle 2: Assertion XML Attribute

Es wird weiters zwischen drei verschiedenen Assertions unterschieden. Eine **Authentifikations-Assertion** übernimmt den Authentifikationsversuch eines Subjekts zu einer Identität. Hier wird ebenfalls die Authentifikationsmethode spezifiziert. SAML unterstützt eine lange Liste solcher Methoden, darunter auch die gängigsten Authentifikationsmechanismen wie die 'triviale' Passwortmethode, Kerberos, SSL (Secure Sockets Layer) [17] Zertifikate, PGP (Pretty Good Privacy) [14] oder natürlich auch XML Signatur. Kurz zusammengefasst erfasst die Authentifikations Assertion, bei erfolgreicher Authentifikation, dass Subjekt S mit Hilfe der Methode M zur Zeit T identifiziert wurde.

Beim Versuch eines Subjekt S auf eine Ressource R zuzugreifen wird eine **Autorisations-Assertion** benötigt. Listing 4 zeigt die wichtigsten Teile einer Assertion bezüglich dem erlaubten Zugriff auf eine spezifische Ressource.

```

1 <saml: AuthorizationStatement
2   Decision="Permit"
3   Ressource="http://onlinebooks.com/buy.cgi">
4   <saml: Subject >... </saml: Subject
5   <saml: Action>Execute </saml: Action>
6 </saml: AuthorizationStatement >
```

Listing 4: Autorisations-Assertion

Eine **Attribut-Assertion** wiederum verknüpft spezifische Attribute A, B, C, usw. mit einem bestimmten Subjekt. Typische Attribute für unser Online Buchhandlung wären 'Kreditlimit' oder 'Kundennummer'.

### 2.3.2 Protokoll

Zusätzlich zu dem Assertion XML Schema definiert SAML ein weiteres eigenes Schema das Anfrage und Antwort Protokoll, mit dessen Hilfe die verschiedenen Assertions ausgetauscht werden.

Mit Hilfe einer **Authentifikationsanfrage** wird gehandhabt welche Authentifikations-Assertions für das Subjekt vorliegen bzw. verfügbar sind, diese werden wiederum mit der korrespondierenden Antwort zurückgesendet. Listing 5 zeigt die wichtigsten Teile einer solchen Anfrage.

```
1 <saml:Request ... IssueInstant="2005-12-10...">
2   <saml:ResponseWith>
3     saml:AuthenticationStatement
4   </saml:ResponseWith>
5   <saml:pAuthenticationQuery>
6     <saml:Subject>
7       <saml:NameIdentifier
8         SecurityDomain="onlinebooks.com"
9         Name="user123" />
10    </saml:Subject>
11  </saml:pAuthenticationQuery>
12 </saml:Request>
```

Listing 5: Authentifikationsanfrage

Listing 6 enthält die wichtigsten Bestandteile der korrespondierenden Antwort.

```
1 <saml:Response ... IssueInstant="2005-12-10...">
2   <saml:Status>
3     <saml:StatusCode Value="Success" />
4   </saml:Status>
5   <saml:Assertion>
6     ...
7   </saml:Assertion>
8 </saml:Response>
```

Listing 6: Authentifikationsantwort

Das Anfrage und Antwort Protokolle für Attribut-Assertions (Welche Attribute gehören zum Subjekt) und für Autorisations-Assertions (Auf welche Ressourcen darf das Subjekt zugreifen?) sind nach dem gleichen Schema aufgebaut und werden hier nicht mehr näher behandelt.

### 2.3.3 Bindings

SAML Bindings werden dazu verwendet um SAML Protokollanfragen bzw. -antworten auf das eingesetzte Kommunikationsprotokoll abzubilden. Ein gängiges Binding wäre das Einfügen einer SAML Anfrage in eine SOAP Nachricht die mit Hilfe von HTTP versandt wird. Die Richtigkeit, Integrität und Vertraulichkeit der Authentifikation wird hierbei meist über SSL sichergestellt.

## 3 WS-Security in SOAP

In diesem Kapitel wollen wir im Speziellen auf die praktische Anwendung des WS-Security Konzepts mit SOAP (Simple Object Access Protocol) [16] eingehen. Kurz gesagt: WS-Security ist das Pendant zu XML-Security. Sprich es sorgt für die Sicherheit von SOAP Nachrichten. Folgend die wichtigsten Mechanismen von WS-Security (siehe Kapitel 2):

1. digitale Signierung von Teilen oder vollständigen SOAP Nachrichten (mit Hilfe von XML Signaturen)
2. die Verschlüsselung von eben diesen Nachrichten (mit Hilfe von XML Verschlüsselung)
3. 'Vertrauenserklärungen' um eine Nachricht eindeutig einer Identität zuzuordnen, und somit die Authentizität einer Nachricht zu wahren (z.B.: mit Hilfe der WS-Security Komponente SAML)

Grundlegend ist hier eine Unterscheidung zwischen der Sicherheit auf der Transportebene (z.B.: mit SSL), deren Aufgabe es ist eine sichere Verbindung zwischen zwei Punkten herzustellen, und jener auf der Nachrichtebene (z.B.: WS-Security mit SOAP), die die Sicherheit der Nachricht unabhängig vom momentanen Speicherort gewährleisten soll, zu treffen. Wir werden uns hier, wie oben bereits erwähnt, nur mit der Ebene der Nachrichtensicherheit befassen. WS-Security definiert einen 'SOAP Security Header' der für die drei verschiedenen Sicherheitsartefakte Platz bieten:

1. **Security Tokens**: Informationen die zur Authentifikation und Autorisation genutzt werden (Bsp.: Benutzername/Passwort).
2. **'XML Encryption'**: Ein 'encryptedKey' Element und die korrespondierende Referenzliste (siehe Kapitel 2.2)
3. **'XML Signature'**: Die Signatur und die korrespondierende Referenzliste (siehe Kapitel 2.1)

### 3.1 Security Tokens

WS-Security [4] spezifiziert kein allgemeines Security Token Element sondern drei granulare Token Arten. Der UsernameToken und der BinarySecurityToken bedienen sich eines umschließenden Elements um ihre Informationen zu speichern (z.B.: Benutzername und Passwort). Der XMLToken kann je nach Belieben gestaltet werden (z.B.: eine SAML Assertion, siehe Kapitel 2.3). Durch die Verwendung von Security Tokens im 'SOAP-Header' ermöglicht man das gleichzeitige Mitsenden der Daten für die Transaktion, da diese im 'SOAP-Body' abgelegt werden. Dies ermöglicht ein flexibles und ressourcensparendes Einbinden von Sicherheit in Web Services.

Im Folgenden werden wir näher auf diese drei unterschiedlichen Token Arten eingehen.

#### 3.1.1 UsernameToken

Mit Hilfe dieser Token Art wird der einfache Benutzername/Passwort Mechanismus ermöglicht. Hierbei sollte man beachten das Passwort auf eine sichere Art zu behandeln, z.B. durch Benutzung eines Hash-Werts des Passworts.

### 3.1.2 BinarySecurityToken

Zwei verschiedene Verwendungen des Binary-Tokens werden im WS-Security Standard spezifiziert: die eines X.509 Version 3 Zertifikats und die eines Kerberos Tickets. Ein X.509 Zertifikat beinhaltet einen öffentlichen Schlüssel einer asymmetrischen Verschlüsselungsmethode (z.B.: RSA) und wird von einer vertrauenswürdigen dritten Partei signiert. Listing 7 zeigt die wichtigsten Bestandteile eines solchen Tokens.

```
1 <wsse:BinarySecurityToken ID="bspToken">
2   Value="wsse:X509v3"
3   EncodingType="wsse:Base64Binary">
4     ... BinaryData ...
5 </wsse:BinarySecurityToken>
```

Listing 7: BinarySecurityToken

Die Authentifikation und Autorisation ist ebenfalls mit Hilfe von Kerberos möglich. Zwei verschiedene Arten von Tickets (binäre Daten) können hier übermittelt werden, ein Ticket Granting Ticket (wird meist zur Authentifikation benutzt) oder ein Service Ticket (wird meist zur Autorisation benutzt).

### 3.1.3 XMLToken

Ein XML-Token kann frei nach einem gewissen XML Schema gestaltet sein. Die gängigste Art eines solchen Tokens ist das SAML-Token (siehe Kapitel 2.3). Ich möchte mich hier auf die Beschreibung der Verwendung von SAML Tokens beschränken, obwohl noch andere XML-Tokens spezifiziert sind, wie XCBF-Tokens (XML Common biometric Format) oder XrML-Tokens (eXtensible Rights Markup Language).

SAML Assertions haben ein ähnliches Problem wie Zertifikate. Der Empfänger muss die Möglichkeit besitzen den Sender eindeutig zu identifizieren. Sprich das Subject einer SAML Assertion muss eindeutig bestimmbar sein. Hierfür werden zwei Methoden bereitgestellt:

1. **'holder-of-key' Methode:** Zusätzlich zu der Assertion wird eine XML Signatur mit geschickt. Diese beinhaltet einen `KeyInfo` Block der wiederum auf die entsprechende Assertion verweist. Die Assertion beinhaltet bestimmte Information (z.B.: ein X.509 Zertifikat) das die Signatur mit Hilfe des 'ConfirmationMethod' Elements verifiziert.
2. **'sender-vouches' Methode:** Bei dieser Methode wird die Assertion, ähnlich wie bei Zertifikaten, von einer dritten vertrauenswürdigen Partei signiert.

Zurzeit wird die Verwendung von SAML-Tokens im 'SOAP-Header' als die beste und flexibelste Form der Einbindung von Authentifikation und Autorisation in Web Services angesehen [5] [9] [7].

## 3.2 XML Verschlüsselung & Signatur

Dieser Abschnitt wird sich mit der Verwendung der Techniken 'XML Encryption' (siehe Kapitel 2.2) und 'XML Signature' (siehe Kapitel 2.1) in SOAP Nachrichten beschäftigen.

### 3.2.1 'XML Encryption'

'XML Encryption' sorgt für die Vertraulichkeit von SOAP Nachrichten indem sie mit Hilfe symmetrischer, asymmetrischer oder hybrider Verschlüsselungstechniken die Nachricht oder Teile dieser verschlüsselt und somit nur dem Empfänger zugänglich macht. Zusätzlich zu dem verschlüsselten Teil der Nachricht (EncryptedData Block) wird im 'SOAP Security Header' eine Element des Typs ReferenceList angegeben. Diese Liste zeigt auf die verschlüsselten Teile der SOAP Nachricht. Listing 8 zeigt die wichtigsten Teile solch einer Nachricht, wobei hier die gesamte SOAP Nachricht verschlüsselt wird. Es besteht hier natürlich auch die Möglichkeit verschiedene Teile der Nachricht zu verschlüsseln und andere im Klartext zu belassen. In diesem Beispiel wird eine hybride Verschlüsselungstechnik verwendet, sprich der symmetrische Schlüssel, mit dessen Hilfe man die Nachricht verschlüsselt, wird mit Hilfe des öffentlichen Schlüssel des Empfängers verschlüsselt und mit der Nachricht gesendet.

```
1 <S:Header>
2   <wsse:Security>
3     <xenc:EncryptedKey>
4       <xenc:EncryptionMethod Algorithm="..."/>
5       <ds:KeyInfo>
6         ...
7       </ds:KeyInfo>
8     <xenc:CipherData>
9       <xenc:CipherValue >... </xenc:CipherValue>
10    </xenc:CipherData>
11    <xenc:ReferenceList>
12      <xenc:DataReference URI="body"/>
13    </xenc:ReferenceList>
14  </xenc:EncryptedKey>
15 </wsse:Security>
16 </S:Header>
17 <S:Body>
18   <xenc:EncryptedData Id="body">
19     <xenc:CipherData>
20       <xenc:CipherValue >... </xenc:CipherValue>
21     </xenc:CipherData>
22   </xenc:EncryptedData>
23 </S:Body>
```

Listing 8: XMLEncryption in SOAP Nachrichten [5]

### 3.2.2 'XML Signature'

'XML Signature' sorgt für die Integrität von SOAP Nachrichten indem sie die Nachricht selbst oder die im 'SOAP Security Header' befindlichen 'Security Tokens' (z.B.: ein X.509 Zertifikat) signiert. Da XML Signaturen bereits im Kapitel 2.1 behandelt wurden und das Einbinden dieser in eine SOAP Nachricht sich im Prinzip darauf beläuft diese in den 'SOAP Security Header' einzufügen, wird hier auf Beispiele oder zusätzliche Erklärungen verzichtet.

## 4 WS-Policy

Dieses Kapitel wird sich kurz mit 'WS-Policy' befassen und anhand eines kurzen Beispiels die Grundidee und Funktionsweise erläutern.

Das grundlegende Ziel von 'WS-Policy' ist es Mechanismen bereitzustellen, die es uns ermöglichen spezifische Richtlinien bzw. Regelsets (Policies) für 'Web Services' zu entwickeln. Hierzu werden spezielle Regel-Ausdrücke erstellt (Policy Expressions) die Domain-spezifische Informationen enthalten. Weiters werden Konstrukte angeboten die die Kombination der vorher besprochenen Ausdrücke regeln. Listing 9 zeigt ein Beispiel einer 'Authentifikations-Policy'. Es werden hier verschiedene Möglichkeiten zur Authentifikation geboten, wobei exakt eine dieser bei der Authentifikation gewählt werden muss.

```
1 <wsp:Policy>
2   <wsp:ExactlyOne>
3     <wsse:SecurityToken>
4       <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
5     </wsse:SecurityToken>
6     <wsse:SecurityToken>
7       <wsse:TokenType>wsse:X509v3</wsse:TokenType>
8     </wsse:SecurityToken>
9   </wsp:ExactlyOne>
10 </wsp:Policy>
```

Listing 9: WS-Policy Beispiel [11]

Für Informationen zur 'WS-Policy' (Notation, Terminologie, Model,...) empfehle ich [11].

## 5 WS-Security Implementierung

Dieses Kapitel soll einen Überblick über eine Implementierung des OASIS WS-Security Standards geben. Hierzu werden die Konzepte von WSS4J v.1.1 (einer WS-Security Implementierung von Apache) beleuchtet und hinterfragt.

### Beispiel Szenario

Hierzu soll unter anderem, mit Hilfe dieser WS-Security Implementierung, eine Lösung für folgendes Szenario aufgezeigt werden:

1. Ein Client soll die Möglichkeit haben ein Web Service aufzurufen, wobei einzelne XML Elemente des SOAP Body signiert und verschlüsselt sind
  - a) Dazu soll definiert werden welche Elemente signiert bzw. verschlüsselt werden
  - b) Weiters muss der Schlüssel und die Art der Signierung bzw. Verschlüsselung spezifiziert werden
2. Jener Server, der dieses Webservice hostet, soll die entsprechenden Teile des SOAP Request Body entschlüsseln und verifizieren können
3. Abschließend soll der Client einen, ebenfalls signierten und verschlüsselten, SOAP Response erhalten

Allgemein ist darauf hinzuweisen, dass der in weiterer Folge angeführte Code, aus den mitgelieferten Beispielapplikationen, der entsprechenden Implementierung entnommen wurde. Dieser wurde zusätzlich geringfügig adaptiert um dem Beispielszenario zu entsprechen. Weiters soll es nicht das Ziel dieser Arbeit sein, sämtliche Implementierungsdetails mit Hilfe von Codebeispielen aufzuzeigen, da dies den Rahmen dieser Arbeit sprengen würde. Es wird jedoch versucht die grundsätzliche Vorgangsweise für die Erstellung eines sicheren Web Services aufzuzeigen. Tiefer gehende Details und ausführlichen Beispielapplikationen sind in [3] zu finden.

### 5.1 Apache WSS4J

#### 5.1.1 Allgemeines

WSS4J stellt eine Java library dar, die für die Signierung, Verifizierung und Verschlüsselung von SOAP Nachrichten verwendet werden kann. Grundsätzlich ist sie in ihrer Verwendung nicht auf eine SOAP Engine beschränkt, jedoch bietet sie eine besondere Unterstützung für Axis basierte Web Services. Hierzu implementiert sie spezielle Handler, die in die Axis spezifischen Web Service Deskriptoren (wsdd) eingebunden werden können. [3].

Durch diese Handler wird eine Art WS-Security Layer in das, zu schützende, Web Service integriert, ohne dass der eigentliche Code des Web Services angepasst werden muss. Um dies zu ermöglichen, können mit Hilfe der beiden WSS4J Axis Handler „WSDoAllSender“ und „WSDoAllReceiver“ SOAP Requests mit entsprechendem WS-Security Teil, standardkonform verarbeitet werden und auch erzeugt werden.

Auffallend an diesem Lösungsansatz ist, dass all die dahinter liegenden Standards und

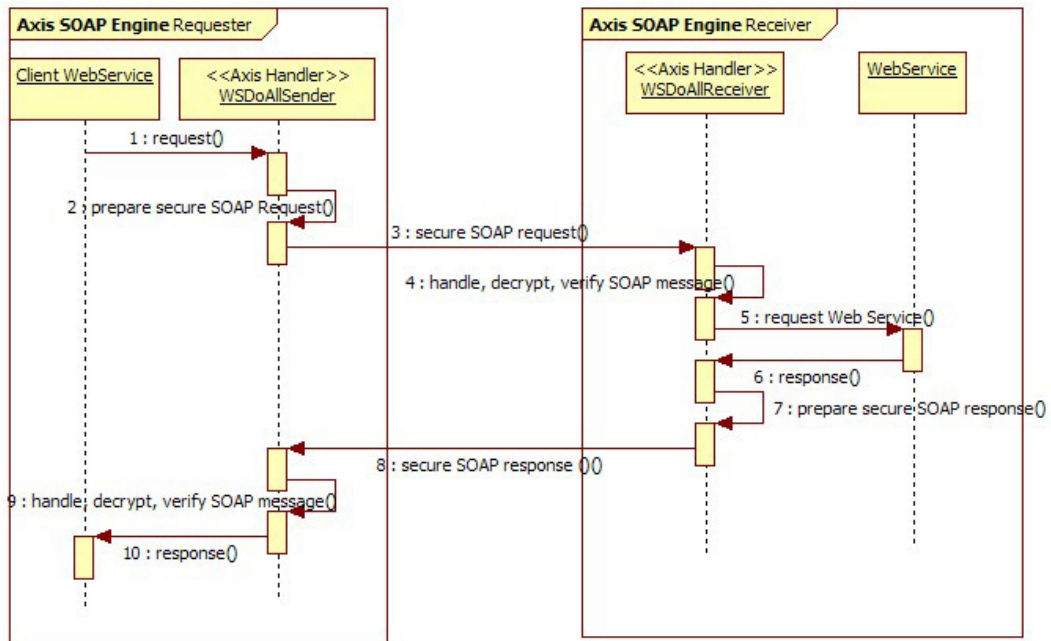


Abbildung 2: Webservice Interaktion mit WSS4J

Technologien, wie XML Signature und XML Encryption aber auch SAML, für den Benutzer völlig transparent sind. Deren Benutzung spiegelt sich einzig und allein in der Deklaration von XML Tags im Axis Web Service Deployment Descriptor (wsdd) wider (siehe Abschnitt 5.1.2). Dies ist unserer Meinung nach durchaus gut gelungen, da der Benutzer einer WS-Security Implementierung von der Flut an Standards der WS-\* Family, möglichst abgeschirmt werden sollte. Abbildung 2 stellt die Kommunikation zwischen einem Client und einem Webservice dar, die durch Nutzung von WSS4J eine sichere Kommunikation unterhalten.

Nachfolgend möchten wir kurz unsere Erfahrungen mit der praktischen Umsetzung des Beispiels erläutern. Zur Zeit gibt es bedauerlicherweise noch keine wirkliche Toolunterstützung in den, von uns eingesetzten, IDEs (Integrated Development Environment) Eclipse und NetBeans. Weiters steht noch sehr wenig Dokumentation zur Verfügung, deshalb mussten wir uns auf das Lesen der JavaDocs und das Testen der Beispielimplementierungen beschränken.

Wir setzten für die praktische Umsetzung folgende Komponenten ein

- Apache Tomcat in der Version 5.5.12 als Web Container
- Apache Axis in der Version 1.2.1 als SOAP Engine
- WSS4J in der Version 1.1 und WSS4J-otherJars (beinhaltet von WSS4J benötigte Libraries wie z.B.: XMLSecurity und Axis Libraries)
- Eclipse 3.1.1 und dessen integrierter Ant Runner

Sehr hilfreich fanden wir weiters die Ant Taskdefinitionen für Axis. Diese befinden sich in der „axis-tasks.properties“ Datei der Library „axis-ant-2.1.1.jar“, die mit Axis mitgeliefert wird. Um diese nutzen zu können muss jener Code aus Listing 10 in das entsprechende Ant Build Script eingebunden werden.

```
1 <taskdef resource="axis-tasks.properties">
2   <classpath refid="axis.classpath"/>
3 </taskdef>
4
5 <path id="axis.classpath">
6   <fileset dir="Ersetzung_durch_Pfad_zu_Axis_libs">
7     <include name="axis-ant-2.1.1.jar" />
8   </fileset>
9 </path>
```

Listing 10: Ant Taskdefinition für Axis

Anschließend stehen die Ant Tasks `<axis-java2wsdl>` (generiert ein WSDL aus Java Klassen), `<axis-wsdl2java>` (generiert Stub und Tier Klassen aus einem gegebenen WSDL File) und `<axis-admin xmlfile="wsdd Deployment Descriptor" />` (ermöglicht das deployen oder undeployen mit Hilfe der Admin Konsole von Axis) zur Verfügung

### **Grundsätzliche Vorgehensweise um ein WS-Security konformes Webservice mit Axis und WSS4J zu erstellen und zu deployen:**

- Erstellung eines Interfaces, das von `java.rmi.Remote` abgeleitet ist und die öffentlichen zugänglichen Methoden des Web Services deklariert.
- Implementierung einer Java Klasse, die dieses Interface implementiert und die Methoden des Interfaces entsprechend ausimplementiert
- Erstellung von Axis Deployment Descriptoren (wsdd) für das Web Service, die zusätzliche WSS4J Deklarationen beinhalten (siehe Abschnitt 5.1.2)
- Generierung eines WSDL Files mit Hilfe des Ant Tasks `<axis-java2wsdl>`. Dazu wird das, im ersten Schritt erstellte, Remote Interface übergeben.
- Generierung der entsprechenden „Stub“ Klassen (für Web Service Client) und „Tie“ (für Server) Klassen mit Hilfe des `<axis-wsdl2java>` Ant Tasks.
- Packen eines jar Files, das das WSDL File, die erzeugten Klassen des Servers und optional eine „Keystore“ Datei und eine `signature.property` Datei enthält
- Deployen des jar Files in die Axis SOAP Engine. Dies geschieht vorzugsweise mit Hilfe des Axis Ant Taks `<axis-admin>`. Zusätzlich muss der erstellte Axis Deployment Descriptors (wsdd) mitübergeben werden. Axis muss zuvor bereits in einem Applikationsserver mit entsprechender Servlet Engine, als Webapplikation deployed sein.
- Das Webservice kann nun mit Hilfe der erzeugten Stub Klassen und dem Client Deployment Descriptors angesprochen werden. Hierfür werden die SOAP Request Messages mit entsprechenden Security Header erzeugt und an den Server gesendet.

Um die Erzeugung der diversen SOAP Messages durch Axis nachvollziehen zu können, eignet sich hervorragend der SOAP Monitor von Axis. Dieser ist jedoch, aus Sicherheitsgründen, standardmäßig deaktiviert. Mit dessen Hilfe lässt sich erkennen welche SOAP Security Header die WSS4J Handler durch die Konfiguration in den Deployment Deskriptoren erzeugen.

### 5.1.2 Umsetzung des Beispielszenario

Zur Umsetzung des in Kapitel 5 angeführten Beispielszenario bedarf es in WSS4J grundsätzlich der Anpassung der Axis Deployment Deskriptoren für den Client (client\_deploy.wsdd) und für den Server (server\_deploy.wsdd). Listing 11 zeigt Ausschnitte des entsprechenden client\_deploy.wsdd Deployment Descriptor.

```
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2           xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3   <!--
4     define the service , using the WSDoAllSender security handler in request flow
5   -->
6   <service name="secureWebService">
7     <requestFlow>
8       <handler name="DoSecuritySender"
9         type="java.org.apache.ws.axis.security.WSDoAllSender" >
10        <parameter name="user" value="16c73ab6-b892-458f-abf5-2f875f74882e"/>
11        <parameter name="passwordCallbackClass"
12          value="org.apache.ws.axis.samples.wssec.doall.PWCallback"/>
13        <parameter name="action" value="Signature Encrypt"/>
14        <parameter name="signatureKeyIdentifier" value="DirectReference" />
15        <parameter name="signaturePropFile" value="crypto.properties" />
16        <parameter name="signatureParts"
17          value="{ Content } { http://example.org/paymentv2 } CreditCard ;
18              { Element } { } UserName" />
19        <parameter name="encryptionKeyIdentifier" value="X509KeyIdentifier" />
20        <parameter name="encryptionParts"
21          value="{ Element } { http://example.org/ } CreditCard ; { Element } { } UserName" />
22      </handler>
23    </requestFlow>
```

Listing 11: WSS4J Axis Client Deployment Descriptor

Nachfolgend werden die wesentlichen Elemente dieses Listings erläutert:

#### „user“

Dieses Element beinhaltet einen Identifier für den User, der den SOAP Request signiert.

#### „passwordCallbackClass“ bzw. „signaturePropFile“

Das Element „passwordCallbackClass“ definiert jene Klasse, die aufzurufen ist, falls ein Passwort, beispielsweise für den Zugriff auf den privaten Schlüssel, benötigt wird. Dabei greift diese Klasse beispielsweise auf einen Keystore zu, der in der Crypto Property Datei bzw. Signature Property Datei definiert ist.

Ist wie in diesem Fall für die Verschlüsselung kein eigenes „encryptionPropFile“ Element definiert, so werden die Signature Properties auch für die Verschlüsselung verwendet. Es fällt

hierbei die starke Ähnlichkeit zu „JAAS (Java Authentication and Authorization Service)“<sup>1</sup> auf und dessen Mechanismus um Passwörter zu beziehen.

#### „action“

Das Element Action definiert welche Security Action ausgeführt werden soll. In diesem Fall wurde die Action „Signature Encrypt“ gewählt, um alle definierten Elemente des SOAP Body zuerst zu signieren und anschließend zu verschlüsseln. Weitere mögliche Werte wären:

- „Signature“ bzw. „Encrypt“ für ausschließliches Signieren bzw. Verschlüsseln
- „UsernameToken“ für das Einfügen eines Tokens in den SOAP Body

#### „signatureParts“ bzw. „encryptionParts“

Innerhalb dieser Elemente werden jene Teile der SOAP Nachricht definiert die signiert bzw. verschlüsselt werden sollen. In diesem konkreten Beispiel sind dies die Elemente „CreditCard“, das sich im Namespace „http://example.org“ befindet und das Element „UserName“.

#### „encryptionKeyIdentifier“

Dieses Element definiert in welchem Format der Verschlüsselungsschlüssel in den SOAP Request integriert werden soll. Mögliche Werte hierzu sind:

- „SKIKeyIdentifier“ bindet einen Identifier für das Zertifikat ein
- „DirectReference“ bindet das Zertifikat in Form eines BinarySecurityToken in den SOAP Header ein

Listing 12 zeigt abschließend den Webservice Deployment Descriptor des Servers (server\_deploy.wsdd). Die Bedeutung der Elemente wurde bereits erläutert. Wesentlicher Unterschied zum Descriptor des Clients ist die Referenz auf den WSDoAllReceiver Handler anstatt des WSDoAllSender Handlers.

```
1 <requestFlow>
2   <handler name="DoSecurityReceiver"
3     type="java:org.apache.ws.axis.security.WSDoAllReceiver">
4     <parameter name="passwordCallbackClass"
5       value="org.apache.ws.axis.samples.wssec.doall.PWCallback"/>
6     <parameter name="action" value="Signature Encrypt"/>
7     <parameter name="signaturePropFile" value="crypto.properties" />
8   </handler>
9 </requestFlow>
```

Listing 12: WSS4J Axis Server Deployment Descriptor

<sup>1</sup>Java Authentication and Authorization Service (JAAS) ist eine Anzahl von APIs, die es Services ermöglicht Zugriffskontrollen für Users umzusetzen. Hierzu unterstützt es User basierte Authorisation [2].

## 6 Resüme und Ausblick

In den vergangenen Jahrzehnten gab es meist eine klare Unterteilungen zwischen internen Applikationen und solchen die mit externen Systemen (Kunden, Lieferanten, Partner) kommunizierten. Bei Intranet Applikationen waren meist weniger strenge Security Richtlinien notwendig. Mit der wachsenden Bedeutung von EAI (Enterprise Application Integration) enden viele Business Applikationen jedoch nicht mehr bei den Unternehmensgrenzen. Im speziellen bedeutet das, dass Applikationen nicht automatisch sicher sind, nur weil sie hinter der Unternehmensfirewall stehen. Man benötigt meist einen Security Level, der sowohl bei der internen Integration als auch bei der externen Integration Sicherheit bietet.

Weiters wird das Deployment von Web Services schwieriger. Es wird immer öfter notwendig zwischen Client und Web Service Monitoring, Auditing, Orchestration usw. einzugliedern. Dies ist jedoch beispielsweise mit einer SSL basierten Sicherheitslösung nur äußerst schwierig zu bewältigen.

Die WS-Security Spezifikation ist ein wichtiger Schritt um diesen Security Level näher zu kommen. Es stehen sehr viele große und bedeutende Unternehmen hinter diesem Standard, dies erhöht die Wahrscheinlichkeit einer umfassenden Etablierung.

Mit WSS4J haben wir in dieser Arbeit versucht einen Einblick in den derzeitigen Stand der Umsetzung dieses Standards in der Java Welt zu geben. Man merkt jedoch, dass WSS4J noch in den Anfängen steckt. Auch die nahezu nicht vorhandene Dokumentation ist ein starkes Indiz dafür. Mit der letzten Release (1.1.1) unterstützt es die grundlegenden Basisfunktionen wie etwa XML Signature und Encryption, Username Tokens, Timestamps, SAML Tokens. Derzeit wird unter anderem an der Integration von den weiteren Mitgliedern der WS-Family gearbeitet, wie beispielsweise WS-Trust und WS-Conversation, die sich derzeit in einem „sandbox“ Stadium befinden. Auf diese wurde nicht näher eingegangen, da sie sonst den Rahmen der Arbeit gesprengt hätten. Weitere Standards, die sich gerade zu etablieren versuchen sind:

- WS-Federation für zusammengesetzte Web Services, WS-Authorization (für Integrationszwecke)
- WS-Secure Conversation ( gemeinsam mit WS-Policy bedeutend für die Interoperabilität)

Eine wichtige Schlüsselrolle wird in Zukunft die Interoperabilität zwischen den unterschiedlichen Implementierungen des Standards sein. Dafür wurde unter anderem die WS-I (Web Services Interoperability Organization) ins Leben gerufen, die plattformübergreifende und implementierungsübergreifende Kommunikation zwischen Web Services sicherstellen sollen.

## Literatur

- [1] DAVID BOOTH, HUGO HAAS, FRANCIS MCCABE ERIC NEWCOMER IONA MICHAEL CHAMPION CHRIS FERRIS DAVID ORCHARD: *Web Services Architecture*. <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, 2004.
- [2] *Java™ Authentication and Authorization Service (JAAS) Reference Guide*. <http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html>.
- [3] *Apache WSS4J Webseite*. <http://ws.apache.org/wss4j/>.
- [4] *Web Services Security: SOAP Message Security V1.0*. <http://www.oasis-open.org>.
- [5] JOTHY ROSENBERG, DAVID REMY: *Securing Web Service with WS-Security*. Sams, 2004.
- [6] KLAS, UNIV.PROF. DR. WOLFGANG: *Electronic Commerce I - Skriptum zur Vorlesung*. 2000-2003.
- [7] M. HONDO, N. NAGARATNAM, A. NADALIN: *Securing Web services*. 41,:228–241,, 2002.
- [8] MARK BERTEL, JOHN BOYER, BARB FOX BRIAN LAMACCHIA ED SIMON: *XML-Signature, Syntax and Processing*. <http://www.w3.org/TR/xmlsig-core>, 2002.
- [9] NAEDELE, MARTIN: *Standards for XML and Web Services Security*. Band 36,, Seiten 96–98,, Computer, ABB Corporate Research, 2004.
- [10] REAGLE, JOSEPH: *XML-Signature Requirements*. <http://www.w3.org/TR/xmlsig-requirements>, 1999.
- [11] SIDDHARTH BAJAJ (VERISIGN), DON BOX (MICROSOFT), DAVE CHAPPELL (SONIC SOFTWARE) FRANCISCO CURBERA (IBM) GLEN DANIELS (SONIC SOFTWARE) PHILLIP HALLAM-BAKER (VERISIGN) MARYANN HONDO (IBM) CHRIS KALER (MICROSOFT) DAVE LANGWORTHY (MICROSOFT) ASHOK MALHOTRA (MICROSOFT) ANTHONY NADALIN (IBM) NATARAJ NAGARATNAM (IBM) MARK NOTTINGHAM (BEA) HEMMA PRAFULLCHANDRA (VERISIGN) CLAUS VON RIEGEN (SAP) JEFFREY SCHLIMMER (MICROSOFT) CHRIS SHARP (IBM) JOHN SHEWCHUK (MICROSOFT): *Web Services Policy Framework (WS-Policy)*. 2004.
- [12] SIDDIGUI, BILAL: *Exploring XML Encryption, Part 1 (Demonstrating the secure exchange of structured data)*. <http://www-128.ibm.com/developerworks/xml/library/x-encrypt>, 2002.
- [13] TAKESHI IMAMURA, BLAIR DILLAWAY, ED SIMON: *XML Encryption Syntax and Processing*. <http://www.w3.org/TR/xmlenc-core/>, 2002.
- [14] *PGP*. <http://www.pgp.com/>.
- [15] *SAML Version 2.0 - Standard*. <http://www.oasis-open.org/specs/index.php>.

[16] *SOAP - w3c*. <http://www.w3.org/TR/soap/>.

[17] *SSL 3.0 Spezifikation*. <http://wp.netscape.com/eng/ssl3/>.