SEMINARARBEIT IM FACHBEREICH WIRTSCHAFTSINFORMATIK

Service-orientierte Architekturen

Sicherheit in WebServices

Eingereicht von	*Melanie Storm* *Schulstr. 19* *22880 Wedel*
Fachrichtung	*B_Winf*
Matrikelnummer	*2914*
Fachsemester	*4*
Eingereicht bei	*Prof. Dr. Sebastian Iwanowski* *Fachhochschule Wedel* *Feldstr. 143* *22880 Wedel*

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

WebServices gewinnen immer mehr an Bedeutung und somit steigen auch die sicherheitsgefährdenden Angriffe. Um die sichere Übertragung von SAOP-Nachrichten im Internet zu gewährleisten wurde die *Security Roadmap* verabschiedet.

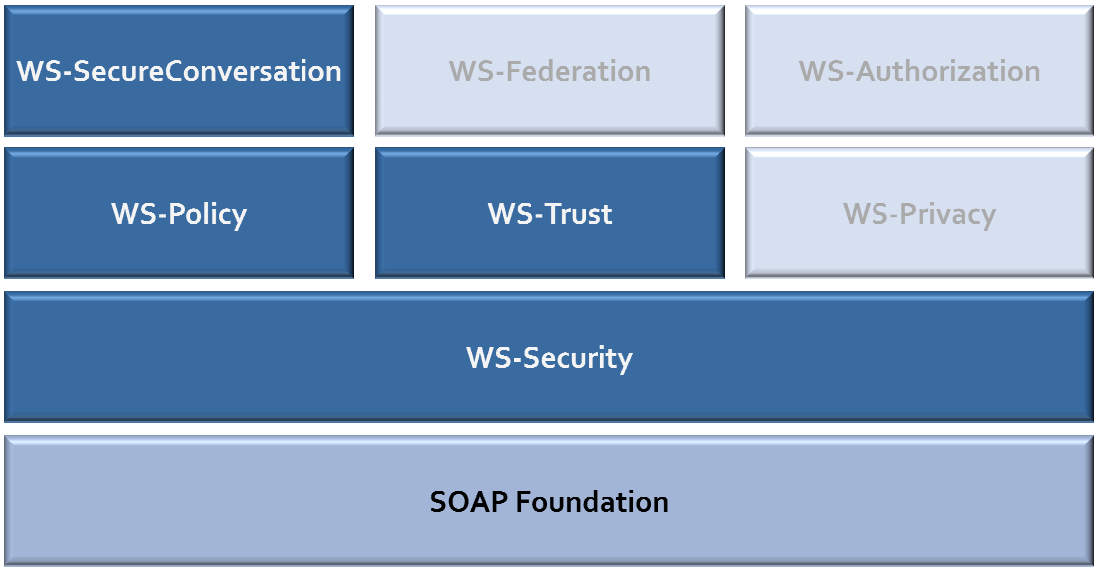

Abbildung 1: Security Roadmap

Die Roadmap stellt die unterschiedlichen Bausteine dar, welche im Endeffekt die Sicherheit garantieren sollen. Sie ist von unten nach oben zu lesen, d.h. alle Bausteine bauen auf dem SOAP-Protokoll auf. Diese Arbeit legt ihren Schwerpunkt auf *WS-Security*, welches *XML Encryption* und *XML Signature* in SOAP-Nachrichten einbettet. Außerdem werden noch *WS-Policy, WS-Trust* und *WS-SecureConversation* in ihren Grundzügen erläutert. Auf *WS-Federation, WS-Authorization* und *WS-Privacy* wird kaum eingegangen, da nicht alle Spezifikationen veröffentlicht sind und dies im Rahmen der Arbeit zu weit ins Detail gehen würde.

Die Absicht dieser Arbeit besteht nicht darin auf alle Spezifikationen im Detail einzugehen. Viel mehr ist sie so angelegt das man einen groben Überblick über die Spezifikationsvielfalt erhält und als Grundlage für eine intensivere Auseinandersetzung mit den Spezifikationen dienen soll.

2 Motivation

WebServices müssen *Integrität, Vertraulichkeit, Autorisierung* und *Authentizität* gewährleisten.

Die *Integrität* überprüft, ob eine Nachricht während der Übertragung manipuliert wurde bzw. stellt sicher das dies nicht geschehen kann.

Das nur ein bestimmter Personenkreis Zugriff auf sensible Daten hat stellt die *Vertraulichkeit* sicher.

Ob eine Funktionalität eines WebServices überhaupt vom Anfrager genutzt werden darf, wird mittels *Autorisierung* reguliert.

Die *Authentizität* überprüft die Herkunft der Nachricht und somit die Identität des Absenders.

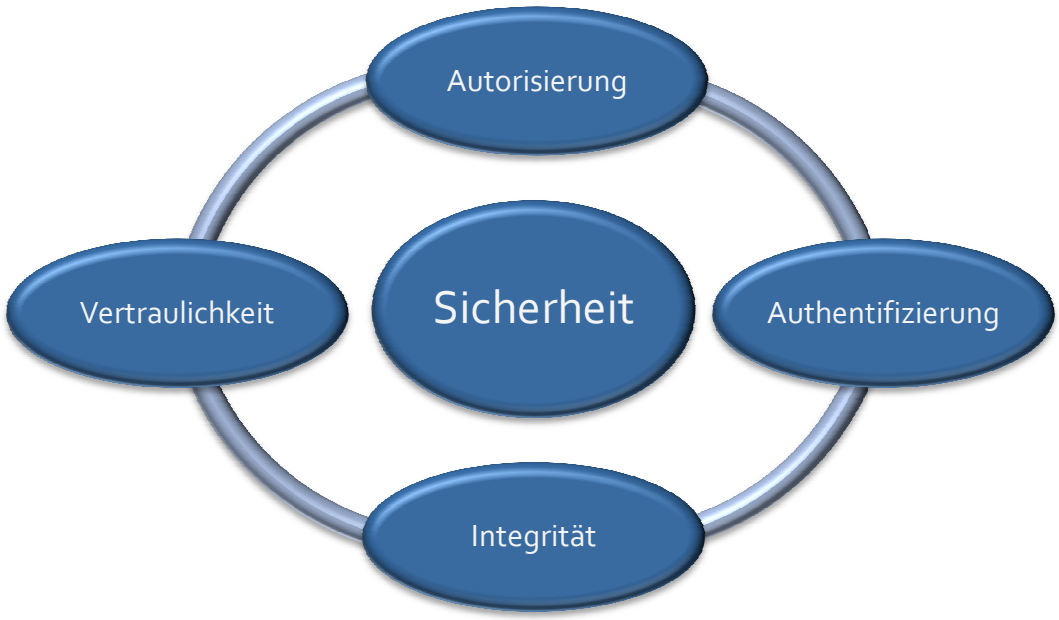

Abbildung 2: Sicherheitsaspekte

Gleich zu Beginn stellt sich die Frage, wieso man die Sicherheit von WebServices nicht mit den vorhandenen Sicherheitslösungen, wie SSL, erreichen kann. Das Problem ist die reine Transportsicherheit die SSL bietet,

d.h. die Nachrichten werden nur während des Transports gesichert und liegen bei evtl. Zwischenstationen ungesichert vor.

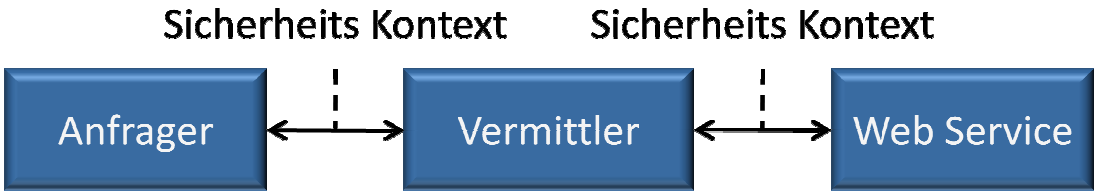

Abbildung 3: Ende zu Ende Sicherheit

Dies ist so nicht hinnehmbar und man musste die Sicherheit an die Information, also die Nachricht selbst, binden. So entsteht dann statt einer *Ende zu Ende Sicherheit* eine *Punkt zu Punkt Sicherheit*.

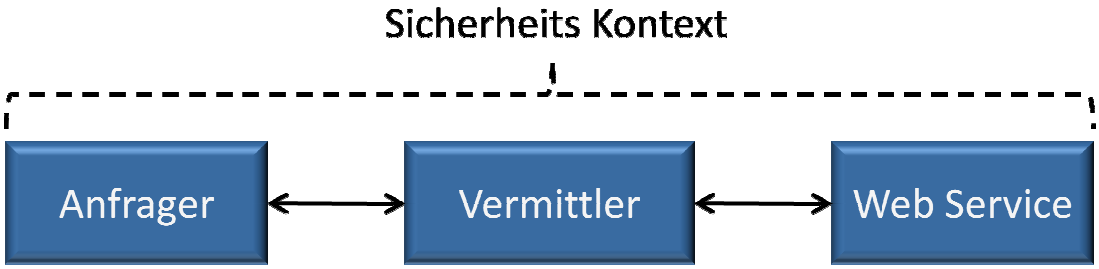

Abbildung 4: Punkt zu Punkt Sicherheit

Außerdem können bei SSL nur die gesamten Nachrichten gesichert werden und nicht nur Teile davon, welches aber bei WebServices teilweise sinnvoll genutzt werden könnte. Zum Beispiel könnte es sinnvoll sein, das ein Vermittler den Header der Nachricht lesen kann, so dass er die Nachricht korrekt weiterleiten kann, aber die sensiblen Daten der Nachricht sollen trotzdem nicht einsehbar sein.

3 Fallbeispiel

Um die folgenden Code-Beispiele etwas verständlicher zu gestalten, betrachten wir im weiteren Verlauf der Arbeit das folgende kleine Beispiel.

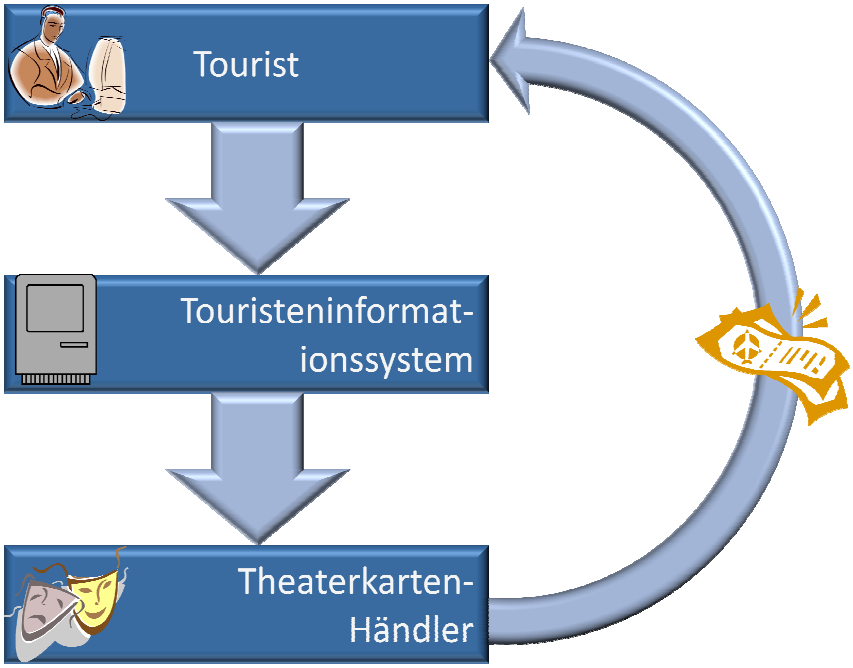

Abbildung 5: Bsp. Tourist kauft Theaterkarten

Initiator ist ein Tourist, welcher in seiner Besichtigungstour einen Theaterbesuch integrieren möchte. Dazu benutzt er wie schon für die Zusammenstellung der Besichtigungstour das Touristeninformationssystem, welches wiederum in Kontakt mit einem Theaterkartenhändler steht, welchem er die Anfrage des Touristen weiterleitet.

Die SOAP-Nachricht könnte beispielsweise wie folgt aufgebaut sein. Sie besteht nur aus einem `Body` (05-14), welcher die eigentlichen Daten enthält. In diesem Fall wird einmal die gewünschte Vorstellung (06-08) übertragen. Diese kann z.B. das gewünschte Theater, Beginn der Vorstellung, Preiskategorie und Name des Stücks enthalten. Da die Karte direkt bezahlt werden soll, damit sie dem Touristen umgehend zugeschickt werden kann, werden ebenfalls die Bankdaten des Touristen übertragen (09-13).

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <SOAP-ENV:Envelope
03    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
04    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
05    <SOAP-ENV:Body>
06      <vorstellung>
07        …
08      </vorstellung>
09      <kontoverbindung>
10        <inhaber type="xsd:string">Max Mustermann</inhaber>
11        <blz type="xsd:string">20050550</blz>
12        <ktonr type="xsd:string">1379345876</ktonr>
13      </kontoverbindung>
14    </SOAP-ENV:Body>
15  </SOAP-ENV:Envelope>
```

Abbildung 6: SAOP-Bsp. Tourist kauft Theaterkarte

4 WS-Security

WS-Security definiert die Einbindung der bereits vorhandenen XML Sicherheitslösungen in SOAP-Nachrichten.

4.1 XML Encryption

XML Encryption dient der Verschlüsselung von XML Dokumenten. Damit ist es möglich auch mehrere Teile einer SOAP-Nachricht zu verschlüsseln. Eine Verschachtelung ineinander ist allerdings nicht zulässig.

In unserem Beispiel wollen wir die sensiblen Bankdaten verschlüsseln.
Als erstes fügen wir der SOAP-Nachricht einen Header hinzu, welcher das Sicherheitselement `<wsse:Security/>` enthält (06-10). Innerhalb dieses Elements werden die verschlüsselten Bestandteile referenziert (07-09) um dem Empfänger von vornherein mitteilen zu können, welche Bestandteile zu entschlüsseln sind. In unserem Beispiel haben sich Sender und Empfänger auf einen Schlüssel geeinigt, so dass eine Übertragung des Schlüssels nicht notwendig ist. Der zu verschlüsselnde Teil der Nachricht wird vollständig durch ein `<xenc:EncryptedData/>` (16-26) Element ersetzt, welches per ID, in diesem Fall BankData, identifiziert wird. Über diese ID erfolgt auch die Referenzierung aus dem Header (16 und 8). Der Typ des Elements gibt an, ob ein gesamtes Element oder nur der Inhalt eines Elements verschlüsselt wird. In der `<xenc:EncryptionMethod>` (18-20) wird die Verschlüsselungsmethode angegeben, in diesem Fall wird der RSA-Algorithmus angewandt. Die verschlüsselten Daten sind in das `<xenc:CipherValue>` Element (22-24) eingebettet.

```
01 <SOAP-ENV:Envelope
02   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
03   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
04   xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
05   <SOAP-ENV:Header>
06     <wsse:Security>
07       <xenc:ReferenceList>
08         <xenc:DataReference URI="#BankData"/>
09       </xenc:ReferenceList>
10     </wsse:Security>
11   </SOAP-ENV:Header>
12   <SOAP-ENV:Body>
13     <vorstellung>
14       …
15     </vorstellung>
16     <xenc:EncryptedData Id="BankData"
17       Type="http://www.w3.org/2001/04/xmlenc#Element">
18       <xenc:EncryptionMethod
19         Algorithm=
20         "http://www.w3.org/2001/04/xmlenc#rsa-1_5/">
21       <xenc:CipherData>
22         <xenc:CipherValue>
23           sdjGhfHhsky…
24         </xenc:CipherValue>
25       </xenc:CipherData>
26     </xenc:EncryptedData>
27   </SOAP-ENV:Body>
28 </SOAP-ENV:Envelope>
```

Abbildung 7: Bsp. SOAP-Nachricht für XML Encryption

4.2 XML Signature

Wie bei der Encryption wird dem Header ein `<wsse:Security>` Element hinzugefügt (06-34). Dieses enthält im `<ds:Signature>` Element (07-33) die Signatur. Diese widerum enthält Angaben nur Normalisierungmethode[1] (`CanonicalizationMethod` (09-11)), zur Signierungsmethode (`SignatureMethod` (12-14)) und eine Referenz auf die signierten Daten innerhalb des `<ds:Reference>` Elements (15-22). Dieses Element enthält nicht nur als Attribut die URI der signierten Daten, in diesem Fall der MessageBody (vergleiche 15 und 35), sondern auch den verwendeten

[1] Die Normalisierungsmethode muss auf ein XML Dokument angewendet werden damit XML Dokumente mit gleichem Inhalt, aber unterschiedlicher Struktur den gleichen Hashwert erhalten.

Hashwert-Algorithmus, hier SHA-1. Der damit ermittelte Hashwert wird im `<ds:DigestValue>` Element (19-21) festgehalten. Da mehrere Bestandteile der Nachricht signiert werden können, wird pro Bestandteil ein `<ds:Reference>` Element benötigt. Diese drei Angaben, die Normalisierungsmethode, die Signierungsmethode und die Referenz(en) werden zusammenfasst zur `SignedInfo` (08-23). Auf diese wird widerum ein *Hash-Algorithmus* angewendet. Der entstandene Hashwert wird anschließend verschlüsselt und im `<ds:SignatureValue>` (24-26) Element übertragen. In diesem Fall wird der *Hashwert* der `SignedInfo` mittels SHA-1 berechnet und anschließend symmetrisch per DAS verschlüsselt (14). Wenn der verwendete Schlüssel, so wie in unserem Bsp., dem Empfänger nicht bekannt ist muss er mit übertragen werden. Dies geschieht mittels des Elements `<ds:KeyValue>` (27-32).

```
<SOAP-ENV:Envelope
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
  <SOAP-ENV:Header>
    <wsse:Security>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm=
            "http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
            Algorithm=
            "http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:Reference URI="#MessageBody">
            <ds:DigestMethod
              Algorithm=
              "http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>
              JwFsd3eQc0iXlJm5PkLh7...
            </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          BSxlJbSiFdm5Plhk...
        </ds:SignatureValue>
        <ds:KeyValue>
          <ds:DSAKeyValue>
            <ds:P>…</ds:P> <ds:Q>…</ds:Q>
            <ds:G>…</ds:G> <ds:Y>…</ds:Y>
          </ds:DSAKeyValue>
        </ds:KeyValue>
      </ds:Signature>
    </wsse:Security>
  <SOAP-ENV:Body wsu:Id="MessageBody">
    …
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Abbildung 8: Bsp. SOAP-Nachricht für XML Signature

4.3 Exkurs: Digitale Signatur

Wenn man eine Quelle signieren möchte, wendet man als erstes einen *Hash-Algorithmus* darauf an. Der *Hash-Algorithmus* berechnet den sogenannten *Hashwert*. Dieser Wert ist eindeutig, d.h. es gibt kein anderes Dokument aus welchem dieser *Hashwert* erzeugt werden kann. Außerdem kann man vom *Hashwert* nicht zurück auf das zugrundeliegende Dokument schließen. Der *Hashwert* wird nun verschlüsselt, hier mit dem *PrivateKey* des Absenders.

Der Empfänger entschlüsselt die Nachricht mit dem *PublicKey* des Absenders und erhält den *Hashwert*. Um nun zu kontrollieren, ob die Quelle nicht verändert wurde wendet er den *Hash-Algorithmus* auf die Quelle an. Wenn der neu errechnete *Hashwert* mit dem entschlüsselten *Hashwert* aus der Nachricht übereinstimmt wurde die Quelle nicht verändert und so die *Integrität* der Quelle nachgewiesen. Außerdem stellt dieses Verfahren auch die *Authentizität* sicher.

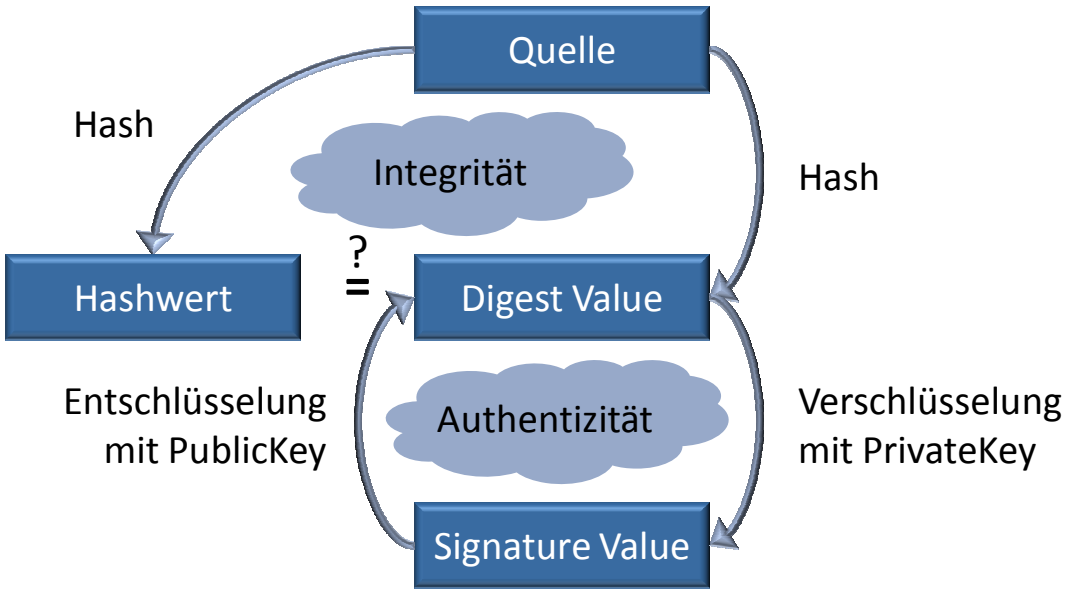

Abbildung 9: schematischer Ablauf der digitalen Signatur

5 WS-Policy

Policies regeln die Kommunikation zwischen den WebServices. In ihnen können Anforderungen festgehalten werden, welche dann vom Partner-Service einzuhalten sind, ansonsten kommt keine Kommunikation zustande. Um diese Anforderungen zu spezifizieren wurde keine neue Syntax erfunden, sondern es wird auf die schon existierenden Spezifikationen zurückgegriffen. Sicherheitsanforderungen werden in einer speziellen *WS-SecurityPolicy* eingebettet, welche wiederum die bekannten *WS-Security* Elemente enthalten kann, wie z.B. die *XML Signature*.

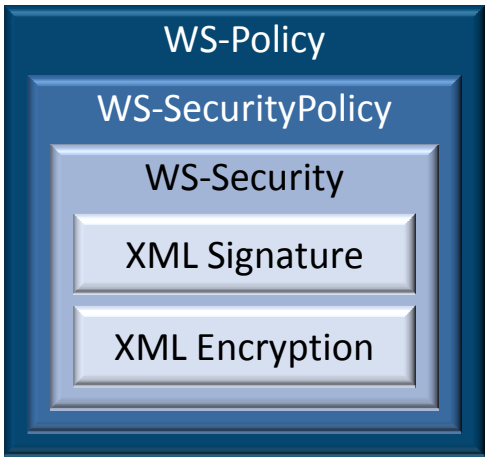

Abbildung 10: Aufbau WS-Policy

`Wsp` ist im folgendem Beispiel der Namensraum für die *WS-SecurityPolicy*, in welche die Restriktionen eingebettet sind. Dort wird verlangt das der Aufrufer entweder ein *Kerberos-Ticket* oder ein *X509-Zertifikat* besitzt. Dies geschieht mit dem für *Policies* üblichen Elementen `ExactlyOne` und `All`. Das `ExactlyOne` sagt aus das sich der Aufrufer für eine der dort enthaltenen Möglichkeiten entscheiden muss, hier sind dies zwei, die jeweils in ein `All` Element eingebettet sind. Es müssen innerhalb eines AllElements alle Forderungen erfüllt werden. Im Beispiel enthält es nur jeweils eine Restriktion, aber es sind auch durchaus mehrere vorstellbar, so könnte man

zum Beispiel zum Zertifikat noch einen speziellen Verschlüsselungsalgorithmus verlangen.

```
01  <wsp:Policy>
02    <wsp:ExactlyOne>
03      <wsp:All>
04        <wsse:SecurityToken>
05          <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
06        </wsse:SecurityToken>
07      </wsp:All>
08      <wsp:All>
09        <wsse:SecurityToken>
10          <wsse:TokenType>wsse:X509v3</wsse:TokenType>
11        </wsse:SecurityToken>
12      </wsp:All>
13    </wsp:ExactlyOne>
14  </wsp:Policy>
```

Abbildung 11: Bsp. WS-Policy

Die Restriktionen werden mit Hilfe von *Security Policy Assertions* definiert. Dabei werden die folgenden sechs *Assertions* werden unterschieden:

SecurityToken Assertion

Es werden wie im obigen Beispiel die zulässigen *Token* spezifiziert.

Confidentiality Assertion (6-12)

Bezieht sich auf einen Teil der Nachricht, welcher verschlüsselt warden muss. Auch der Verschlüsselungsalgorithmus kann vorgegeben werden.

Integrity Assertion (13-27)

Spezifiziert einen zu signierenden Teil der Nachricht und die zu verwendenden Algorithmen.

Visibility Assertion (28-32)

Im Gegensatz zu den beiden vorherigen *Assertions* wird hier für einen angebenen Nachrichtenteil verlangt das dieser sichtbar sein muss, also nicht verschlüsselt werden darf.

SecurityHeader Assertion (33-35)

Sie macht genauere Angaben zur Verwendung von *SecurityHeadern*, z.B. kann dort die Nutzung von `References` vorgeschrieben werden.

MessageAge Assertion (36)

Dort kann das maximale Alter der Nachricht angegeben werden.

```
01  <wsp:Policy
02    xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
03    xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
04    <wsp:SpecVersion    wsp:Usage="wsp:Required"
05      URI="http://schemas.xmlsoap.org/ws/2002/07/secext"/>
06    <wsse:Confidentiality    wsp:Usage="wsp:Required">
07      <wsse:Algorithm   Type="wsse:AlgEncryption"
08        URI="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
09      <MessageParts>
10        wsp:GetInfosetForNode(wsp:GetBody(.))
11      </MessageParts>
12    </wsse:Confidentiality>
13    <wsse:Integrity    wsp:Usage="wsp:Required">
14      <wsse:Algorithm    Type="wsse:AlgCanonicalization"
15        URI=
16        "http://www.w3.org/Signature/Drafts/xml-exc-c14n"/>
17      <wsse:Algorithm Type="wsse:AlgSignature"
18        URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
19      <wsse:SecurityToken>
20        <wsse:TokenType>wsse:X509v3</wsse:TokenType>
21      </wsse:SecurityToken>
22      <MessageParts
23        Dialect=
24        "http://schemas.xmlsoap.org/2002/12/wsse#soap">
25        S:Body
26      </MessageParts>
27    </wsse:Integrity>
28    <wsse:Visibility wsp:Usage="wsp:Required">
29      <MessageParts>
30        wsp:GetInfosetForNode(wsp:GetBody(.))
31      </MessageParts>
32    </wsse:Visibility>
33    <wsse:SecurityHeader  wsp:Usage="wsp:Required"
34                  Must Prepend="true"
35                  MustManifestEncryption="true"/>
36    <wse:MessageAge wsse:Usage="wsp:Required" Age=3600"/>
37  </wsp:Policy>
```

Abbildung 12: Bsp. SecurityAssertions

Bezogen auf unser Fallbeispiel müsste man verlangen das die Bankdaten signiert und verschlüsselt werden, aber die Vorstellungsdaten für das Touristeninformationssystem sichtbar sein müssen, damit es die Nachricht zum richtigen Kartenhändler weiterleiten kann.

6 WS-Trust

Wir wissen nun wie man Nachrichten(teile) verschlüsseln und signieren kann und wie man Sicherheitsanforderungen spezifiziert, aber was ist, wenn der Aufrufer nun diese Anforderungen nicht erfüllen kann? Dieses Problem versucht *WS-Trust* zu lösen.

Es gibt zwei unterschiedliche Arten von *SecurityTokens*. Zum einen *UnsignedSecurityTokens*, in welchen beispielsweise ein Username mit Passwort übertragen wird. Das *Token* selber ist nicht verschlüsselt oder signiert. Es bedarf also zusätzlicher Mechanismen damit die Informationen nicht von Unbefugten ausgelesen werden können. Zum anderen gibt es *SignedSecurityTokens*, wie z.B. *X509-Zertifikate* und *Kerberos-Tickets*. Diese werden nur von speziellen Services ausgegeben.

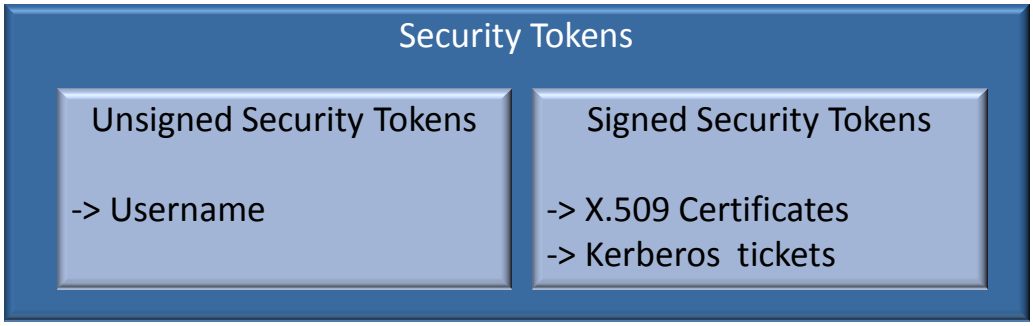

Abbildung 13: Security Tokens

Ein zusätzlicher Service, der *Security Token Service*, stellt benötigte *SecurityTokens* aus mit denen der Aufrufer dann doch noch ans Ziel kommt und den gewünschenten Service nutzen kann.

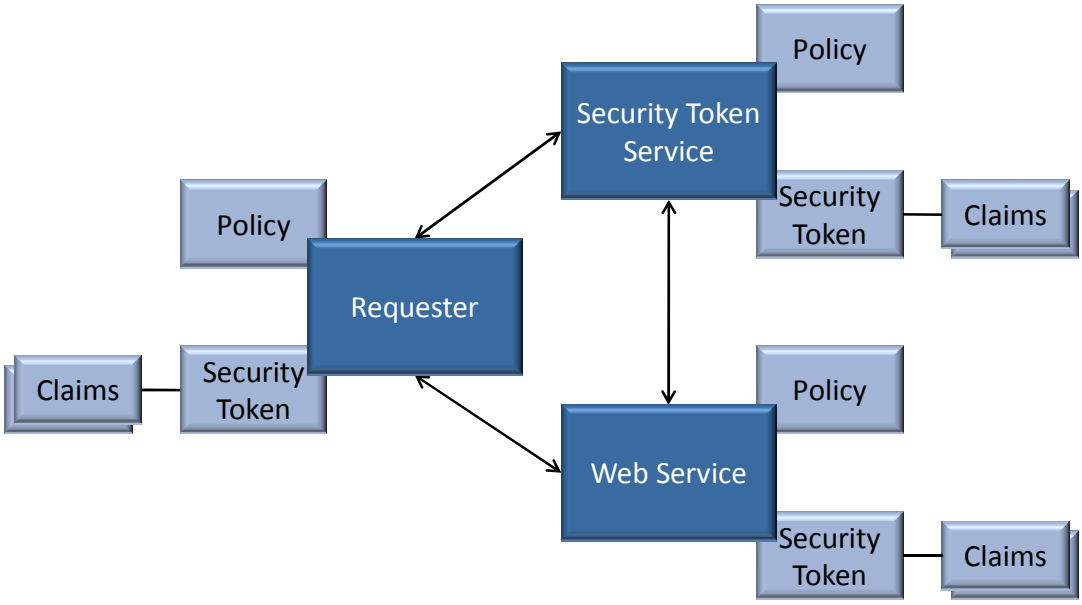

Abbildung 14: WS-Trust Zusammenhang

Der Aufrufer möchte einen Service nutzen und fordert dazu dessen *Policy* an. Es wird versucht eine der geforderten Restriktionen einzuhalten, vorausgesetzt es gibt überhaupt mehrere Alternativen. Kann keine vollständig erfüllt werden, kann der Service nicht benutzt werden. In diesem Fall könnte man natürlich nach anderen Services suchen, wo die Anforderungen erfüllt werden können, oder man versucht über weitere Services den Anforderungen gerecht zu werden. Zum Beispiel könnte ein *Kerberos-Ticket* verlangt werden, aber es ist „nur" ein *X509-Zertifikat* vorhanden, so könnte ein weiterer Service auf Grund des *X509-Zertifikats* ein *Kerberos-Ticket* ausstellen, mit welchem nun die eigentliche Anfrage gestellt werden kann.

Mit Hilfe von *WS-Trust* können *Tokens* angefordert, ausgegeben, ausgetauscht und validiert werden. Im folgenden ein Beispiel zum Ausstellen von *SecurityToken*. Ein Requester besitzt ein *X509-Zertifikat* und möchte einen Web Service nutzen. Er fordert die *Policy* des gewünschten Web Services an und stellt fest, das dieser ein eigenes spezielles Tokenformat (Custom Token) fordert. Daraufhin wendet sich der Requester an einen

Security Token Service, welcher authorisiert ist Custom Tokens auszustellen, und signiert seine Nachricht mit Hilfe seines *X509-Zertifikats*. Angenommen dabei wird ein „neuer" *Timestamp* übermittelt, welcher den Anforderungen des *Security Token Service* genügt, und das *X509-Zertifikat* ist gültig, dann wird ein Custom Token ausgestellt und übermittelt. Sollte der *Timestamp* zu alt sein oder gar nicht vorhanden sein, schickt der *Security Token Service* eine *Challenge* zurück. Dies ist ein String, welcher vom Requester mit seinem Zertifikat signiert werden muss um die *Authentizität* sicherzustellen. Der *Security Token Service* überprüft die Signatur und übermittelt im Erfolgsfall das gewünschte Custom Token.

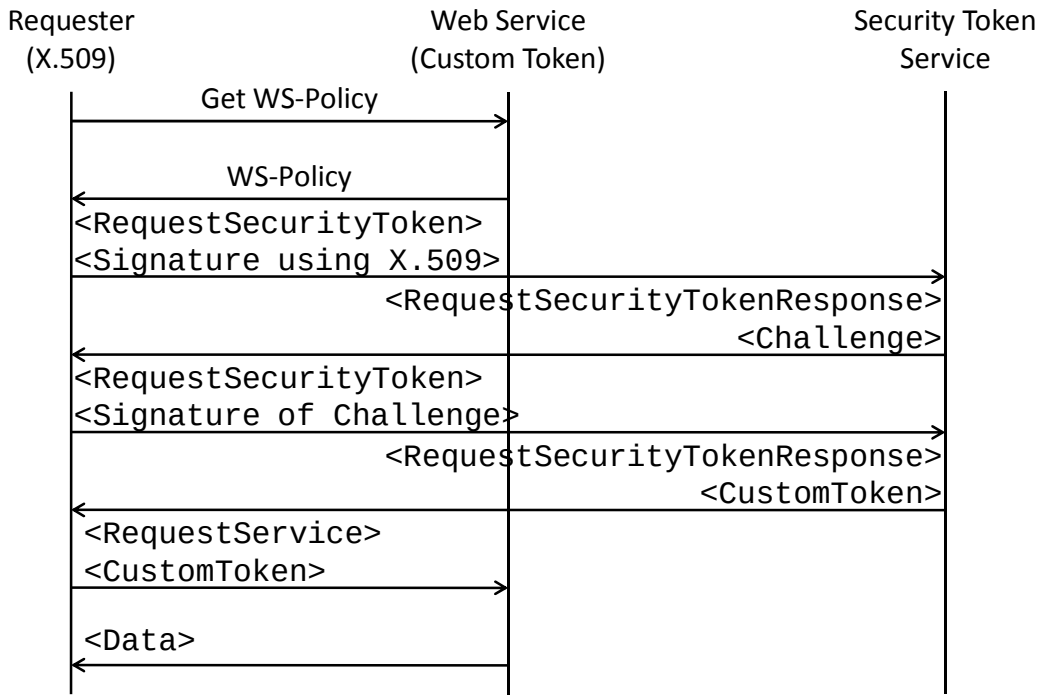

Abbildung 15: Bsp.-Ablauf WS-Trust

7 WS-SecureConversation

Im Moment muss jede Nachricht mit Hilfe von *WS-Trust* gesichert werden. Dies führt zu einem relativ großen Aufwand, welchen man reduzieren kann, wenn man nicht pro Nachricht ein *Token* benutzt, sondern pro *Conversation*. Dieses Problem löst *WS-SecureConversation* indem es `SecurityContextToken` ausstellt, welche dann für mehrere Nachrichten verwendet werden können.

Der Requester fragt zu Beginn des Nachrichtenaustauschs beim *Security Token Service* ein `SecurityContextToken` an und verwendet es dann für die gesamte weitere Kommunikation zum eigentlichen WebService.

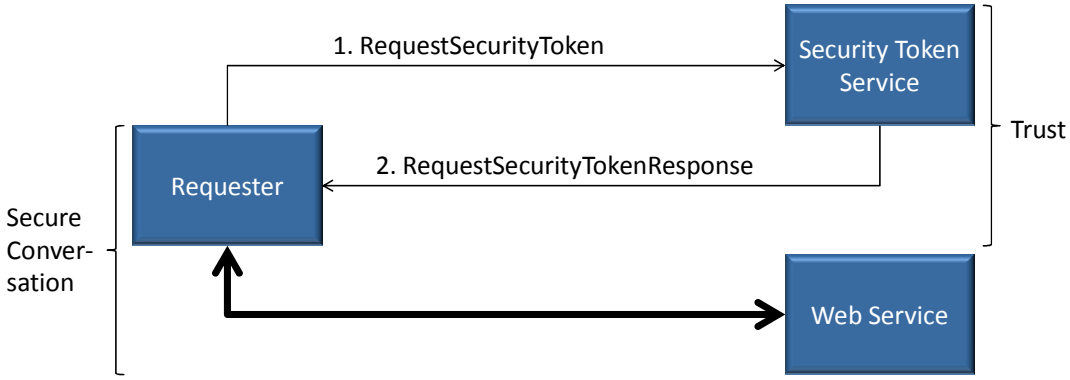

Abbildung 16: Ablauf WS-SecureConversation

8 WS-*

Zu den bereits vorgestellten Spezifikationen gibt es noch weitere Erweiterungen, welche aber noch nicht vollständig spezifiziert sind bzw. auf Grund des Umfangs nicht weiter eingegangen wurde.

8.1 WS-Federation

Es gibt nicht nur eine Sicherheitsdomäne in welcher ein Trust-Verhältnis besteht, sondern mehrere. *WS-Federation* regelt die Vereinigung dieser Sicherheitsdomänen, so dass die *Security Token Services* nun auch Zertifikate einer anderen Domäne erstellen dürfen. Dazu müssen sie selbstverständlich vorher von dieser Domäne über einen anderen *Security Token Service* berechtigt worden sein. Die Koppelung erfolgt also dadurch das *Security Token Services* mehreren Sicherheitsdomänen angehören können.

8.2 WS-Privacy

Die *Policies* sollen um Angaben zu Datenschutzrichtlinien erweitert werden können.

8.3 WS-Authorization

Übernimmt die Prüfung der Zugriffsberechtigung und baut wahrscheinlich auf dem *SecurityToken*-Konzept auf.

9 Fazit

Nachdem wir uns nun mit sämtlichen Spezifikationen der *WS-Security Roadmap* beschäftigt haben, mit manchen intensiver als mit anderen, kann man sich unterschiedlicher Meinung zu dieser Spezifikationsansammlung sein.

Es ist nicht abzustreiten das der Sicherheitsaspekt bei Web Services sehr wichtig ist und für ihre weitere Verbreitung unbedingt berücksichtigt werden muss. Sicherheit zu gewährleisten ist immer eine sehr schwierige und komplexe Angelegenheit wie man hier wieder einmal schön sehen kann, wenn man das gesamte Spektrum abdecken möchte. Es wurde versucht das Ganze etwas zu modularisieren, was meiner Ansicht nach auch gelungen ist. Andererseits ist dadurch eine Vielzahl an Spezifikationen entstanden und man muss sich erstmal darin zurechtfinden um die richtige für seine eigenen speziellen Anforderungen zu finden. Außerdem gibt es teilweise sehr starke Abhängigkeiten zwischen den einzelnen Spezifikationen. Wenn man z.B. *WS-Trust* nutzen möchte, dann muss man auch *WS-Policy* und *WS-Security* implementieren. Außerdem sind leider noch nicht alle Spezifikationen vorhanden, so fehlen noch die kompletten Spezifikationen für *WS-Privacy* und *WS-Authorization*. Dies führt unweigerlich zu Unsicherheiten beim Nutzer, da man sich nicht sicher sein kann wann diese veröffentlicht werden und ob dies nicht sogar evtl. nie der Fall sein wird. Deswegen, vermute ich, ist auch die Toolunterstützung aktuell noch sehr gering und bis auf *WS-Security* sind die Spezifikationen noch wenig praxisrelevant. Das *WS-Security* scheinbar schon häufiger zum Einsatz kommt liegt allerdings hauptsächlich daran, das dort bereits bestehende und etablierte Techniken, wie z.B. *XML Encryption*, genutzt werden.

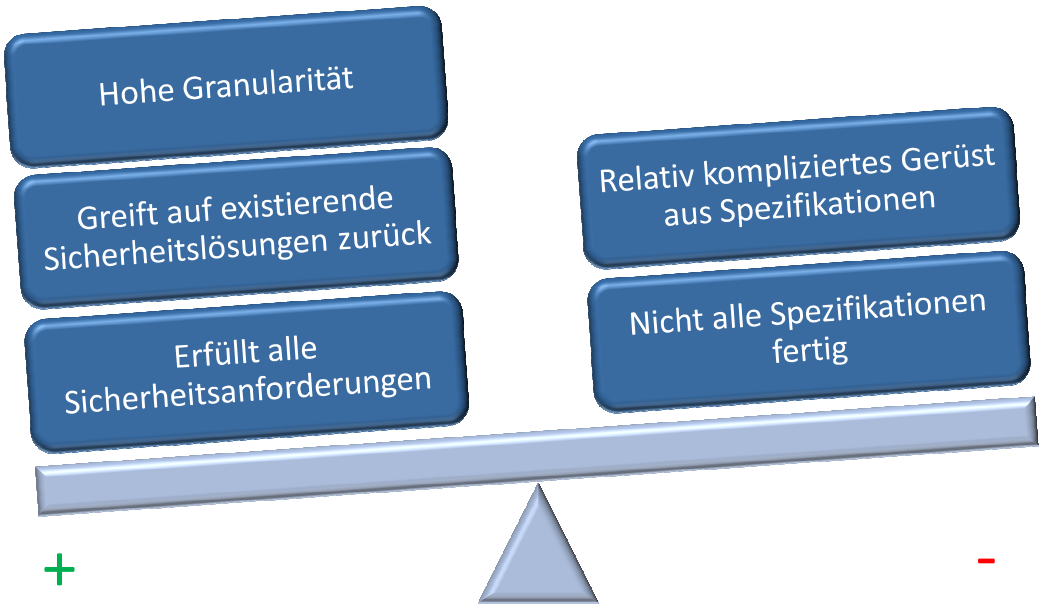

Abbildung 17: Vor- und Nachteile

Alles in allem sind die Spezifikationen brauchbar und erfüllen alle Anforderungen an Sicherheitsimplementierungen, mit der Einschränkung das die fehlenden Spezifikationen noch veröffentlicht werden müssen.

Literaturverzeichnis

Wolfgang Dostal / Mario Jeckle / Ingo Melzer / Barbara Zengler
Service-orientierte Architekturen mit Web-Services, Konzepte – Standards – Praxis
Spektrum 2005, ISBN 3-8274-1457-1

Security in a Web Services World: A Proposed Architecture and Roadmap
Stand: April 2002
URL: http://www-106.ibm.com/developerworks/library/ws-secmap/

OASIS
Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
Stand: März 2004

W3C
XML Encryption Syntax and Processing
W3C Recommendation 10.12.02
URL: http://www.w3.org/TR/xmlenc-core/

IETF/W3C
XML-Signature Syntax and Processing
W3C Recommendation 12.02.02
URL: http://www.w3.org/TR/xmldsig-core/

Web Services Policy Framework (WS-Policy)
Stand: September 2004

Web Services Security Policy Language (WS-SecurityPolicy)
Stand: Juli 2005 Version: 1.1

Web Services Trust Language (WS-Trust)
Stand: Februar 2005

Web Services Secure Conversation Language (WS-SecureConversation)
Stand: Februar 2005

Web Services Federation Language (WS-Federation)
Stand: Juli 2003 Version: 1.0