



Mit Sicherheit

# WS-Security in komplexen Landschaften

Michel Alessandrini, Willi Nüßer, Michael Pollmeier

*Komplexe SOA-Landschaften erfordern besondere Maßnahmen und Kenntnisse für den Einsatz von Sicherheitslösungen. Dieser Artikel beschreibt die Verwendung der WS-Security-Spezifikation für ein Szenario mit heterogenen Services und einem SOAP-Mediator als Zwischenstation.*



## Sicherheit verteilter Services

► Unternehmen müssen sich heute schneller auf Marktänderungen einstellen als noch vor einigen Jahren. Die Informationstechnik muss sich diesem Tempo anpassen. Serviceorientierte Architekturen (SOA) liefern ein mächtiges Konzept, um Software in handhabbare Bausteine aufzuteilen und entsprechend den Markterfordernissen zu einem Geschäftsablauf zu verknüpfen. Mit einem neuen Architektur-Konzept sind natürlich auch Veränderungen auf technischer Ebene verbunden, insbesondere auch für die Sicherheit. Diese sind teilweise so tiefgreifend, dass bewährte Technologien nicht auf SOA adaptiert werden können. Aus diesem Grund werden neue Ideen, Spezifikationen und Produkte entwickelt, von denen einige zusammenarbeiten und andere miteinander konkurrieren.

Für die sichere Zusammenarbeit verteilter Webservices beginnt sich die von der OASIS entwickelte WS-Security-Spezifikation durchzusetzen [OWS04]. Sie definiert die Integration von Sicherheitselementen in SOAP-Nachrichten und verwendet dafür teilweise bereits bestehende Spezifikationen. Auf diese Weise ermöglicht WS-Security die Verwendung von digitalen Signaturen (XML-Signature) für die Integrität und die Unabstreitbarkeit der Daten sowie die Integration von verschlüsselten Elementen (XML-Encryption) für die vertrauliche Kommunikation zwischen Webservices [BUM02]. Für die Authentifizierung werden verschiedene Sicherheits-Token unterstützt. Dazu gehören einfache Benutzername/Kennwort-Kombinationen ebenso wie die Integration eines X.509-Zertifikats oder die Verwendung von SAML-Assertions.

## Warum nicht einfach SSL?

Für die sichere Kommunikation zwischen Applikationen sind Verfahren wie TLS/SSL etabliert, die Daten auf der Transportebene sichern. Sie sind als zuverlässig anerkannt und finden eine breite Unterstützung in Softwarelandschaften. Warum also sollten diese Verfahren nicht auch in serviceorientierten Architekturen verwendet werden? Damit bliebe dem Entwickler lästige Konfigurationsarbeit mit einer komplexen Technologie wie WS-Security erspart. Und tatsächlich: für die meisten der derzeit in Betrieb bzw. noch in Entwicklung befindlichen Architekturen bietet WS-Security kaum Vorteile. Sicherheitsanforderungen wie Integrität und Unabstreitbarkeit der Daten sowie die Vertraulichkeit der Nachricht können auch mit Sicherheitsmaßnahmen auf der Transportebene umgesetzt werden. Auch eine Authentifizierung kann beispielsweise mit X.509-Zertifikaten über SSL erfolgen. Es ist daher wichtig zu bestimmen, in welchen Szenarien WS-Security überhaupt erforderlich ist.

Solange Service-Consumer und Service direkt miteinander kommunizieren (s. Abb. 1a), ist der einzig nennenswerte Vorteil von WS-Security die Unabhängigkeit vom Transportprotokoll. Wenn die Kommunikation beispielsweise von HTTP(S) auf JMS umgestellt werden soll, wird die Verschlüsselung zu einem Problem. WS-Security arbeitet hingegen auf Nachrichtenebene, daher spielt das verwendete Transportprotokoll für die Sicherheit der Services keine Rolle. Andere Vorzüge von WS-Security, wie die Absicherung eines Teildokumentes statt des gesamten Datenstroms, sind in solchen einfachen Szenarien nur in wenigen Situationen wirklich erforderlich.

In komplexeren Umgebungen können die Sicherheitsanforderungen an Software jedoch nicht immer auf der Transportebene umgesetzt werden. Sobald eine Zwischenstation in der Kommunikation existiert, welche die Nachricht interpretieren oder verändern soll, reichen Verfahren wie SSL nicht mehr aus (s. Abb. 1b). Dies lässt sich am einfachsten anhand der Nachrichtenverschlüsselung nachvollziehen: Ein Sender sendet eine Nachricht an eine Zwischenstation, welche die Nachricht anhand der enthaltenen Metainformationen anpasst und an den Empfänger weiterleitet. Wenn die Verschlüsselung auf Transportebene stattfindet, kann die Zwischenstation die gesamte Nachricht lesen. Es ist demnach keine Ende-zu-Ende-Vertraulichkeit zwischen Sender und Empfänger hergestellt.

Mit WS-Security können gezielt Bereiche der Nachricht verschlüsselt werden. Auf diese Weise können die Metainformationen unverschlüsselt und für die Zwischenstation lesbar übertragen werden.

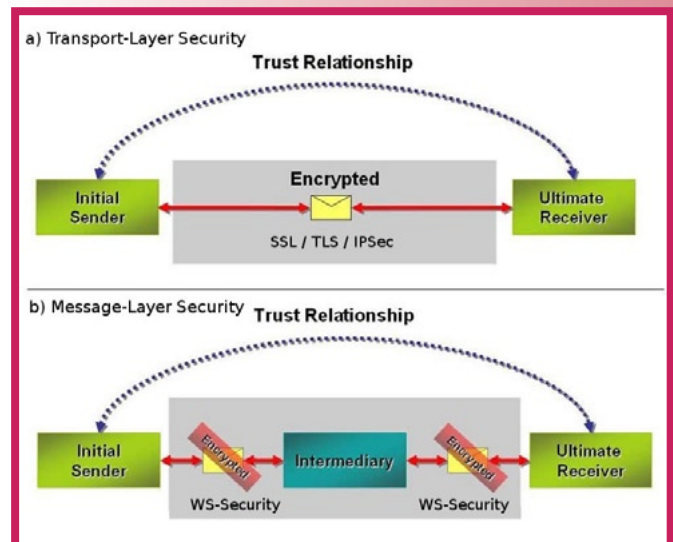


Abb. 1: Sicherheit auf Transport- und auf Nachrichtenebene

gen werden, während vertrauliche Daten nur für den Empfänger dechiffrierbar sind. Diese Vertraulichkeit hängt dabei an der Nachricht selbst und kann damit auch bei Abspeicherung erhalten werden, da die Signatur jederzeit wieder überprüfbar ist. Bei einer Sicherung über TLS/SSL ist der Schutz dann zuende, wenn die Nachricht den gesicherten SSL-Kanal verlassen hat.

Analoge Szenarien lassen sich für andere Sicherheitsanforderungen bilden. Wenn die Integrität der Daten auf Transportebene sichergestellt wird, dürfen diese auf dem Weg zum Empfänger nicht verändert werden. Hat die Zwischenstation jedoch die Aufgabe, eingehende Nachrichten anhand der Metadaten anzupassen, bricht sie damit die Signatur der Nachricht. In diesem Fall kann keine Ende-zu-Ende-Integrität gewährleistet werden.

## Dreifachtalent SOAP-Mediator

Die im obigen Szenario erwähnte Zwischenstation könnte z. B. ein SOAP-Mediator sein. Dabei handelt es sich um einen speziellen Service, der im Allgemeinen drei verschiedene Aufgabengebiete abdecken kann.

- ▼ Zunächst kann der Mediator als SOAP-Firewall agieren und eingehende Nachrichten anhand von definierten Regeln behandeln.
- ▼ Der zweite Einsatzbereich eines SOAP-Mediators ist das Routing von Nachrichten. Anhand von in der Nachricht enthaltenen Metadaten (z. B. WS-Addressing-Informationen) leitet der Mediator diese an einen entsprechenden Zielservice weiter. Auf diese Weise kann zum Beispiel eine Lastverteilung realisiert werden, indem der Mediator dynamisch den zu der Zeit besten von mehreren gleichartigen Services auswählt. Mit dieser Funktionalität ist es möglich, eine Architektur zu entwerfen, in der die eigentlichen Services nur über den Mediator erreichbar sind. In diesem Fall muss jeder Service-Consumer seine Anfrage an den Mediator richten, welcher die Nachricht an den entsprechenden Service weiterleitet.
- ▼ Eine weitere Funktionalität eines SOAP-Mediators ist der Einsatz als Gateway. Wenn ein Service-Consumer beispielsweise lediglich HTTP beherrscht, der Service jedoch JMS-Nachrichten verlangt, konvertiert der Mediator zwischen den beiden Protokollen. Voraussetzung ist natürlich, dass er selbst beide Protokolle kennt.

## Fallstudie

Das hier beschriebene Szenario wurde in einer Fallstudie innerhalb des EU-Forschungsprojektes SENSORIA umgesetzt [SEU05]. Der in diesem Artikel beschriebene Beispielcode findet sich unter [SIGS]. Inhaltlich integriert sich diese Fallstudie in einen Geschäftsablauf einer Bank, bei der ein Unternehmen einen Kredit beantragt. Ein Teil des Prozesses, der mit der Open-Source-Engine ActiveBPEL umgesetzt wurde [ABP06], ist die Validierung der Bilanz des Kreditnehmers. Neben der Plausibilitätsprüfung der Bilanzdaten wird hierbei zusätzlich die Kredit-

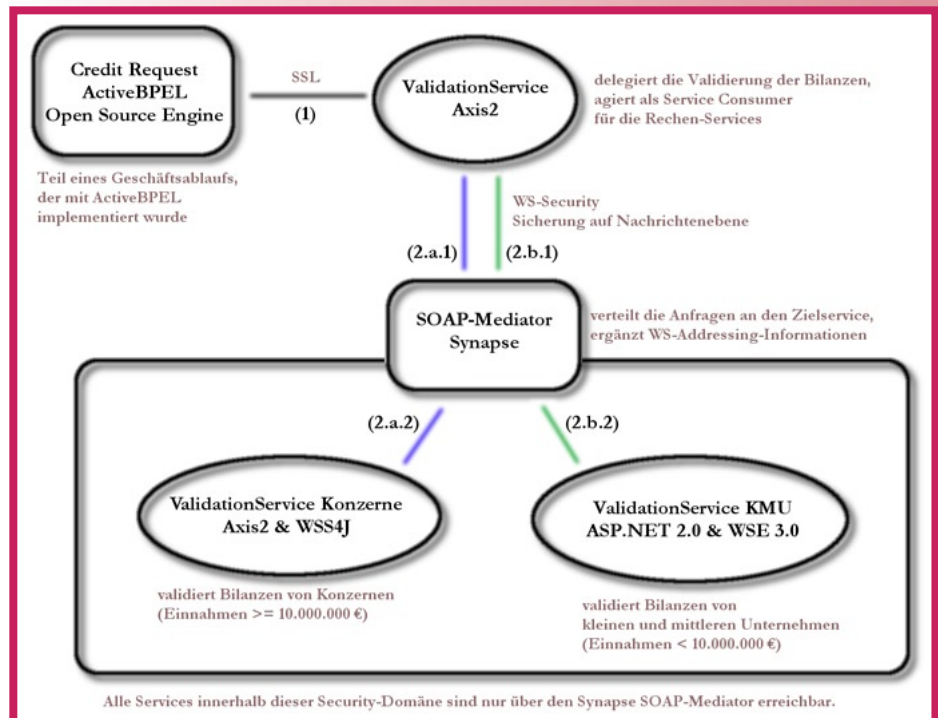


Abb. 2: Architekturskizze des Szenarios

würdigkeit des Unternehmens ermittelt. Der Service-Consumer *Credit Request* ist Teil des Geschäftsablaufs und delegiert die Bilanzprüfung an den *ValidationService*. Weil es sich dabei um einen einfachen Service-Aufruf ohne spezielle Sicherheitsanforderungen handelt, reicht es an dieser Stelle aus, die Transportebene mit SSL zu verschlüsseln (s. Abb. 2, Sequenznummer 1).

Dieser *ValidationService* führt die Bilanzprüfung allerdings nicht selbst durch. Stattdessen delegiert er die Aufgabe abhängig von der Höhe der Bilanzsumme an einen von zwei Zielservices weiter: ist diese größer oder gleich 10 Millionen Euro, wird der Kreditnehmer als Konzern eingestuft, andernfalls als kleines bis mittleres Unternehmen (KMU). Bilanzen von kleinen und mittleren Unternehmen werden vom *ValidationService für KMUs* geprüft, für Konzernbilanzen ist der *ValidationService für Konzerne* zuständig.

Diese beiden Services liegen in einer eigenen Domäne, die für den *ValidationService* nicht direkt erreichbar ist. Der einzige Weg zu den Ziel-Services führt über den SOAP-Mediator.

Analog zum oben beschriebenen Szenario muss die Kommunikation zwischen *ValidationService* und den Ziel-Services mit WS-Security abgesichert sein. Der Grund dafür ist, dass die Zwischenstation (der SOAP-Mediator) die vertraulichen Bilanzinformationen im SOAP-Body nicht lesen darf. Die Metadaten im SOAP-Header müssen jedoch für den Mediator einsehbar sein, weil darin Informationen für die Behandlung der Nachricht liegen können. Außerdem muss der Mediator in der Lage sein, den SOAP-Header zu verändern und zum Beispiel WS-Addressing-Informationen zu ergänzen. Der *ValidationService* entscheidet sich demnach für einen der beiden Ziel-Services und erstellt die Anfrage-Nachricht. Diese wird mit WS-Security verschlüsselt und an den SOAP-Mediator übermittelt (Sequenznummer 2.a.1 und 2.b.1 in Abb. 2). Der Mediator analysiert und erweitert die Nachricht, bevor er sie an den Ziel-Service weiterleitet. Weil der SOAP-Body mit dem öffentlichen Schlüssel des Ziel-Services chiffriert wurde, kann nur dieser den Inhalt der zu prüfenden Bilanz lesen.



```

<synapse xmlns="http://ws.apache.org/ns/synapse">
  <definitions>
    <sequence name="ValidationServiceCorporates">
      <!-- axis 2 service; Bilanzpruefung fuer Konzerne -->
      <header name="To"
value= "http://InternHost1:8310/axis2/services/ValidationService"/>
      <header name="Action"
value= "http://validation/Validation/validateRequest" />
    </sequence>
    <sequence name="ValidationServiceKMUs">
      <!-- .Net WSE 3.0 service; Bilanzpruefung fuer KMUs-->
      <header name="To"
value="http://InternHost2:1053/validationServiceKMU/Service.asmx"/>
      <header name="Action"
value="http://validation/Validation/validateRequest" />
    </sequence>
  </definitions>
  <rules>
    <in>
      <switch source="get-property('To')">
        <case regex=".*\/ValidationServiceCorporates.*">
          <sequence ref="ValidationServiceCorporates" />
        </case>
        <case regex=".*\/ValidationServiceKMUs.*">
          <sequence ref="ValidationServiceKMUs" />
        </case>
        <default>
          <makefault>
            <code value="tns:Receiver"
xmlns:tns="http://www.w3.org/2003/05/soap-envelope" />
            <reason value="No matching service found!" />
          </makefault>
        </default>
      </switch>
    </in>
    <send />
  </rules>
</synapse>

```

Listing 1: synapse.xml

## Apache Synapse als SOAP-Mediator

Für die Fallstudie wurde Apache Synapse (Milestone 2) als SOAP-Mediator gewählt [ASP06]. Das Projekt hat gerade den Sprung vom Apache Incubator in das Webservices-Projekt der Apache Software Foundation geschafft. Es liegt aktuell in Version 0.91 vor.

Synapse erlaubt die Prüfung der eingehenden Nachrichten mit Hilfe von regulären Ausdrücken. Mit dieser Funktionalität kann es die Rolle einer SOAP-Firewall übernehmen. Gleichzeitig können damit aber auch Regeln für das Routing der Nachrichten implementiert werden. In dieser Fallstudie wird lediglich anhand der URL geprüft, an welchen Ziel-Service die Nachricht weitergeleitet wird (s. Listing 1, Tag `switch-case`). Wenn in der Anfrage-URL vom *ValidationService* die Zeichenfolge *ValidationServiceCorporates* vorkommt, wird die Nachricht an den *ValidationService für Konzerne* weiter geleitet (Listing 1, Tag `case-sequence`). Synapse belässt dabei den (mit WS-Security signierten und verschlüsselten) SOAP-Body unverändert. Lediglich der SOAP-Header wird um WS-Addressing-Informationen erweitert. Wenn kein entsprechender Ziel-Service gefunden wird, liefert Synapse einen SOAP-Fault an den *ValidationService* zurück.

Durch diese Architektur kann Synapse als zentrale Verteilstation für einen geschützten Bereich eingesetzt werden. Wenn sich die Adressen der Ziel-Services ändern, müssen nicht mehr alle Service-Consumer umgestellt werden. Stattdessen können derartige Änderungen in der *synapse.xml* (s. Abb.2 und Listing 1)

umgesetzt werden. Auch wenn Sonderbehandlungen in serviceorientierten Architekturen nicht gern gesehen sind, kommen diese in der Praxis dennoch an manchen Stellen vor. Insbesondere wenn mehrere verschiedene Plattformen eingesetzt werden, kommt es unter Umständen zu Effekten, die eine spezielle Behandlung erfordern. Beim Einsatz eines zentralen Mediators bietet sich die Möglichkeit, diese Spezialfälle von diesem behandeln zu lassen.

## Theorie und Praxis

Neben der Verwendung einer Zwischenstation in der Kommunikation enthält dieses Szenario eine weitere Besonderheit: die Interoperabilität von heterogenen Webservice-Umgebungen. Während der verteilende *ValidationService* sowie der *ValidationService für Konzerne* auf Axis2 basieren, läuft der *ValidationService für KMUs* unter ASP.NET 2.0. Weil ein gemeinsamer Nenner gefunden werden muss, wird die Konfiguration von WS-Security entsprechend komplexer. Die Axis2-Services verwenden das Rampart-Modul (WSS4J) für WS-Security [ARM06]. Unter ASP.NET kommen die Web Service Enhancements (WSE 3.0) zum Einsatz [MWC06].

Plattformunabhängigkeit ist definitionsgemäß eine Kernanforderung an serviceorientierte Architekturen, dies gilt genauso für WS-Security. Hier – wie auch bei anderen WS-Standards – gibt es Spezifikationen, die sich die Interoperabilität besonders auf die Fahne geschrieben haben. Hervorzuheben ist dabei das Basic Security Profile der WS-I [BSP06]. Wenn sich beide verwendeten Plattformen gleichermaßen korrekt an solche Spezifikationen hielten, gäbe es auch wenig Probleme bei der Interoperabilität. In der Praxis bestehen allerdings noch einige Ungeheimheiten, die zum Teil auf Fehler („Kinderkrankheiten“) in der Implementierung der WS-Security-Frameworks zurückzuführen sind, zum Teil aber auch auf Spielräume in den Spezifikationen. So machen die genannten Spezifikationen z. B. keine verpflichtenden Aussagen über die Werte und Generierungsalgorithmen von Sicherheits-Token, sodass hier die Eindeutigkeit selbst bei Frameworks, die explizite BSP-Unterstützung anbieten (wie Rampart und WSE), nicht garantiert werden kann. In homogenen Szenarien werden darauf beruhende Interoperabilitätsprobleme in der Regel nicht sichtbar: Wenn sich Client und Service in gleicher Weise nicht an die Spezifikation halten, gibt es kein Problem zwischen diesen beiden.

Im Rahmen der heterogenen Frameworks in der Fallstudie wurden die Schwierigkeiten jedoch sichtbar. An dieser Stelle werden nicht alle detailliert analysiert. Die Entwicklung der einzelnen Frameworks ist zurzeit noch so schnell, dass viele Anleitungen bald überholt sind. Wir konzentrieren uns deshalb nur auf einige Aspekte, die sich wiederholt als problematisch erwiesen haben. Als Beispiel dient der Fall, in dem Axis2 als Client und ASP.NET als Service für KMUs agiert. Die Erläuterungen lassen sich aber auch analog auf andere – homogene und heterogene – Szenarien anwenden.

## Die wichtigsten Fallstricke

Eine wichtige Rolle spielt die WSS4J-Konfiguration in Listing 2: Dieser Auszug definiert die WS-Security-Parameter für Nachrichten an den *ValidationService für KMUs*. Wenn der *ValidationService* eine Anfrage an diesen richtet, wird der SOAP-Body beispielsweise vom *ValidationService* signiert. Anschließend wird sie mit dem öffentlichen Schlüssel des *ValidationService für KMUs* chiffriert und ein Zeitstempel hinzugefügt (s. Listing 2,

```

<parameter name="OutflowSecurity">
  <action>
    <items>Signature Encrypt Timestamp</items>
    <user>validationService</user>
    <encryptionUser>validationServiceKmus</encryptionUser>
    <passwordCallbackClass>PWCallback</passwordCallbackClass>
    <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
    <encryptionKeyIdentifier>SKIKeyIdentifier</encryptionKeyIdentifier>
    <signaturePropFile>clientSecurity.properties</signaturePropFile>
    <encryptionPropFile>clientSecurity.properties</encryptionPropFile>
  </action>
</parameter>

```

Listing 2: WS-Security-Konfiguration für Anfragen an den ValidationService für KMUs

Zeile 3 <items>-Tag). Die `.properties`-Datei enthält Informationen über den Zugriff auf den Zertifikatsspeicher, in dem u. a. der öffentliche Schlüssel liegt.

Damit der Ziel-Service die Nachricht entschlüsseln und die enthaltene Signatur verifizieren kann, muss er der Nachricht Informationen darüber entnehmen können, welche Zertifikate für die Verschlüsselung und die Signatur verwendet wurden. Daher integriert der *ValidationService* Verweise in den SOAP-Header, um eine Beziehung zwischen den signierten Bereichen und seinem Zertifikat herzustellen. Grundsätzlich kann dabei zwischen zwei Methoden unterschieden werden:

- ▼ Integration des gesamten Zertifikats in die Nachricht (*DirectReference*) oder
- ▼ Verweis auf das verwendete Zertifikat (*IssuerSerial*, *X509KeyIdentifier*, *SKIKeyIdentifier*).

Damit der Ziel-Service die digitale Signatur des Service-Consumers validieren kann, wird das Zertifikat des *ValidationService* in die Nachricht integriert. Weil es den Ziel-Services im Vorfeld nicht bekannt ist, muss das Zertifikat als **DirectReference** in die Nachricht integriert werden (s. Listing 2, Parameter **signatureKeyIdentifier**). Mit der Nachricht wird also das gesamte Zertifikat als binärer Token an den Ziel-Service gesendet. Dieser extrahiert das Zertifikat und prüft, ob es von einer vertrauenswürdigen Zertifizierungsstelle signiert wurde.

Wenn der Empfänger der Nachricht das Zertifikat bereits kennt, muss es nicht mehr verschickt werden. Weil für den Ziel-Service erkennbar sein muss, welche Teile für ihn dechiffrierbar sind, wird dessen Zertifikat in der Nachricht referenziert. Mit welcher der oben genannten Methoden dies geschieht, wird durch den Parameter **encryptionKeyIdentifier** festgelegt. Der Ziel-Service kennt natürlich sein eigenes Zertifikat, daher empfiehlt sich eines der Verfahren, in dem nur ein Verweis auf dieses eingefügt wird.

Welche der Methoden verwendet wird, um das Zertifikat des Empfängers für die Verschlüsselung zu referenzieren, sollte eigentlich keine Rolle spielen. In homogenen Umgebungen ist dies auch tatsächlich kein Problem. Genau dies erweist sich jedoch häufig als die größte Hürde, eine interoperable WS-Security-Lösung zu konfigurieren.

Aufgrund eines mittlerweile von Microsoft bestätigten Fehlers in WSE 3.0, durch den die XML-Namespaces der Elemente falsch behandelt werden, kann das **IssuerSerial**-Verfahren in heterogenen Umgebungen nicht verwendet werden [MWE06]. Auch das Verfahren namens **X509KeyIdentifier** ist nur in homogenen Szenarien verwendbar, da WSE und WSS4J es auf verschiedene Weisen behandeln. Die Lösung des Problems ist die Verwendung eines **SKIKeyIdentifier**. Nur mit diesem können sich beide Plattformen auf einen gemeinsamen Nenner für die Integration der Zertifikate einigen.

Erstaunlich ist dabei, dass auch die Integration des Zertifikats als **DirectReference** versperrt ist. In diesem Fall integriert

der *ValidationService* das Zertifikat des Ziel-Service komplett in die Nachricht. Beim Aufruf des *ValidationService für KMUs* gibt WSE daraufhin an, keinen Zugriff auf seinen eigenen privaten Schlüssel zu haben.

Auf Seiten des *ValidationService für KMUs* wird WS-Security an zwei Stellen konfiguriert. In der Anwendungskonfiguration (z. B. **web.config**) müssen die verwendeten Algorithmen für die Verschlüsselung und Signatur definiert werden. Diese Deklaration sollte eigentlich für das Entschlüsseln und Verifizieren der Nachricht überflüssig sein, weil diese Informationen bereits in der Nachricht definiert sind. WSE verweigert andernfalls jedoch die Bearbeitung der eingehenden Nachrichten. Die eigentliche Konfiguration der Aktionen und zu verwendenden Zertifikate für WS-Security erfolgt in WSE 3.0 durch eine Policy (s. Listing 3). Wichtig ist hierbei die **messageProtectionOrder** (hier: **SignBeforeEncrypt**) und die Angabe des zu verwendenden Zertifikats.

Neben der Zertifikat-Behandlung ist die Konfiguration der Kommunikation ein weiteres Problem. Standardmäßig werden Nachrichten von Axis2 im *chunked*-Modus gesendet [HCE06]. Dieses Verfahren ist insbesondere für große Nachrichten sinnvoll, die nicht als Ganzes, sondern in kleinen Teilen – den *Chunks* – verschickt werden. Leider ist der eingesetzte ASP.NET 2.0 Service (*ValidationService für KMUs*) nicht in der Lage, chunked-kodierte Nachrichten zu interpretieren.

An dieser Stelle kann Synapse als Gateway verwendet werden, indem es zwischen den beiden verwendeten Modi konvertiert. Der Mediator empfängt die Anfrage vom *ValidationService* im chunked-Modus und leitet sie im normalen HTTP-Transfermodus an den Ziel-Service weiter.

## Fazit und Ausblick

Die erweiterten WS-Spezifikationen wie WS-Security sind kein reiner Selbstzweck. Während in vielen Fällen die bewährten Sicherheitsmaßnahmen auf Transportebene ausreichen, muss die Absicherung in komplexeren Szenarien zwingend innerhalb der Nachrichten erfolgen. Die vorgestellte Fallstudie ist ein Beispiel dafür.

Bei der Implementierung zeigten sich einige Stolperfallen auf dem Weg zur Interoperabilität von WS-Security-Frameworks. Es ist in diesem Umfeld nicht unüblich, dass manche

```

<policy name="x509">
  <mutualCertificate10Security establishSecurityContext="false"
    renewExpiredSecurityContext="false"
    requireSignatureConfirmation="false"
    messageProtectionOrder="SignBeforeEncrypt"
    requireDerivedKeys="false" ttlInSeconds="300">
    <serviceToken>
      <x509 storeLocation="CurrentUser" storeName="My"
        findValue="CN=MP Service dotNet, O=SN, S=NRW, C=DE"
        findType="FindBySubjectDistinguishedName" />
    </serviceToken>
    <protection>
      <request signatureOptions="IncludeSoapBody"
        encryptBody="true" />
      <response signatureOptions="IncludeSoapBody"
        encryptBody="true" />
      <fault signatureOptions="IncludeSoapBody"
        encryptBody="false" />
    </protection>
  </mutualCertificate10Security>
  <requireActionHeader />
</policy>

```

Listing 3: WSE 3.0 Policy des ValidationService für KMUs



Probleme erst nach dem Durchprobieren aller Möglichkeiten gelöst werden können. Als besonders anfällig erweist sich die Verwaltung der Zertifikate. Es ist deshalb naheliegend, einen Standard wie WS-Trust zu betrachten, der gerade die Zertifikate in seinen Fokus stellt. Jedoch auch ohne WS-Trust sollten die skizzierten Beispiele es in Zukunft einfacher machen, Sicherheit in heterogenen Umgebungen zu erreichen.

## Literatur und Links

[ABP06] Active BPEL, <http://www.activebpel.org>  
 [ARM06] Apache, Rampart, [http://ws.apache.org/axis2/modules/rampart/1\\_1/security-module.html](http://ws.apache.org/axis2/modules/rampart/1_1/security-module.html)  
 [ASP06] Apache, Synapse, <http://ws.apache.org/synapse>  
 [BSP06] Web Services Interoperability Organization, Basis Security Profile, <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>  
 [BUM02] XML-Security, <http://www.cs.fhm.edu/~koehler/vortraegeSS02/FHMXMLSecurity.pdf>  
 [HCE06] HTTPWatch, <http://www.httpwatch.com/httpgallery/chunked>  
 [MWC06] Windows Communication Foundation, <http://wcf.netfx3.com>  
 [MWE06] Microsoft, Knowledge Base, <http://support.microsoft.com/kb/922779>  
 [OWS04] OASIS Web Services Security (WSS), <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>  
 [SEU05] Sensoria, Software Engineering for Service-Oriented Overlay Computers, <http://www.sensoria-ist.eu>  
 [SIGS] Beispielcode, <http://www.sigs-datacom.de/sd/publications/js/2007/04/index.htm>



**Michel Alessandrini** ist an der FHDW als wissenschaftlicher Mitarbeiter mit den Forschungsschwerpunkten SOA und künstlicher Intelligenz tätig. Des Weiteren ist er für die S&N AG als Projektleiter in einem EU-Forschungsprojekt im Einsatz.  
E-Mail: [michel.alessandrini@fhdw.de](mailto:michel.alessandrini@fhdw.de).

**Dr. Willi Nüßer** ist seit 2002 Professor für angewandte Informatik an der Fachhochschule der Wirtschaft (FHDW) in Paderborn und Inhaber der Heinz Nixdorf Stiftungsprofessur. Er war von 1996 bis 2002 bei der SAP AG u. a. im LinuxLab in Walldorf tätig. Seit 2002 beschäftigt er sich in Theorie und Praxis mit der Webservice-Technologie.  
E-Mail: [wilhelm.nuesser@fhdw.de](mailto:wilhelm.nuesser@fhdw.de).

**Michael Pollmeier** ist Absolvent der FHDW. Er ist bei der S&N AG, Paderborn im Bereich des Designs, der Entwicklung und der Sicherung von Webservices tätig. E-Mail: [pollmeier@gmx.de](mailto:pollmeier@gmx.de).



## Weitere Informationsquellen

<http://www.sigs-datacom.de/sd/publications/js/2007/04/index.htm>