International Journal of Web Services Research , Vol.X, No.X, 200X

# Interoperability and Functionality of WS-* Implementations

Andreas Schönberger[1], Johannes Schwalb[2] and Guido Wirtz[1]

[1]Distributed Systems Group
University of Bamberg
Bamberg, Germany
{andreas.schoenberger | guido.wirtz}@uni-bamberg.de

[2]Johannes Schwalb
Senacor Technologies AG
Schwaig b. Nürnberg, Germany
{johannes.schwalb}@senacor.com

**ABSTRACT:**

Recently, the Web Services Interoperability Organization (WS-I) has announced to have completed its interoperability standards work. The latest deliverables include the so-called "Basic Security Profile" and the "Reliable Secure Profile". This gives rise to the question whether or not Web Services adopters can rely on interoperability and functionality of Web Services stacks, in particular in terms of security and reliability features. To answer this question, we thoroughly analyze two important Web Services stacks for interoperability of WS-Security and WS-ReliableMessaging features. Our analysis shows that security and reliability features are far from being implemented in an interoperable manner. Additionally, we reveal that some of those interoperability problems are not even covered by WS-I profiles and therefore conclude that WS-I's work has not yet resulted in Web Services interoperability. Finally, we investigate support for the so-called "Secure WS-ReliableMessaging Scenario" in order to find out whether WS-* adopters can at least rely on the availability of real-world functionality in homogeneous environments.

**KEY WORDS:**
*Web Services Interoperability; Interoperability Testing; WS-Security; WS-ReliableMessaging; Secure WS-ReliableMessaging Scenario*

# INTRODUCTION

Quality-of-Service (QoS) features such as security and reliability are brought to the Web Services world by the so-called *WS-** standards. *WS-Security 1.1* (WS-Sec, OASIS 2006) and *WS-ReliableMessaging 1.2* (WS-RM, OASIS 2009a) are prominent representatives of WS-* standards that define data formats and processing instructions for extending the *SOAP* (W3C 2007) messages that implement Web Services exchanges. For example, an XML `Signature` tag together with `SignedInfo`, `SignatureValue` and `KeyInfo` tags would have to be inserted into the SOAP `Header` tag to provide integrity protection. For convenience, a Web Services developer is not supposed to '*manually*' insert all that information into SOAP messages. Instead, *Web Services Security Policy 1.3* (WS-Sec-Pol, OASIS 2009b) and *Web Services Reliable Messaging Policy Assertion 1.2* (WS-RM-Pol, OASIS 2009c) can be used to extend the WSDL definition of a Web service with assertions that instruct the Web Services stack implementations (WS stack in the following) in use to apply WS-Sec and WS-RM features to the

SOAP message exchanges. This policy-based realization of QoS features for Web Services has been postulated in several publications (Martino and Bertino 2009; Gruschka, Luttenberger et al. 2006; Gruschka, Jensen et al. 2007; Curbera, Khalaf et al. 2008; Khalaf, Keller et al. 2006; Nezhad, Benatallah et al. 2006; Menzel, Warschofsky et al. 2010a) and promises better interoperability than letting all Web Services developers implement QoS features *their own way*.

On November 10th 2010, the *Web Services Interoperability Organization* (WS-I) released a press announcement with the following title:

"WS-I Completes Web Services Interoperability Standards Work
*Industry Collaboration Enables Interoperability in the Cloud*"[1]

This announcement followed the release of new versions of WS-I's core deliverables, most notably the *Basic Security Profile* (BSP, WS-I 2010a) and *Reliable Secure Profile* (RSP, WS-I 2010b) in 2010. WS-I *profiles* are the core deliverables of WS-I and document *"[..] clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability"* (WS-I 2010b). The main target of BSP and RSP are WS-Sec and WS-RM. As WS-I promotes interoperability and has declared its standardization work to be completed, this means that Web Services adopters *should* be able to rely on the interoperable implementation of Web Services security and reliability features which are pivotal for Web Services according to (Nezhad, Benatallah et al. 2006; Moser, Melliar-Smith et al. 2007). However, WS-Sec and WS-RM as well as WS-Sec-Pol and WS-RM-Pol are highly complex specifications so that interoperability across WS stacks does not come easy. Consistently, (Martino and Bertino 2009) stress that *"although one of the main purposes of the standard [i.e., a WS security standard] is to guarantee the interoperability between different platforms, it might be necessary to test it on the field."*

This paper investigates in how far WS-I's work has led to interoperability across WS stacks regarding the implementation of WS-Sec(-Pol) and WS-RM(-Pol) and whether real-world functionality is available. In order to operationalize the question whether or not a Web Services adopter can rely on interoperability between different WS stacks, we use two 'optimistic' hypotheses:

**H1:** *The overwhelming majority of WS-Sec-Pol/WS-RM-Pol features are implemented by Web Services stacks.*
**H2:** *Out of those WS-Sec-Pol/WS-RM-Pol features implemented by two platforms, the overwhelming majority is implemented in an interoperable manner.*

For analyzing availability of real-world functionality, support of the so-called *Secure WS-ReliableMessaging Scenario* (SecRM scenario, Gavrylyuk, Hrebicek et al. 2005; Backes, Mödersheim et al. 2006) is investigated. The SecRM scenario describes the WS-Sec and WS-RM based implementation of important security and reliability properties that have been identified as crucial for integration scenarios (Schönberger, Wirtz et al. 2010) and therefore provides a good benchmark for real-world functionality. In anticipation of the interoperability results of section *INTEROPERABILITY ASSESSMENT*, the SecRM scenario is tested in homogeneous environments only.

This paper is an extended version of (Schönberger, Schwalb et al. 2011) and the main new content is the analysis of the SecRM scenario presented in section *FUNCTIONALITY*

[1] http://ws-i.org/docs/press/pr_101110.pdf, 08/30/2011

*ASSESSMENT*. The contribution described in this paper is fourfold where the first three contributions originally have been published in (Schönberger, Schwalb et al. 2011). First, we assess the coverage of WS-Sec-Pol/WS-RM-Pol specifications by two major Java-based WS stacks. Second, we assess the interoperability of implemented features. Based on these results, both hypotheses (**H1**, **H2**) will have to be rejected. Third, we analyze in how far the detected interoperability issues could have been avoided by strict compliance to the BSP and RSP. Fourth, we present a WS-Policy operationalization of the SecRM scenario and we show that this advanced real-world WS-* functionality is available at least for one platform.

In section *TEST APPROACH*, the notion of interoperability is operationalized for the work at hand and the approach towards interoperability testing WS-* standards is described. Section *INTEROPERABILITY ASSESSMENT* presents the results of our interoperability and coverage investigation while section *WS-I TO THE RESCUE?* analyzes in how far strict compliance to WS-I's BSP/RSP could have been of help. In section *FUNCTIONALITY ASSESSMENT*, the definition of the SecRM scenario is introduced, a WS-Policy definition for implementing it is given and the test results for SecRM support are presented. Section *RELATED WORK* discusses related work and section *CONCLUSION AND FUTURE WORK* concludes and points out directions for future work.

# TEST APPROACH

In order to provide a sound foundation for this work, we sketch our approach for testing WS-* interoperability (Schwalb, Schönberger et al. 2010; Schwalb and Schönberger 2010) by operationalizing the notion of interoperability, describing a concept for executing test cases and outlining the systematic derivation of test cases.

(Wegner 1996) defines *'interoperability'* as the *"the ability of two or more software components to cooperate despite differences in language, interface, and execution platform."* While this definition is good enough for an abstract characterization of interoperability in arbitrary systems, it has to be refined for the purpose of WS-* interoperability testing. Remember that we require the use of WS-Policy for asserting QoS properties of Web Services interactions. Hence, the following sources of interoperability issues between two WS stacks have to be considered. First, one of the WS stacks under test may not know/refuse a particular WS-Policy assertion that specifies a particular communication feature. Second, one of the WS stacks may accept a WS-Policy assertion, but ignore it. Third, a WS stack may deviate from one or more of the processing instructions that are specified by a WS-* standard for the implementation of a particular WS-Policy assertion. Considering these sources of interoperability issues and taking into account that a Web service interaction typically takes place between a client role and a server role, 12 interoperability levels can be identified that range from a policy being refused/ignored by one of the roles over abrupt termination of communication to full protocol success (for details, please see Schwalb, Schönberger et al. 2010). So, for the purpose of this work, interoperability is defined as follows:

*Definition 1 (Interoperability):*
*Two WS stacks are interoperable with respect to a WS-* policy assertion if client and server process the assertion such that the exchange of corresponding SOAP messages succeeds without errors and such that WS-* processing rules are applied.*

For determining interoperability as defined above, the approach visualized in Figure 1 is applied. WS-Sec-Pol/WS-RM-Pol definitions are used to specify concrete test cases. For each test case, the WSDL of a sample Web service is extended with such a definition to be used by the WS stacks of the Web service client and provider for determining the number, sequence and contents of the SOAP messages to be exchanged (upper part of Figure 1). The test case is then performed for four different WS stack configurations. Assume that two WS stacks A and B are to be tested for interoperability and that a configuration 'X-Y' expresses that WS stack X takes the client role and WS stack Y takes the server role. Then, the test case is first performed in homogeneous environments (A-A, B-B) for checking whether or not the functionality is implemented and afterwards in heterogeneous environments (A-B, B-A) for checking interoperability. Note that if only one of the homogeneous environments does not work, then the heterogeneous environments still are worth testing. Our practical tests show (cf. Schwalb and Schönberger 2010) that some features do not work in a homogeneous environment (A-A or B-B), but in a heterogeneous one (B-A or A-B).

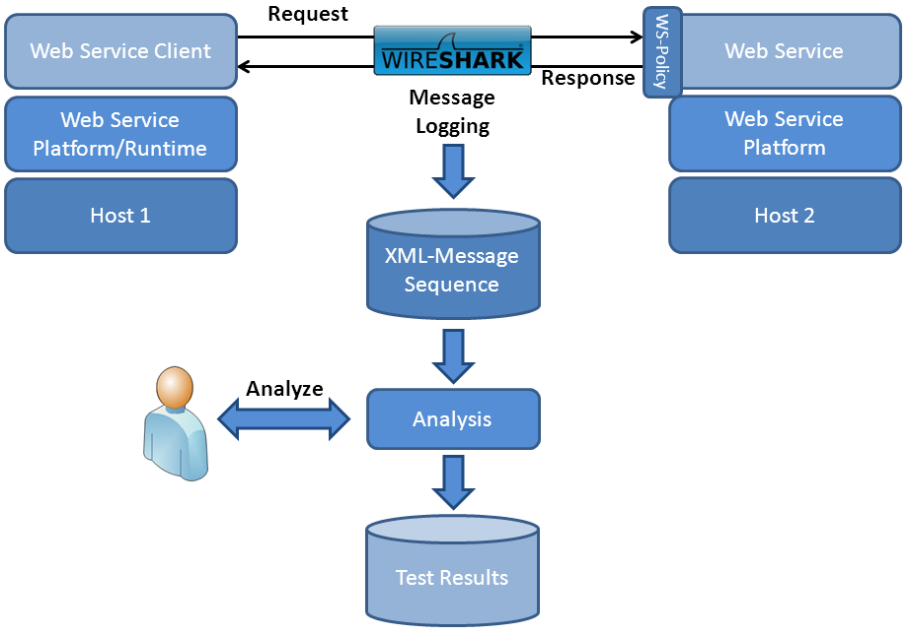


**Figure 1: Setup of Test Environment**

The interoperability levels are to be examined for each test case and WS stack configuration. Some of the interoperability levels can be verified without investigating the SOAP messages exchanged, e.g., refusal of the policy by the server. The analysis and determination of other interoperability levels require the use of network analysis tools like Wireshark[2] that enable capturing the SOAP messages exchanged (lower part of Figure 1). However, we do not check the strict conformance of SOAP messages to WS-* standards in our interoperability testing approach. Instead, SOAP messages are only analyzed for the existence of WS-* headers as well as for unexpected errors and premature termination. Not checking conformance allows for the possibility of 'interoperable' communication that violates WS-* standards. Consistently, definition 1 deliberately does not require that WS-* processing rules are applied correctly. From our experience, this is a purely theoretical limitation for heterogeneous environments.

[2] www.wireshark.org, 08/30/2011

For deriving the configuration options for a single policy assertion, we propose to make use of the assertion structure definitions that are published in the WS-Policy extension standards. Listing 1 shows the structure definition of the WS-RM-Pol standard's `RMAssertion` assertion (note that usual regular expression operators are used to define structural constraints on the assertion). This assertion basically says that delivery semantics options `ExactlyOnce`, `AtLeastOnce` and `AtMostOnce` of WS-RM must be combined with either `InOrder` delivery or not. Checking these features in isolation frequently is not possible because a deployable WS-Policy configuration may require additional assertions, e.g., testing a WS-Sec-Pol *protection* assertion requires declaring assertions for a valid *security binding* (options for so-called `Asymmetric-/Symmetric-/TransportBindings`). To solve this issue, we propose to start with sample policy configurations that ship with WS stacks or are retrievable from the web and then to permute the options of the assertion under test only. By using the WS-Policy structure definitions and sample policy configurations as proposed, it is possible to identify test cases that test a WS-* feature almost in isolation and to achieve reasonable coverage of the WS-Policy standards. Based on the results of these *"isolated"* test cases, *"combined"* test cases that cover the interplay of several WS-* features can be derived.

**Listing 1. Structure Definition of RMAssertion (cf. OASIS 2009c)**

```
1 <wsrmp:RMAssertion (wsp:Optional="true")?
        ...>
2       <wsp:Policy>
3           (<wsrmp:SequenceSTR/> |
4           <wsrmp:SequenceTransportSecurity/> ) ?
5
6           <wsrmp:DeliveryAssurance>
7               <wsp:Policy>
8                   (<wsrmp:ExactlyOnce/> |
9                   <wsrmp:AtLeastOnce/> |
10                  <wsrmp:AtMostOnce/> )
11                  <wsrmp:InOrder/> ?
12              </wsp:Policy>
13          </wsrmp:DeliveryAssurance> ?
14      </wsp:Policy>
15      ...
16 </wsrmp:RMAssertion>
```

# INTEROPERABILITY ASSESSMENT

For evaluating WS-Sec-(Pol)/WS-RM-(Pol), we have chosen two of the most reputable JAVA-based WS stacks, namely Oracle's (Sun's) *Metro* WS-stack that comes with the GlassFish Application Server[3] and Apache's *Axis2* WS stack as reused in IBM's WebSphere Application Server[4]. Below, we show that considerable interoperability problems between these WS stacks exist which justify rejecting both hypotheses **H1** and **H2**.

[3] http://glassfish.dev.java.net, 08/30/2011
[4] http://www-01.ibm.com/software/webservers/appserv/was/, 08/30/2011

The following groups of *isolated* test cases have been identified for WS-Sec-(Pol) and WS-RM-(Pol). In anticipation of the results, *combined* test cases are left out:

1) *WS-RM-Pol Assertions*
   This test group essentially comprises the test cases derivable from listing 1 and cover the delivery semantics of reliable messaging.
2) *WS-Sec-Pol Protection Assertions*
   This test group covers the various ways of asserting the need for signing or encrypting SOAP messages or parts of SOAP messages.
3) *WS-Sec-Pol Tokens*
   This test group covers assertions for configuring how security tokens are processed, e.g., whether or not tokens have to be included in every SOAP message, and the assertions for declaring the various token types themselves such as `UsernameTokens` or `X509Tokens`.
4) *WS-Sec-Pol Security Bindings*
   Security binding assertions configure the algorithms to be used for signing/encrypting as well as options for configuring the basic security mechanisms *transport level security*, *symmetric key security* as well as *asymmetric key security*.
5) *WS-Sec-Pol Supporting Tokens*
   This test group covers assertions that are used to augment the claims provided by the token of the basic Security Binding. For example, an `EndorsingSupporting Tokens` assertion may be used to require a signature of the signature of a SOAP message (cf. OASIS 2009b, section 8.3).
6) *WS-Sec-Pol WS-Sec and WS-Trust Options*
   This test group covers general WS-Sec and WS-Trust assertions such as what kind of token references must be supported, whether client or server challenges must be supported, or whether client or server entropy is required.

All in all, the number of test cases derived amounted to 169. This number proved to still be manageable. For the majority of test cases, it was possible to retrieve executable sample configurations for at least one of the WS stacks from the web. Executable sample configurations for the remaining test cases then could be derived by just replacing or reconfiguring an assertion, e.g., using a 'WssX509V3Token11' instead of a 'WssX509V3Token10'. 109 of the test cases could successfully be performed in at least one of the homogeneous environments. This fact taken together with the exception messages of the WS stacks under test about not supporting particular features indicates that our policy configurations in itself were correct (in the sense of complying to WS-Sec-Pol/WS-RM-Pol) for most test cases and therefore not the source of the detected interoperability problems. In the following, section *Core Interoperability Issues* describes the core issues detected and section *Overall Results* summarizes the interoperability results per group of test cases.

## Core Interoperability Issues

In order to protect solution provider interests we have made the following interoperability issues anonymous (more detailed test results are available as a technical report in Schwalb and Schönberger 2010) and stick to the A,B-notation of section *TEST APPROACH*:

1) *No WS-ReliableMessaging Policy Support*
   Platform A uses a proprietary API for configuring reliable messaging features that is accessible via a GUI and does not accept WS-RM-Pol for configuration. So, if platform

A is used for the client, then no interaction is possible. However, if platform A is used for the server then platform B can be configured using WS-RM-Pol such that interaction is possible for some test cases.

2) *No* `TransportBinding` *Support*
Platform A does not support the `TransportBinding` assertion as defined in WS-Sec-Pol (OASIS 2009b, section 7.3). This assertion allows for configuring the use of transport protocol features for securing messages, in particular using HTTPS. Note that this does not mean that platform A does not support HTTPS at all, it just means that the `TransportBinding` assertion cannot be used.
3) *No* `XPath` *Support for Element Identification*
The `EncryptedElements` assertion (OASIS 2009b, section 4.2.2) and the `SignedElements` assertion (OASIS 2009b, section 4.1.2) of WS-Sec-Pol define an XML tag named XPath for specifying the elements of a SOAP message to be encrypted/signed. However, platform B does not support this tag.
4) *No* `OnlySignEntireHeadersAndBody` *Support*
This optional WS-Sec-Pol element (OASIS 2009b, section 7.4, 7.5) enforces that *"[..] digests over the SOAP body and SOAP headers MUST only cover the entire body and entire header elements"* (OASIS 2009b, section 6.6). Although the WS-Sec-Pol standard explicitly recommends to use this element in order to *"[..] combat certain XML substitution attacks"* (OASIS 2009b, section 12), platform A does not support it.
5) *No* `EncryptBeforeSigning` *Support*
This optional WS-Sec-Pol element (OASIS 2009b, section 7.4, 7.5) can be used to override the default value `SignBeforeEncrypting` of the protection order property (OASIS 2009b, section 6.3). However, only platform B supports this element.
6) *Deviating Processing of* `UsernameToken`
The WS-Sec-Pol `UsernameToken` assertion can be used to leverage username/password authentication for interactions and version 1.0 of the so-called UsernameToken profile (OASIS 2004) is accepted by both platforms. However, platform A leaves the `Username` and `Password` elements empty whereas platform B by default encrypts the whole `UsernameToken`. Platform A essentially does not allow for configuring `UsernameTokens` using WS-Sec-Pol whereas platform B disregards the following WS-Sec-Pol recommendation by applying encryption by **default**: *"When the UsernameToken is to be encrypted it SHOULD be listed as a SignedEncryptedSupportingToken (Section 8.5), EndorsingEncryptedSupportingToken (Section 8.6) or SignedEndorsingEncryptedSupportingToken (Section 8.7)"* (OASIS 2009b, section 5.4.1).
7) *Deviating Signing Strategy for Timestamp*
The optional WS-Sec-Pol `IncludeTimestamp` element (OASIS 2009b, section 7.3, 7.4, 7.5) can be used to require the inclusion of a `Timestamp` element in the SOAP headers of an interaction and is supported by both platforms. Additionally, WS-Sec-Pol requires that if `IncludeTimestamp` is specified and if there is no transport layer encryption specified then the `Timestamp` has to be integrity protected at the message level, i.e., signed (OASIS 2009b, section 6.2). However, platform A does not directly implement this rule but requires the Web Services developer to add an according `SignedElements`/`XPath` expression to sign the timestamp.
8) *Ignored* `IncludeToken` *Values*
The optional attribute `IncludeToken` allows for specifying in which SOAP messages of an interaction a corresponding token, e.g., a `UsernameToken`, should be present. For example, the `IncludeToken` value `AlwaysToRecipient` specifies that a token

should be present in all messages from the initiator of the interaction to the recipient, but not vice versa. Alternative values are `Never`, `Once`, `AlwaysToInitiator` and `Always` (OASIS 2009b, section 5.1.1). Both platforms under test accept all `IncludeToken` values, but platform A simply ignores the actual value and always includes the corresponding token in the SOAP messages. This leads to an interoperability error with platform B in case `Never` is configured even in the A-B configuration because platform B rejects SOAP messages that carry a token if `Never` is specified. Inconsistently, platform B does not stop communication if SOAP messages carry a token that is not permitted due to the `AlwaysToRecipient`/`AlwaysToInitiator` values.

9) *Deviating Processing of* `SignedParts/Body`
The WS-Sec-Pol `SignedParts` assertion (OASIS 2009b, section 4.1.1) can be used to specify the integrity protection of a SOAP message's `Header`, `Body` or `Attachment` parts. Both platforms under test support the optional element `Body` which requires that *"[..] the soap:Body element, its attributes and content, of the message needs to be integrity protected"* (OASIS 2009b, section 4.1.1). However, platform B signs the first element of the SOAP message `Body` whereas platform A signs the `Body` element itself.
10) *Deviating* `EncryptedParts` *Implementation*
Both platforms support this WS-Sec-Pol assertion (OASIS 2009b, section 4.2.1) that can be used to specify the encryption of a SOAP message's `Header`, `Body` or `Attachment` parts. However, platform B only supports the use of `EncrpytedParts` for the `IncludeToken` value `Never`. So, if platform B is used as a server, an interoperability issue arises because platform A *always* (cf. issue 8) includes a token which is rejected by the first platform.

## Overall Results

Due to space limitations, we only present the most interesting figures of the interoperability tests. The interoperability levels detected for all test cases are available as a technical report (Schwalb and Schönberger 2010). In Table 1, the column headers provide the following information:

a) # counts the number of test cases per test group
b) A-A (B-B) counts the number of test cases per test group for which full interoperability could be detected with platform A (B) as both, client and server.
c) A-A $\wedge$ B-B counts the number of test cases per test group for which full interoperability could be detected for both homogeneous environments.
d) A-A $\vee$ B-B counts the number of test cases per test group for which full interoperability could be detected for at least one of the homogeneous environments.
e) A-B $\vee$ B-A counts the number of test cases per test group for which full interoperability could be detected for at least one of the heterogeneous environments.
f) A-B $\wedge$ B-A counts the number of test cases per test group for which full interoperability could be detected for both homogeneous environments.
g) (A-A $\wedge$ B-B) $\wedge$ $\neg$(A-B $\wedge$ B-A) counts the number of test cases per test group for which full interoperability could be detected for both homogeneous environments, but where an interoperability problem was detected for at least one of the heterogeneous environments.

In turn, the row headers simple distinguish the different test groups. The overall figures reveal that platform A implements only 30.2% (51 test cases) and platform B only 58.6% (99 test cases) of the WS-Sec-Pol/WS-RM-Pol functionality. Based on this data, hypothesis **H1** must be rejected.

Luckily, the functionalities implemented by the two platforms largely overlap which can be inferred by observing that (A-A ∧ B-B) is close to Min(A-A,B-B) for all test groups. Therefore, there are at least 41 test cases that could be performed successfully on both homogeneous platforms which allows for testing hypothesis **H2**. Out of those 41 test cases there were 13 (31.7%) test cases that could not be performed successfully in both heterogeneous environments. Considering the complexity of WS-RM-Pol and especially WS-Sec-Pol this number does not seem to be too high, but for practical purposes an error rate of about one third is not acceptable and therefore **H2** must be rejected as well.

**Table 1: Interoperability Results per Test Group**

| Test group | # | A-A | B-B | A-A ∧ B-B | A-A ∨ B-B | A-B ∨ B-A | A-B ∧ B-A | (A-A ∧ B-B) ∧ ¬(A-B ∧ B-A) |
|---|---|---|---|---|---|---|---|---|
| Basic | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| WS-RM | 9 | 2 | 8 | 2 | 8 | 2 | 0 | 2 |
| Protection Assertions | 19 | 8 | 4 | 4 | 8 | 2 | 1 | 3 |
| Token Assertions | 65 | 10 | 37 | 10 | 37 | 8 | 5 | 5 |
| Binding Assertions | 46 | 22 | 38 | 18 | 42 | 27 | 15 | 3 |
| Supporting Token Assertions | 8 | 0 | 4 | 0 | 4 | 0 | 0 | 0 |
| WS-Sec and WS-Trust Options | 20 | 8 | 7 | 6 | 9 | 7 | 6 | 0 |
| Overall | 169 | 51 | 99 | 41 | 109 | 47 | 28 | 13 |

Taking the effects of low coverage and bad interoperability together results in very low WS-Sec-Pol/WS-RM-Pol functionality that is supported in an interoperable manner. Only 47 (27.8%) out of 169 test cases can be performed successfully in at least one of the heterogeneous environments and only 28 (16.6%) test cases for both heterogeneous environments. This means that implementing security and reliability for Web Services based on WS-RM-Pol and WS-Sec-Pol for the heterogeneous platform configurations investigated here is at least a challenge. In particular, it is not possible to exchange a SOAP message between the two platforms that is both, confidentiality and integrity protected. Platform B does not support XPath for identifying the elements to be encrypted. Instead, it relies on using the `EncryptedParts` assertion and assumes an `InlcudeToken` value of `Never` for using X509 tokens (which is the only basic token type supported for the heterogeneous environments). Platform A ignores any `IncludeToken` value an always inserts the token into the SOAP messages which is then rejected by platform B. Additionally, platform A does not support the `TransportBinding` assertion so that SSL encryption cannot be asserted either. In consequence, deriving combined test cases from isolated test cases (cf. section *TEST APPROACH*) essentially is senseless.

At least, there is an integrity and confidentiality protected interaction that comes close to WS-Sec-Pol/WS-RM-Pol based QoS implementation. For confidentiality protection, SSL is used which is configured for platform B using a standard `TransportBinding` assertion and for platform A using proprietary configuration. For integrity protection, an `AsymmetricBinding` together with an `X509Token` is specified and the elements to be signed are identified using the `SignedParts` assertion. However, apart from not being fully standards based, only using platform A as server and platform B as client can be performed successfully because the other way round a platform A client tries to retrieve proprietary configuration information in vain.

## WS-I TO THE RESCUE?

The two main deliverables of the WS-I that cover the application of WS-Sec and WS-RM are the BSP and the RSP (cf. section *INTRODUCTION*). Those profiles are complemented by test tools

and sample applications, but the profiles are authoritative. In the standard document itself, the purpose of the BSP is described as follows:

*"This document defines the WS-I Basic Security Profile 1.1, based on a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability."* (WS-I 2010a)

Those clarifications and amendments become manifest in so-called requirements together with some explaining text and, in case of the RSP, test expressions for evaluating SOAP messages. In (Barbir, Hobbs et al. 2007), one of the BSP editors explains that by using the requirements *"the BSP limits the set of common functionality that vendors must implement and thus enhances the chances for interoperability. This in return reduces the complexities for the testing of Web Services security."* Moreover, she explains that *"the security consideration statements provide guidance that is not strictly interoperability related but are testable best practices for security."*

For example, requirement *R2001* of the BSP says that *"A SENDER MUST NOT use SSL 2.0 as the underlying protocol for HTTP/S."*. The explanatory text justifies the requirement by pointing out that *"SSL 2.0 has known security issues and all current implementations of HTTP/S support more recent protocols"*.

A common characteristic of those requirements is that they define constraints on the level of SOAP messages, i.e., the existence, order and content of XML elements within SOAP messages or the actual exchange of SOAP messages is described. Consequently, the WS-I testing tools take SOAP messages as input and check them for compliance to the BSP and RSP requirements. From the perspective of facilitating interoperability, this amounts to replacing actual interoperability testing as described in section *TEST APPROACH* by checking standard compliance of SOAP messages. However, checking standard compliance itself is subject to errors and therefore merely an add-on to true interoperability testing but not a replacement. Even worse, the relation between WS-Sec-Pol/WS-RM-Pol assertions and the corresponding SOAP messages exchanged is not described in BSP and RSP at all. In section 5.1.1, the BSP explicitly allows for out of band agreement for specifying the use of WS-Sec. Moreover, it states in several sections (9, 10, 13.1) that *"[..]no security policy description language or negotiation mechanism is in scope for the Basic Security Profile[..]"*. The RSP recommends (though not requires) the use of WS-RM-Pol for configuring the use of WS-RM in its section 2.4, but it does not define the relation between WS-RM-Pol assertions and SOAP messages either.

In so far, the interoperability issues 2, 3, 4, 6, 7, 8 and 10 of section *Core Interoperability Issues* are not covered by the BSP/RSP at all. For issue 1 (no WS-RM-Pol support), platform A can be considered to ignore a WS-I recommendation. But as the RSP does not explicitly require the use of WS-RM-Pol, platform A nonetheless cannot be said to violate the RSP. For issue 5 (no `EncryptBeforeSigning` support), the BSP explicitly states in its section 6.1 ('Processing Order'), that both, encryption before signing as well as signing before encryption, may be appropriate depending on the application scenario. In so far, both protection orders must be supported by a WS-I compliant stack. However, that actually has got nothing to do with supporting the WS-Sec-Pol `EncryptBeforeSigning` assertion. As the BSP explicitly allows for out of band agreement for specifying the use of WS-Sec, not supporting `EncryptBeforeSigning` can be considered to be WS-I compliant. Finally, for issue 9 (deviating Processing of `SignedParts/Body`), the BSP states in its section 19.4 that *"it is RECOMMENDED that applications signing any part of the SOAP body sign the entire body."*

However, the WS-Sec-Pol specification is absolutely clear about this point itself (cf. OASIS 2009b, section 4.1.1).

All in all, none of the core interoperability problems detected is due to violating WS-I profiles. Only 1 issue out of 10 is reinforced by the BSP. This, taken together with the test results of the previous section, seems to imply that the approach of replacing true interoperability testing by WS-I compliance checking and neglecting the relation between WS-Sec-Pol/WS-RM-Pol assertions and SOAP messages is not sufficient for ensuring interoperability between WS stacks. Note that WS-I does not question the use of WS-Policy standards. The RSP recommends using WS-RM-Pol, the BSP states that *"strict policy specification and enforcement regarding which message parts are to be signed"* (WS-I 2010a, section 19.4) is a countermeasure against attacks, and the so-called *delivery package* of the RSP ships with a few WS-Policy definitions (though without defining the effect on SOAP messages in detail). However, it leaves out a detailed treatment of WS-RM-Pol and WS-Sec-Pol.

# FUNCTIONALITY ASSESSMENT

The content presented so far demonstrates that interoperable implementation of WS-* functionality according to definition 1 cannot be expected and that current WS stack implementations do not cover considerable parts of WS-RM-Pol and WS-Sec-Pol functionality. This information does not reveal in how far real-world scenarios of WS-* functionality can be implemented using today's stacks. Even if interoperable WS-* functionality is not available there are use cases of WS-RM and WS-Sec that are desirable for homogeneous environments as well. Content-level encryption and signing of XML messages allows for multi-party scenarios that barely can be matched using transport level security methods. The combination with additional WS-* standards such as WS-Trust (OASIS 2009d) and WS-SecureConversation (OASIS 2009e) allows integration scenarios that obviate the need of mutual security certificate exchanges outside the actual Web Services interaction. In order to assess the accessibility of advanced WS-* functionality, we review the implementability of the so-called *Secure WS-ReliableMessaging Scenario* (SecRM scenario) as a benchmark for the two WS stacks under test of this work. This scenario resulted from the interaction of major Web Services vendors who conceived it as real world use case for Web Services interaction (Gavrylyuk, Hrebicek et al. 2005). Additionally, this scenario has formally been validated by two independent research groups (Backes, Mödersheim et al. 2006; Bhargavan, Corin et al. 2007) and the analyzed properties such as mutual authentication have been identified as essential for Web Services-based Business-to-Business integration (B2Bi) (Schönberger, Wirtz et al. 2010).

The purpose of the SecRM scenario is the reliable exchange of confidential, authenticated and integrity-protected messages without frequent key exchanges (cf. Backes, Mödersheim et al. 2006; Bhargavan, Corin et al. 2007, for more detailed descriptions of security properties). In order to achieve these goals WS-Sec, WS-RM, WS-Trust, WS-SecureConversation as well as WS-Addressing are combined for establishing a WS-SecureConversation *session* (or *security context*) which is then used to reliably and securely exchange messages. Basically, the SecRM scenario consists of a key-exchange phase, a message sending phase and a termination phase (Backes, Mödersheim et al. 2006). The key-exchange phase generates a so-called Security Context Token (SCT) and uses asymmetric keys for integrity as well as confidentiality protection of WS-SecureConversation bootstrap messages. After the generation of the security context, a WS-RM sequence is initiated which is used for exchanging payload messages. Once the exchange of all

payload-messages has been acknowledged, the WS-RM sequence as well as the security context are subsequently terminated.

The steps listed below describe the interactions between client and sender role of the SecRM scenario and reflect the definitions of (Backes, Mödersheim et al. 2006) and (Gavrylyuk, Hrebicek et al. 2005, pages 34-60). The prefixes `wsrm`, `wss`, `wst`, `wsu` and `env` are used to identify concepts of the WS-RM, WS-Sec, WS-Trust, WS-Sec utility and SOAP standards, respectively.

1) *RequestSecurityToken RST*
   The client sends a message containing a RST to the service, asking the service to issue a SCT. This message must be signed and encrypted by the client using the service's public key.
2) *RequestSecurityTokenResponse RSTR*
   The service responds with a RSTR message, containing the requested SCT. This message must be signed and encrypted by the service using the client's public key. The SCT must be used for signing and encrypting any message of the subsequent message flow.
3) *CreateSequence*
   The client sends a `wsrm:CreateSequence` message to the service. This message includes a `wss:SecurityContextReference` to reference the SCT received in step 2. This SCT is used to sign the `wsrm:CreateSequence` message and encrypt the `wss:Signature`.
4) *CreateSequenceResponse*
   The service responds to the CreateSequence request with a `wsrm:CreateSequenceResponse` message. This message is also signed and the `wss:Signature` is encrypted using the SCT.
5) *Payload Message*
   The client now sends signed and encrypted messages containing the payload of this communication. Each payload message contains a WS-ReliableMessaging sequence header containing at least the sequence identifier and the sequence number of the according message. In contrast to the previous two and following five messages, the payload messages have an encrypted `env:Body`.
6) *SequenceAcknowledgement*
   The service acknowledges the receipt of the payload message(s) with a `wsrm:SequenceAcknowledgement`. The scenario definition proposes a single `wsrm:SequenceAcknowledgement` message with an empty `env:Body`. The acknowledgment headers are also signed and the `wss:Signature` is encrypted.
7) *TerminateSequence*
   As soon as the client has received the acknowledgments for each message within the message sequence, it closes this sequence using the `wsrm:TerminateSequence` message defined by WS-RM. This message is signed and the `wss:Signature` is encrypted, too.
8) *TerminateSequenceResponse*
   The service confirms the termination of the sequence with a signed `wsrm:TerminateSequenceResponse` message. The `wss:Signature` of this message is encrypted.
9) *CancelSecurityToken*
   After termination of the WS-ReliableMessaging sequence, the client asks the service for cancellation of the WS-SecureConversation context using a `wst:CancelTarget`

message. The WS-Addressing header elements, the `wss:Signature`, and the `wsu:Timestamp` are integrity protected. Additionally, the signature is encrypted whereas the message `env:Body` is not protected.

10) *CancelSecurityTokenResponse*
The service confirms the `wst:CancelTarget` with a `wst:RequestedTokenCancelled` message. This message is protected in the same way as the `wst:CancelTarget` message.

Once the security context is started, sender and receiver may create multiple WS-RM sessions for message transmission as the publications do not impose any restrictions on that. However, in order to fulfill the requirements of the SecRM scenario each WS-RM session that is started within the WS-SecureConversation security context must be closed or terminated within the same context. After the security context is established, all signature and encryption processes are performed using keys derived from the SCT. The last two messages, canceling the security context, are protected using SCT-derived keys, too.

Note that (Gavrylyuk, Hrebicek et al. 2005) and (Backes, Mödersheim et al. 2006) make use of WS-ReliableMessaging version 1.0 (BEA, IBM et al. 2005) which does not define `wsrm:CloseSequence`, `wsrm:CloseSequenceResponse`, or `wsrm:TerminateSequenceResponse` messages. However, these messages have been defined for WS-RM since version 1.1 (OASIS 2008). Therefore, message number eight `wsrm:TerminateSequenceResponse` is inserted into the message sequence as recommended by WS-I's RSP (WS-I 2010b, pages 28/29).

Table 2 lists the requirements of the scenario definitions for confidentiality and integrity protection. The corresponding message type number is put in the first column. The other columns show the key required for encrypting (enc) or signing (sig) the message elements `wss:Signature` (Sign), `env:Body`, `wsu:Timestamp` (TS), WS-Addressing headers (WS-A), WS-ReliableMessaging headers (WS-R), and `wss:EncryptedKey` (EncKey). The entry $SK_X$ stands for a session key, which is usually encrypted using the receivers public key ($PuK_Y$ ). The private key of a party is abbreviated $PrK_Y$. If a `wsc:SecurityContextToken` is used to derive keys, these keys are labeled as $DK_{YX}$. The indices X and Y are variables that are substituted in the table. Instead of the X a number is inserted to identify different instances of the corresponding key type. These instances are independent of the type of party. The index Y determines whether the client (C) or the service (S) is the owner of the key, e.g., $PrK_S$ denotes that the key used for the specified operation is the private key of the service, while $DK_{C1}$ stands for a key derived from a `SecurityContextToken` by the client. Since multiple derived keys may be used in a SOAP message, each derived key has an assigned number, here '1'. The '•' symbol indicates that the corresponding element is present, but not protected, whereas the '○' means that the corresponding element is not present in this message.

The defining sources of the SecRM scenario describe the messages to be exchanged in detail. However, no WS-Policy definitions are given to instruct the WS stacks under test to create the message as intended. Therefore we sketch the policy configuration of the SecRM scenario in section *Policy Configuration* that has been derived from the scenario descriptions and the sample messages provided in (Gavrylyuk, Hrebicek et al. 2005). Section *SecRM Scenario Test Results* then checks the generated message exchanges by platform A and B following the approach of section *TEST APPROACH*.

**Table 2: Message Protection Requirements and Keys Required for Protection Realization as Defined by the SecRM Scenario**

| Msg No | Sign | Body | | TS | WS-A | WS-R | EncKey |
|---|---|---|---|---|---|---|---|
| | enc | enc | sig | sig | sig | sig | enc |
| 1 | $SK_1$ | | $PrK_C$ | | | ○ | $PuK_S$ |
| 2 | $SK_2$ | | $PrK_S$ | | | ○ | $PuK_C$ |
| 3 | $DK_{C1}$ | • | $DK_{C2}$ | | | ○ | ○ |
| 4 | $DK_{S1}$ | • | $DK_{S2}$ | | | ○ | ○ |
| 5 | $DK_{C1}$ | | $DK_{C2}$ | | | | ○ |
| 6 | $DK_{S1}$ | • | • | $DK_{S2}$ | | | ○ |
| 7 | $DK_{C1}$ | • | $DK_{C2}$ | | | | ○ |
| 8 | $DK_{S1}$ | • | $DK_{S2}$ | | | | ○ |
| 9 | $DK_{C1}$ | • | $DK_{C2}$ | | | ○ | ○ |
| 10 | $DK_{S1}$ | • | $DK_{S2}$ | | | ○ | ○ |

## Policy Configuration

The policy configurations below sketch the platformindependent specification of the service's WSDL definition for the SecRM scenario. In order to deploy these configurations on platform A and B as referred to above some specific assertions have to be defined. For example, XPath-based `Encrypted`- and `SignedElements` assertions have to be added to work around platform A's non-compliant handling of `EncryptSignature`, `EncryptBeforeSigning` or `Timestamp` assertions (cf. above). For such details, please see (Schwalb and Schönberger 2010).

The presentation of the WS-Policy definition is split up into four listings where listing 2 shows the security binding and WS-RM configuration for the actual payload messages, listing 3 shows the configuration for setting up the security context, and listings 4 and 5 show the definition of WS-Sec protection assertions for incoming and outgoing SOAP messages, respectively.

The policy for the actual message exchange (listing 2) defines the use of WS-RM (lines 5-7) and WS-Addressing (lines 9-11) as well as the binding for the security context (lines 13-43). The WS-RM assertion activates the use of WS-RM within the security context. Neither (Gavrylyuk, Hrebicek et al. 2005) nor (Backes, Mödersheim et al. 2006) allow to draw a conclusion about the delivery assurance to be used in the scenario or whether the WS-RM sequence should be bound to a security token using the `wsrmp:SequenceSTR` assertion. Conversely, the use of WS-Addressing is required by both publications. The WS-Addressing assertion enables the use of WS-Addressing message header properties such as `wsa:To`, `wsa:Action`, `wsa:MessageID`, or `wsa:RelatesTo` for implementing functionality as required by the SecRM scenario definition.

The setup of the security context is denoted in the `sp:SymmetricBinding` assertion, since the session key is symmetric. A `sp:SecureConversationToken` is established as protection token. This token is the base for signature and encryption key derivation (`sp:RequireDerivedKeys` assertion). The `sp:BootstrapPolicy` is the policy used to obtain the `sp:SecureConversationToken` from the token issuer (see listing 3 for a specification of this policy). Within the security context the `sp:Basic128` algorithm is used for encryption, since (Gavrylyuk, Hrebicek et al. 2005) and (Backes, Mödersheim et al. 2006)

propose 128-bit cryptography. Considering the sample message structures in (Gavrylyuk, Hrebicek et al. 2005), the `sp:Layout` of the SOAP messages is a `sp:Strict` layout (see OASIS 2009b, pages 52/53) and a `wsu:Timestamp` must be included. Both aspects are not explicitly specified in (Backes, Mödersheim et al. 2006). A significant difference between the two scenario specifications is the protection order. Whereas (Gavrylyuk, Hrebicek et al. 2005) state that "signature occurs before encryption" (Gavrylyuk, Hrebicek et al. 2005, page 35) (first sign the body and then encrypt body and signature), (Backes, Mödersheim et al. 2006) propose to encrypt the message body first, then to sign the corresponding message parts including the `env:Body`, and finally to encrypt the signature. Since (Backes, Mödersheim et al. 2006) give a formal cryptographic analysis of this scenario, the `sp:EncryptBeforeSigning` assertion has been chosen. The `sp:EncryptSignature` assertion then requires the encryption of the `wss:Signature`.

**Listing 2. The WS-Policy for the SecRM Scenario**

```
<wsp:Policy wsu:Id ="SecureRMSessionBinding">
    <wsp:ExactlyOne>
        <wsp:All>

            <wsrmp:RMAssertion>
                <wsp:Policy/>
            </wsrmp:RMAssertion>

            <wsam:Addressing>
                <wsp:Policy/>
            </wsam:Addressing>

            <sp:SymmetricBinding>
                <wsp:Policy>
                    <sp:ProtectionToken>
                        <wsp:Policy>
                            <sp:SecureConversationToken>
                                <wsp:Policy>
                                    <sp:RequireDerivedKeys/>
                                    <sp:BootstrapPolicy>
                                        <! --
                                            See the XML listing containing the BootstrapPolicy
                                        -->
                                    </sp:BootstrapPolicy>
                                </wsp:Policy>
                            </sp:SecureConversationToken>
                        </wsp:Policy>
                    </sp:ProtectionToken>
                    <sp:AlgorithmSuite>
                        <wsp:Policy>
                            <sp:Basic128/>
                        </wsp:Policy>
                    </sp:AlgorithmSuite>
                    <sp:Layout>
                        <wsp:Policy>
                            <sp:Strict/>
```

```
37                      </wsp:Policy>
38                  </sp:Layout>
39                  <sp:IncludeTimestamp/>
40                  <sp:EncryptBeforeSigning/>
41                  <sp:EncryptSignature/>
42              </wsp:Policy>
43          </sp:SymmetricBinding>
44
45      </wsp:All>
46  </wsp:ExactlyOne>
47 </wsp:Policy>
```

Listing 3 gives the policy for the `sp:BootstrapPolicy` (OASIS 2009b, p. 41) of listing 2. The `sp:BootstrapPolicy` defines the policy for the `sp:SecureConversationToken` request and the `sp:SecureConversationToken` issuance. (Gavrylyuk, Hrebicek et al. 2005) and (Backes, Mödersheim et al. 2006) stipulate the use of X509 certificates in the `sp:BootstrapPolicy`, which are certified by a certificate authority. This is important for real-world use cases, however, for the purpose of testing platforms A and B, client and service are in possession of a trusted X509 certificate of each other. This change of the scenario does not have any effect on the policy configuration or the SOAP message traffic between client and service, and is therefore not relevant in this work.

The `sp:BootstrapPolicy` in listing 3 uses an `sp:AsymmetricBinding` (public-key cryptography) with a `sp:X509Token` for the initiator and the recipient. The initiator should send his public key as `wss:BinarySecurityToken` to the recipient, whereas the key of the recipient must not necessarily be transmitted to the client. The `sp:AlgorithmSuite`, the `sp:Layout`, the `wsu:Timestamp` inclusion, the protection order, and the `sp:EncryptSignature` instruction are used analogously to the `sp:SymmetricBinding` of the main policy.

**Listing 3. The sp:BootstrapPolicy of the sp:SecureConversationToken in the SecRM Scenario**

```
1 <wsp:Policy>
2     <sp:AsymmetricBinding>
3         <sp:Policy>
4             <sp:InitiatorToken>
5                 <wsp:Policy>
6                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                          securitypolicy/200702/IncludeToken/AlwaysToRecipient">
7                         <wsp:Policy>
8                             <sp:WssX509V3Token10/>
9                         </wsp:Policy>
10                    </sp:X509Token>
11                </wsp:Policy>
12            </sp:InitiatorToken>
13            <sp:RecipientToken>
14                <wsp:Policy>
15                    <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                          securitypolicy/200702/IncludeToken/Never">
16                        <wsp:Policy>
```

```
17                          <sp:WssX509V3Token10/>
18                      </wsp:Policy>
19                  </sp:X509Token>
20              </wsp:Policy>
21          </sp:RecipientToken>
22          <sp:AlgorithmSuite>
23              <wsp:Policy>
24                  <sp:Basic128/>
25              </wsp:Policy>
26          </sp:AlgorithmSuite>
27          <sp:Layout>
28              <wsp:Policy>
29                  <sp:Strict/>
30              </wsp:Policy>
31          </sp:Layout>
32          <sp:IncludeTimestamp/>
33          <sp:EncryptBeforeSigning/>
34          <sp:EncryptSignature/>
35      </wsp:Policy>
36  </sp:AsymmetricBinding>
37
38  <sp:SignedParts>
39      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
40      <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
41      <sp:Body/>
42  </sp:SignedParts>
43
44  <sp:EncryptedParts>
45      <sp:Body/>
46  </sp:EncryptedParts>
47 </wsp:Policy>
```

In addition to the security binding assertions, the `sp:BootstrapPolicy` specifies the message elements to be protected. The scenario definition stipulates that the WS-Addressing headers, the `wsu:Timestamp`, and the message `env:Body` are to be signed. The assertions in lines 39 and 40 of listing 3 indicate that all WS-Addressing headers must be signed (for reasons of compatibility the XML namespace of two WS-Addressing versions is specified), the assertion in line 41 defines that the `env:Body` must be signed. The `wsu:Timestamp` of a SOAP message must always be covered by a signature due to the `sp:IncludeTimestamp` definition in (OASIS 2009b, page 51). The encryption of the `env:Body` is declared in lines 44-46 and the encryption of the signature is asserted with the `sp:EncryptSignature` element in the binding (line 34 of listing 3).

The protection assertions of the messages from the client to the service (see listing 4) and vice versa (see listing 5) are similar to the assertions defined in the `sp:BootstrapPolicy`: the WS-Addressing headers and the `env:Body` are intended for integrity protection, and the `env:Body` is also encrypted. In addition, the WS-RM header sections are signed.

**Listing 4. The WS-SecurityPolicy protection assertions for the input messages**

```
1 <wsp:Policy wsu:Id="SecureRMSessionInput">
```

```
2       <sp:SignedParts>
3           <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
4           <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
5           <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
6           <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
7           <sp:Body/>
8       </sp:SignedParts>
9       <sp:EncryptedParts>
10          <sp:Body/>
11      </sp:EncryptedParts>
12 </wsp:Policy>
```

**Listing 5. The WS-SecurityPolicy protection assertions for the output messages**

```
1 <wsp:Policy wsu:Id="SecureRMSessionOutput">
2       <sp:SignedParts>
3           <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
4           <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
5           <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
6           <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
7           <sp:Body/>
8       </sp:SignedParts>
9       <sp:EncryptedParts>
10          <sp:Body/>
11      </sp:EncryptedParts>
12 </wsp:Policy>
```

### SecRM Scenario Test Results

This section analyzes the SOAP message traffic produced by platforms A and B for the SecRM scenario by checking for the existence of relevant WS-* elements as defined in Table 2 in the exchanged SOAP messages. The approach taken follows the test approach outlined in section *TEST APPROACH* which is, roughly speaking, invocation of the service by a QoS-aware client, logging of the SOAP message traffic, and analysis of the captured message trace.

The analysis of the test run on platform B shows a close match to the scenario definitions. Yet, the results obtained slightly deviate from the expected results, since, in general, the `wss:Signature` is not encrypted if the `env:Body` is not encrypted on platform B. This concerns message types 3, 4, 7, 8, and 9. Message type 6 contains an encrypted and signed `env:Body`, although the scenario definitions intend an empty and unprotected `env:Body`. Although this is a deviation from the scenario definition, the behavior of platform B is standard compliant since the *"primary signature element is NOT REQUIRED to be encrypted [...] when there is nothing in the message that is covered by this signature that is encrypted."* (OASIS 2009b, lines 1730/1731). Conversely, the `env:Body` of message type 10 is encrypted, and therefore the `wss:Signature` also is encrypted.

Interestingly, a platform B client first sends a `wsrm:CloseSequence` message and then a `wsrm:TerminateSequence` message to terminate the sequence. The service answers both messages with a corresponding response message. Although this means that platform B's message sequence differs from the scenario description, this is a minor deviation since both

message types are protected in exactly the same way and both have the same purpose. Therefore, the four messages are treated as two two-part message types 7 and 8 in the scenario analysis below.

Table 3 summarizes the results of the scenario analysis on platform B. It shows the protected message parts and the keys used for protection. The notation conventions correspond to the ones defined for Table 2.

Compared to platform B, the results obtained from the test runs on platform A deviate significantly from the scenario definitions. Concerning the message sequence, the fact that neither the WS-RM sequence nor the WS-SecureConversation security context is terminated or canceled is striking. On platform A, sequences or sessions have to be closed explicitly by addressing the WS-RM sequence in the source code of a client application. However, this does not fulfill the call for policy-based implementation of WS-* functionality (cf. section INTRODUCTION) and is therefore considered to be invalid.

However, not only the message sequence does not comply with the SecRM scenario, but also the structure of several messages. The `wsu:Timestamp` elements have no expiration date, the message `env:Body` always is encrypted, even when the scenario does not intend a confidentiality protection, and platform A only uses a `wss:SecurityTokenReference` instead of an embedded `wsc:SecurityContextToken`. Similar to the results for platform B, message type 6 has an integrity and confidentiality protected `env:Body` which is not scenario compliant. Table 4 lists the protected parts and gives an overview of the keys used to realize the protection. The notation corresponds to the one of Table 2 and Table 3.

**Table 3: Message Protection and Keys Used for Protection Realization by Platform B**

| Msg No | Sign | Body | | TS | WS-A | WS-R | EncKey |
|---|---|---|---|---|---|---|---|
| | enc | enc | sig | sig | sig | sig | enc |
| 1 | $SK_1$ | | | $PrK_C$ | | ○ | $PuK_S$ |
| 2 | $SK_2$ | | | $PrK_S$ | | ○ | $PuK_C$ |
| 3 | • | • | | $DK_{C2}$ | | ○ | ○ |
| 4 | • | • | | $DK_{S2}$ | | ○ | ○ |
| 5 | $DK_{C1}$ | | | $DK_{C2}$ | | | ○ |
| 6 | $DK_{S1}$ | | | $DK_{S2}$ | | | ○ |
| 7 | • | • | | $DK_{C2}$ | | | ○ |
| 8 | • | • | | $DK_{S2}$ | | | ○ |
| 9 | • | • | | $DK_{C2}$ | | ○ | ○ |
| 10 | $DK_{S1}$ | | | $DK_{S2}$ | | ○ | ○ |

**Table 4: Message Protection and Keys Used for Protection Realization on Platform A**

| Msg No | Sign | Body | | TS | WS-A | WS-R | EncKey |
|---|---|---|---|---|---|---|---|
| | enc | enc | sig | sig | sig | sig | enc |
| 1 | $SK_1$ | | | $PrK_C$ | | ○ | $PuK_S$ |
| 2 | $SK_2$ | | | $PrK_S$ | | ○ | $PuK_C$ |
| 3 | $DK_{C1}$ | | | $DK_{C2}$ | | ○ | ○ |
| 4 | $DK_{S1}$ | | | $DK_{S2}$ | | ○ | ○ |
| 5 | $DK_{C1}$ | | | $DK_{C2}$ | | | ○ |
| 6 | $DK_{S1}$ | | | $DK_{S2}$ | | | ○ |

# RELATED WORK

In SOA research, considerable efforts have been undertaken towards testing QoS, for example (Canfora and Di Penta 2006-2008; Bertolino, Angelis et al. 2007), in the traditional sense of measurable network qualities like throughput or latency. (Shopov and Kakanakov 2007) provide an evaluation of the WS-Sec implementation of the Axis2 WS stack, but only consider the processing time and message size when using different WS-Sec features. In contrast, we focus on testing the interoperability of implementations of QoS features as provided by WS-* standards and security and reliability in particular.

(Barbir, Hobbs et al. 2007) discuss challenges of testing Web Services and security in SOA environments. Interoperability is identified as a core requirement for testing service compositions, but an interoperability assessment of different WS-* implementations is not provided. Several approaches such as (Shetty and Vadivel 2009; Bertolino and Polini 2005) target at testing interoperability of Web Services, but do not consider WS-* based QoS features.

In the area of actually testing interoperability of WS-* based QoS features, the majority of approaches follows (Tsai, Zhou et al. 2008) in defining interoperability testing as *"Conformance testing to ensure compliance with SOA protocols and standards"*. (Bhargavan, Fournet et al. 2005) describe a tool for statically validating WS-Sec-Pol configurations and (Nakamura, Sato et al. 2007) demonstrate how to use predicate logic to statically validate security policies. (Gruschka, Luttenberger et al. 2006) present an approach for checking SOAP messages for WS-Sec-Pol conformance at run-time. (Prennschütz-Schützenau, Mukhi et al. 2009) check conformance to the BSP statically and dynamically by inspecting SOAP messages. All these approaches analyze the configurations and message traffic of a single WS stack instance whereas we focus on the interaction of two WS stack instances of different solution providers. While we investigate whether or not communication can be completed successfully these approaches are able to check conformance to the WS-* specifications. In so far, these kind of approaches and our approach can be considered to complement each other.

(Simon, László et al. 2010) present an interoperability assessment of SOA products with respect to WS-* standards for the Hungarian e-Government infrastructure. They evaluate only two security related test cases, but six SOA products and conclude that some products (including Metro/GlassFish) are mature for WS-* interoperability. The problems identified in the work at hand, however, reveal that much more thorough testing is needed.

(Menzel, Warschofsky et al. 2010b) provide a mature framework for testing, monitoring and analyzing Web Services that are secured via WS-Sec-Pol. While interoperability assessment

across different stack vendors is out of scope, it may be useful for streamlining the interoperability assessment of more WS stacks as described in this work. Moreover, the pattern-based approach for deriving WS-Sec-Pol configurations described in (Menzel, Warschofsky et al. 2010a) may be used for deriving test cases that go beyond this work's test cases in providing application level features such as mutual authentication.

The Secure WS-ReliableMessaging Scenario has been analyzed by two different research groups (Backes, Mödersheim et al. 2006; Bhargavan, Corin et al. 2007). However, the focus of these projects was on formal analysis of security and reliability properties and not on policy-based realization. The original definition of the SecRM scenario stems from industry interoperability workshops and (Gavrylyuk, Hrebicek et al. 2005) provide a good description of the messages that are to be exchanged. However, the WS-Policy configurations for implementing the scenario are not given. One of the original authors of the SecRM scenario was asked for an updated version of the SecRM scenario that makes use of the latest WS-RM, WS-RM-Pol and WS-SecPol standards, but no updated version was available. Consistently, we were not able to discover an evaluation of the SecRM scenario support by WS stacks based on policy-based realization.

# CONCLUSION AND FUTURE WORK

In this paper, we have thoroughly analyzed the interoperable implementation of WS-Sec-Pol/WS-RM-Pol by two major Java-based WS stacks. Our results show that Web Services developers cannot rely on interoperability of such WS-* features as provided by arbitrary WS stacks. While WS-I's work indisputably contributed to better Web Services interoperability, it apparently is not sufficient for guaranteeing interoperability of WS-Policy-driven QoS implementation. Using WS-Sec/WS-RM without WS-Policy however places unacceptable burdens on Web Services developers in requiring them to manually manipulate SOAP messages and agreeing on security mechanisms without predefined format. Functionality-wise we could prove that advanced real-world functionality can be implemented using WS-Sec-Pol and WS-RM-Pol. However, the choice of WS stack may severely influence availability and standards compliance of the solutions. Clearly, the identification of a broader range of advanced real-world usage scenarios is necessary to give practitioners a lever for deciding upon the use of WS-* or a particular WS-stack.

Future work therefore will focus on gathering more realistic use cases that justify the use of WS-* standards. Combined with the approach for systematically testing the basic functionality of WS-* standards, a comprehensive set of WS-Policy-based test cases shall be derived that can be used among WS stack vendors to assess cross-stack interoperability on the one hand and to ensure the availability of reasonable functionality on the other hand. Additionally, automated interoperability testing procedures are needed to streamline interoperability testing between various WS stacks in different versions.

## REFERENCES

Backes, M., Mödersheim, S., Pfitzmann, B., Vigano, L. (2006). Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario, in *Proceedings of Foundations of Software Science and Computational Structures* (FOSSACS), ser. Lecture Notes in Computer Science, vol. 3921. Springer, March 2006, pp. 428–445.

Barbir, A., Hobbs, C., Bertino, E., Hirsch, F., Martino, L. (2007). Challenges of testing Web Services and security in SOA implementations, in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds. Springer Berlin Heidelberg, 2007, pp. 395–440.

BEA Systems, IBM Corporation, Microsoft Corporation Inc., TIBCO Software Inc. (2005). Web Services Reliable Messaging Protocol (WS-ReliableMessaging), February 2005, first specification (Version 1.0). [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf

Bertolino, A., Angelis, G.D., Polini, A. (2007). A QoS Test-Bed Generator for Web Services, in *Proceedings of the 7th International Conference on Web Engineering* (ICWE 2007), Como, Italy, July 2007, ser. Lecture Notes in Computer Science, L. Baresi, P. Fraternali, and G.-J. Houben, Eds., vol. 4607. Springer Verlag, Berlin/Heidelberg, Germany, July 2007, pp. 17–31.

Bertolino, A., Polini, A. (2005). The audition framework for testing Web Services interoperability, in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, ser. EUROMICRO '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 134–142.

Bhargavan, K., Corin, R., Fournet, C., Gordon, A.D. (2007). Secure sessions for web services, *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, article no. 8, 46 pages, 2007.

Bhargavan, K., Fournet, C., Gordon, A.D., O'Shea, G. (2005). An advisor for Web Services security policies, in *Proceedings of the 2005 workshop on Secure web services*, ser. SWS '05. New York, NY, USA: ACM, 2005, pp. 1–9.

Canfora, G., Di Penta, M. (2006-2008). Service-Oriented Architecture Testing: A Survey, in *Revised Tutorial Lectures of the International Summer Schools for Software Engineering* (ISSSE 2006 - 2008), Salerno, Italy, ser. Lecture Notes in Computer Science, A. D. Lucia and F. Ferrucci, Eds., vol. 5413. Springer Verlag, Berlin/Heidelberg, 2006 - 2008, pp. 78–105.

Curbera, F., Khalaf, R., Mukhi, N. (2008). Quality of service in SOA environments. An overview and research agenda (quality of service in SOA-Umgebungen), *it - Information Technology*, vol. 50, no. 2, pp. 99–107, 2008.

Gavrylyuk, K., Hrebicek, O., Batres, S. (2005). WCF (Indigo) Interoperability Lab: Reliable Messaging. Microsoft. [Online]. Available: http://mssoapinterop.org/ilab/RM/WCFInteropPlugFest_RM.doc, Retrieved 08/29/2011

Gruschka, N., Jensen, M., Dziuk, T. (2007). Event-based application of WS-security policy on SOAP messages, in *Proceedings of the 2007 ACM workshop on Secure web services*, ser. SWS '07. New York, NY, USA: ACM, 2007, pp. 1–8.

Gruschka, N., Luttenberger, N., Herkenhöner, R. (2006). Event-based SOAP message validation for WS-SecurityPolicy-enriched Web Services, in *Proceedings of the 2006 International Conference on Semantic Web & Web Services*, SWWS 2006, Las Vegas, Nevada, USA, June 26-29, 2006. CSREA Press, 2006, pp. 80–86.

Khalaf, R., Keller, A., Leymann, F. (2006). Business processes for Web Services: principles and applications, *IBM Syst. J.*, vol. 45, pp. 425–446, January 2006.

Martino, L., Bertino, E. (2009). Security for Web Services: Standards and research issues, *Int. J. Web Services Res.*, vol. 6, no. 4, pp. 48–74, 2009.

Menzel, M., Warschofsky, R., Meinel, C. (2010a). A pattern-driven generation of security policies for service-oriented architectures, in *Proceedings of the 2010 IEEE International Conference on Web Services*, Miami, Florida, USA, ser. ICWS '10. IEEE Computer Society, 2010, pp. 243–250.

Menzel, M., Warschofsky, R., Thomas, I., Willems, C., Meinel, C. (2010b). The service security lab: A model-driven platform to compose and explore service security in the cloud, in *Proceedings of the 2010 6th World Congress on Services*, ser. SERVICES '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 115–122.

Moser, L. E., Melliar-Smith, P.M., Zhao, W. (2007). Building dependable and secure Web Services, *Journal of Software*, vol. 2, no. 1, pp. 14–26, 2007.

Nakamura, Y., Sato, F., Chung, H.V. (2007). Syntactic validation of Web Services security policies, in *Proceedings of the 5th international conference on Service-Oriented Computing*, Vienna, Austria, ser. ICSOC '07, Springer- Verlag : Berlin, Heidelberg, 2007, pp. 319–329.

Nezhad, H., Benatallah, B., Casati, F., Toumani, F. (2006). Web Services interoperability specifications, *Computer*, vol. 39, no. 5, pp. 24–32, May 2006.

OASIS (2004). Web Services Security UsernameToken Profile 1.0, OASIS, March 2004. [Online]. Available: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

OASIS (2006). Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS, February 2006. [Online]. Available: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

OASIS (2008). Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1, OASIS, January 2008. [Online]. http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01-e1.pdf

OASIS (2009a). Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.pdf

OASIS (2009b). WS-SecurityPolicy 1.3, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf

OASIS (2009c). Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.2, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.2-spec-os.pdf

OASIS (2009d). WS-Trust 1.4, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf

OASIS (2009e). WS-SecureConversation 1.4, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf

Prennschütz-Schützenau, S., Mukhi, N.K., Hada, S., Sato, N., Satoh, F., Uramoto, N. (2009). Static vs. dynamic validation of BSP conformance, in *Proceedings of the 2009 IEEE International Conference on Web Services*, ser. ICWS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 919–927.

Schönberger, A., Schwalb, J., Wirtz, G. (2011). Has WS-I's work resulted in WS-* interoperability? in *Proceedings of the 9th 2011 International Conference on Web Services* (ICWS 2011), Washington, D.C., USA. IEEE, July 2011, pp. 171-178

Schönberger, A., Wirtz, G., Huemer, C., Zapletal, M. (2010). *A composable, QoS-aware and Web Services-based execution model for ebXML BPSS BusinessTransactions*, in Proceedings of the 6th 2010 World Congress on Services (SERVICES2010), Fourth International Workshop on Web Services and Cloud Services Testing (WS-CS-Testing 2010), Miami, Florida, USA. IEEE, July 2010, pp. 229-236

Schwalb, J., Schönberger, A., Wirtz, G. (2010). Approaching interoperability testing of QoS based on WS-* standards implementations, in *4th Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing* (NFPSLAM-SOC'10), co-located with 8th IEEE European Conference on Web Services (ECOWS 2010), Ayia Napa, Cyprus. IEEE, December 2010.

Schwalb, J., Schönberger, A. (2010). Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations, Technical Report, Otto-Friedrich-Universität Bamberg, *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik 87*, September 2010.

Shetty, S., Vadivel, S. (2009). Interoperability issues seen in Web Services, *International Journal of Computer Science and Network Security* (IJCSNS), vol. 9, no. 8, Seoul, Republic of Korea, pp. 160–169, August 2009.

Shopov, M., Kakanakov, N. (2007). Evaluation of a single WS-Security implementation, in *Proceedings International Conference on Automatics and Informatics*, Sofia, Bulgaria, October 2007, pp. 39–42.

Simon, B., László, Z., Goldschmidt, B., Kondorosi, K., Risztics, P. (2010). Evaluation of WS-* standards based interoperability of SOA products for the hungarian egovernment infrastructure, in *Proceedings of the*

*2010 Fourth International Conference on Digital Society*, ser. ICDS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 118–123.

Tsai, W.-T., Zhou, X., Chen, Y., Bai, X. (2008). On testing and evaluating service-oriented software,"*Computer*, vol. 41, pp. 40–46, August 2008.

W3C (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C, April 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

Wegner, P. (1996). Interoperability, *ACM Comput. Surv.*, vol. 28, no. 1, pp. 285–287, 1996.

WS-I (2010a). Basic Security Profile Version 1.1, WS-I, January 2010. [Online]. Available: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

WS-I (2010b). Reliable Secure Profile Version 1.0, WS-I, November 2010. [Online]. Available: http://www.ws-i.org/profiles/ReliableSecureProfile-1.0-2010-11-09.html

## ABOUT THE AUTHORS

**Andreas Schönberger** is a research assistant and PhD student at the Distributed Systems Group at the University of Bamberg, Germany. His areas of research focus on Business-to-Business integration, Service Science and Choreographies and Orchestrations in particular. Andreas has a degree in information systems from the University of Bamberg.

**Johannes Schwalb** is a software engineer at Senacor Technologies AG, located in the Nuremberg area, Germany. His research is focused on Java Enterprise Edition based SOA implementations and the interoperability of Web Services communication. In particular, the declarative realization of non-functional properties using WS-* technologies is one of his key areas.
Currently, Johannes works for Senacor in one of the largest SOA implementation projects in Germany.
Johannes has a degree in information systems from the University of Bamberg.

**Guido Wirtz** received a doctoral degree from the University of Bonn (1990) and a habilitation degree from the University of Siegen (1995). He worked 6 years as an associate professor at the University of Münster. Since 2002, Guido is a full professor for Practical CS and founder of the Distributed Systems Group at the University of Bamberg, Germany. From 2007 through 2009 he acted as dean of the Faculty of Information Systems and Applied Computer Sciences. His research interests include design methods, visual languages and tools for complex distributed systems and middleware integration in general with a focus on B2Bi and bringing business processes to work in a SOA context. Guido is a long-time member of the GI, IEEE-CS and ACM.