Semantic Matching of WS-SecurityPolicy Assertions

Monia Ben Brahim, Tarak Chaari, Maher Ben Jemaa, and Mohamed Jmaiel

ReDCAD Laboratory, University of Sfax, National School of Engineers of Sfax,
BP 1173, 3038 Sfax, Tunisia
monia.benbrahim@gmail.com, tarak.chaari@redcad.org,
{maher.benjemaa,mohamed.jmaiel}@enis.rnu.tn
http://www.redcad.org

Abstract. The lack of semantics in WS-SecurityPolicy (WS-SP) hampers the effectiveness of matching the compatibility between WS-SP assertions. To resolve this problem, we present in this paper a semantic approach for specifying and matching the security assertions. The approach consists in the transformation of WS-SP into an OWL-DL ontology and the definition of a set of semantic relations that can exist between the provider and requestor security concepts. We show how these relations lead to more correct and flexible matching of security assertions.

1 Introduction

Dynamic service discovery and selection is an essential aspect of Service Oriented Architecture (SOA). To meet modern business requirements, service selection must not only take into account functional aspects, but also non-functional properties of the service. This paper focuses on message security which is one of these non-functional properties. Message security becomes a major concern when using Web services, the most adopted implementation of SOA. Message security mainly means the confidentiality and the integrity of data transmitted through the message. Confidentiality and integrity can be assured by applying security mechanisms such as encryption and digital signature.

WS-Security [1] and WS-SecurityPolicy (WS-SP) [2] are the most important standards for definition and enforcement of message security in systems based on Web services. While WS-Security defines the syntax of security elements in a SOAP message, a WS-SP document describes which security measures are to be applied to which parts of the message. The basic building blocks of SPs are security assertions that consider the WS-Security message security model. Every single assertion may represent a specific requirement, capability, other property, or a behavior related to message security [2]. WS-SP is built on top of WS-Policy framework [3].With the help of WS-Policy operators, security assertions can be composed to complex security policies.

WS-SP is widely accepted in the industry and it is currently a popular standard to be aggregated into the Web service architecture. Nevertheless, WS-SP has a major weakness: it only allows syntactic matching of security policies. In

G. Pallis et al. (Eds.): ICSOC 2011, LNCS 7221, pp. 114–130, 2012.

fact, security policy matching depends on the policy intersection mechanism [3] provided by WS-Policy. The core step in this mechanism is matching the assertions specified in the service provider and requestor policies. This step consists in a pure syntactic comparison between the security assertions, neglecting the domain-specific semantics of assertions such as message security semantics. Although WS-Policy admits that checking the compatibility of policy assertions may involve domain-specific processing, it does not give hints on how to integrate it. Syntactic matching of security assertions restricts the effectiveness of checking the compatibility between them. In fact a simple comparison of the syntactic descriptions of security assertions is prone to get fault negative results. For example, consider syntactically different security assertions with the same meaning. Such assertions are considered incompatible which is counterintuitive. Besides, syntactic matching of security assertions only yields a strict yes/no matching result. A more flexible matching with intermediary matching degrees is needed in order to consider subtle differences that may exist between security assertions and not reject a potential partnership between a service requestor and provider. For example consider the cases when the provider security assertion and the requestor security assertion have the same type but have some different properties that make the provider assertion stronger, from security perspective, than the requestor assertion. Such assertions are considered incompatible which is overly strict.

In this paper we propose an approach to enable semantic matching of Web service security assertions. Firstly, we transform WS-SP into an ontology. Secondly, we extend this WS-SP-based ontology with new semantic relations, such as isEquivalentTo and isStrongerThan relations, between the provider and requestor security assertions. The additional relations allow to semantically interpret the syntactic heterogeneities that may exist between these assertions, particularly when the provider and the requestor security assertions point to the same ontological concept but have different properties. We define a set of SWRL[4] based semantic rules in order to control the dynamic instantiation of the additional semantic relations. Then, we propose an algorithm for matching security assertions.

We implemented a prototype for the semantic matching of security assertions. We used this prototype to match several syntactically different but semantically related assertions and obtained the expected matching degrees between them. Based on the semantic interpretation of the syntactic heterogeneities that may occur between a provider assertion and a requestor assertion, our approach doesn't produce fault negative results and thus supports more correct matching. Besides, it allows to introduce close match and possible match as intermediary matching degrees, which makes security assertion matching more flexible.

The rest of the paper is structured as follows. In Section 2 we present a motivating example to show the need for semantics in matching a pair of requestor-provider assertions. In Section 3 we describe how WS-SP can be extended to incorporate semantics. We firstly present the ontological representation of WS-SP, and then detail the new semantic relations that we propose in order to

semantically interpret the syntactic heterogeneities that may occur between security policy assertions. In Section 4 we present our algorithm for matching security assertions. Section 5 focuses on the implementation of our approach and its application to match examples of assertions. Related work is presented in section 6. Finally, we conclude in Section 7 and give the guidelines of our future work.

2 The Need for Semantics in Matching Security Assertions

Syntactic matching of security assertions restricts the effectiveness of checking the compatibility between them. In order to illustrate this deficiency, suppose that a requestor is looking for a flight reservation Web service that supports the signature of the message body with a symmetric key securely transported using an X509 token. Besides, the necessary cryptographic operations must be performed using Basic256 algorithm suite. This could be formalized, based on the WS-SP standard, by adding the assertions *RAss1* and *RAss2* from Fig. 1 to the requestor security policy (SP). Furthermore, suppose that the requestor finds a Web service that provides flight reservation and whose SP includes *PAss1* and *PAss2* assertions (see Fig. 1). In order to know if the requestor and provider SPs are compatible, we have to check the compatibility of their assertions. It is clear that the assertions specified in the provider SP are syntactically different than those specified in the requestor SP. Syntactic matching will then produce a no match result for these assertions. However, semantic interpretation of the above syntactic heterogeneities leads to decide a different matching result. In fact, in the above scenario *RAss1* and *PAss1* assertions have the same meaning: sign the body of the message. Therefore, matching these two assertions must lead to a perfect match rather than to a no match. Besides, the only differ-ence between *RAss2* and *PAss2* assertions is that the `SymmetricBinding` as-sertion specified in the provider SP contains an extra child element which is

RAss1:
```
<sp:SignedEdElements>
   <sp:XPath>
      /Envelope/Body
   </sp:XPath>
</sp:SignedEdElements>
```

RAss2:
```
<sp:SymmetricBinding>
   <sp:ProtectionToken>
      <sp:X509Token/>
   </sp:ProtectionToken>
   <sp:AlgorithmSuite>
      <sp:Basic256/>
   </sp:AlgorithmSuite>
</sp:SymmetricBinding>
```

PAss1:
```
<sp:SignedParts>
   <sp:Body/>
</sp:SignedParts>
```

PAss2:
```
<sp:SymmetricBinding>
   <sp:ProtectionToken>
      <sp:X509Token/>
   </sp:ProtectionToken>
   <sp:AlgorithmSuite>
      <sp:Basic256/>
   </sp:AlgorithmSuite>
   <sp:IncludeTimeStamp/>
</sp:SymmetricBinding>
```

Fig. 1. Example of syntactically different but semantically related security assertions

`sp:IncludeTimestamp`. The meaning of this element is the following: a timestamp element must be included in the security header of the message. From security perspective this strengthens the integrity service ensured by the message signature [16] and thus makes *PAss2* assertion stronger than *RAss2* assertion. Although it is not a perfect match, it is also overly strict to consider that it is a no match case. In fact, if the requestor can strengthen his security assertion by the inclusion of a timestamp, the perfect compatibility between both assertions will be ensured. We consider that it is more flexible to decide a possible match for this case in order to not reject a potential partnership between the service requestor and provider.

We will show in the rest of this paper how the inclusion of semantics into WS-SP enables more correct and flexible matching of security assertions.

3 Extending WS-SecurityPolicy to Incorporate Semantics

The assertions defined in WS-SP must be augmented with semantic information in order to enable semantic matching. Below we describe how we transform WS-SP into an ontology and detail the semantic relations that we add at the level of the security concepts.

3.1 WS-SecurityPolicy-Based Security Policy Ontology

We redesign WS-SP with an ontological representation of its assertions in order to obtain a WS-SP-based ontology that can be augmented with new semantic relations. A graphical depiction of the main parts of the ontology is shown in Figures 2 to 5. The blue coloured object properties represent the additional semantic relations and will be explained in the next subsection.

Web service security assertions are specified within security policies of the service provider and requestor. Typically, the structure of these policies is compliant with the WS-Policy normal form. In the normal form, a policy is a collection of alternatives, and an alternative is a collection of assertions. It is in the assertion components that a policy is specialized. Fig. 2 shows the main classes of the WS-SP-based ontology. As it is illustrated in this figure, we create the three classes *SecurityPolicy*, *SecurityAlternative* and *SecurityAssertion* in order to enable specifying security assertions within security policies. Then, we transform WS-SP assertions into classes and properties in the security policy ontology based on the semantic meaning of these assertions. *SecurityBinding*, *SupportingSecurityTokens*, *TokenReferencingAndTrustOptions*, and *ProtectionScope* are the security assertions of our ontology and are modelled as subclasses of the *SecurityAssertion* class. Due to space limitations, we will focus, in the rest of the paper on *SecurityBinding* and *ProtectionScope* classes.

SecurityBinding class: represents the various security bindings defined in WS-SP (see Fig. 3). This class allows specifying the main security mechanism to apply for securing message exchanges. Security binding can be either transport level, represented by *TransportBinding* class, or message level represented by

MessageSecBinding class that possesses the two subclasses *SymmetricBinding* and *AsymmetricBinding*. Security bindings have common properties of the following types:

- *SecurityHeaderLayout*:security header layouts are represented by instances of this class. *LayoutValue* is a string data type property having a possible value among "Strict", "Lax", "LaxTsFirst", and "LaxTsLast".
- *AlgorithmSuite*:algorithm suites are captured by this class. The possible values of the string property *AlgSuiteValue* are all algorithm suites defined in WS-SP such as Basic256, Basic192, and Basic256Rsa15 algorithm suites.
- *Timestamp*: the presence of a timestamp in the SOAP message is captured by this class having a boolean data type property called *IncludeTimestamp*.

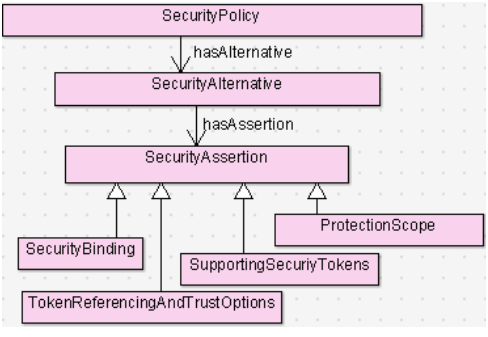

Fig. 2. Security policy ontology: main classes

Message level security bindings have also some common properties such as protection order, signature protection, and token protection that are modelled as data type properties or object properties of the *MessageSecBinding* class.

Besides, each specific type of security binding has binding specific token properties modelled by object properties such as *hasTransportToken*, *hasSignatureToken*, *hasProtectionToken*,etc. These object properties relate a security binding to the *SecurityToken* class (see Fig. 4). This class allows to specify the type of tokens to use to protect or bind claims to the SOAP message. There are different types of tokens with different manners to attach them to messages. Fig. 4 just shows some token types.

ProtectionScope class: is used to group and organize the protection assertions of WS-SP. Fig. 5 shows the property layers of this class. The first level properties are of the following types:

- *EncryptionScope*:this class is used to specify message parts that are to be protected by encryption. Its properties *hasEncryptedPart*, *hasEncryptedElt*, and *hasContentEncryptedElt* allow to specify coarse-grained as well as fine-grained encryption. The *EncryptedPart* class represents a message part to be encrypted. "Body", "Header" and "Attachments" are the possible values of its *PartType* property.

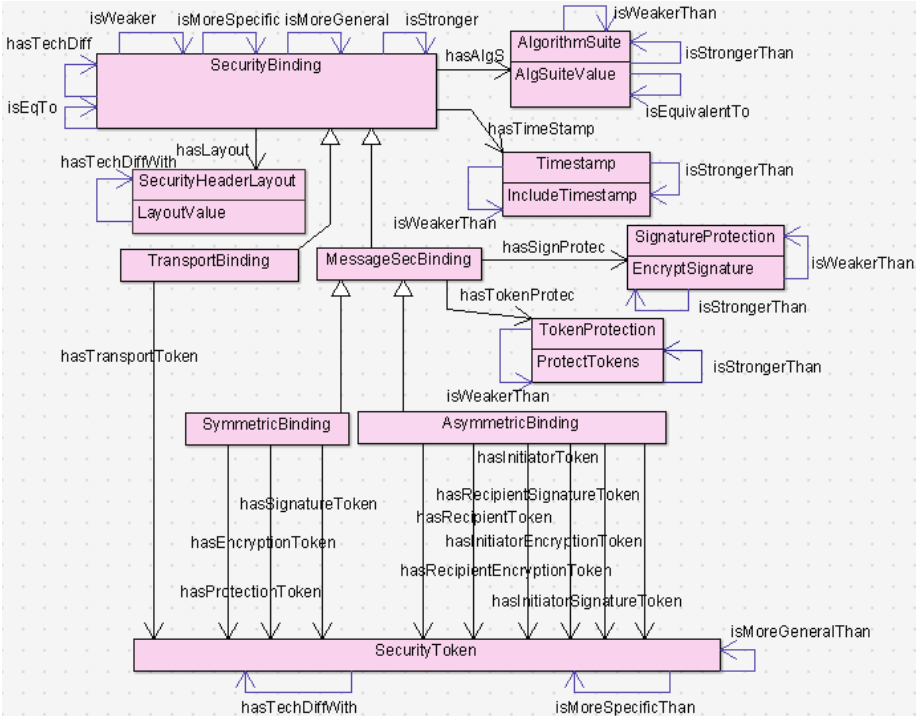

Fig. 3. Security policy ontology: security binding

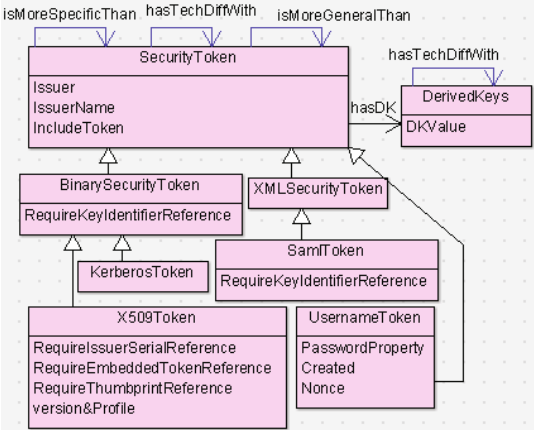

Fig. 4. Security policy ontology: security token

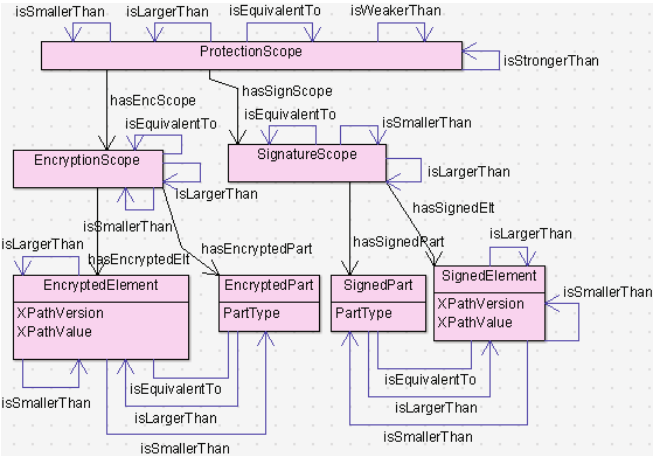

Fig. 5. Security policy ontology: protection scope

– *SignatureScope*:through its properties *hasSignedPart* and *hasSignedElt*, this class allows to describe message parts that are to be digitally signed.
– *RequiredScope*(not shown in the figure due to space limitations): this class is used to specify the set of header elements that a message must contain.

3.2 Adding Semantic Relations

We augment the WS-SP-based ontology with new semantic relations at the level of the security assertions and their properties. These relations are graphically illustrated in Figures 2 to 5 as blue coloured object properties of the ontology classes. They actually link the security concepts of the provider SP to those of the requestor SP.

The automatic instantiation of the semantic relations between concrete provider and requestor security assertions (mapped to instances of the security ontology concepts) is controlled by SWRL [4] based semantic rules. These rules define the conditions that the concrete assertions must satisfy in order to create a given semantic relation between them.

Next, we detail the semantic relations and present (in natural language) some of the semantic rules that control their instantiation.

isIdenticalTo relation[1] . This relation allows to specify that a security concept specified by the provider is identical to a security concept specified by the requestor (no syntactic heterogeneity exists between them). There are two main cases for this relation:

Case 1. It is defined at the level of every security property of ProtectionScope and SecurityBinding assertions. At the level of an atomic property such as

[1] *isIdenticalTo* and *isDifferentFrom* relations are not shown in the figures to avoid saturating them.

Timestamp and AlgorithmSuite, a property specified in the provider assertion is identical to a property specified in the requestor assertion if they point to the same security concept and have equal values. At the level of a composite property such as *SecurityToken* and *EncryptionScope*, a property specified in the provider assertion is identical to a property specified in the requestor assertion if they point to the same security concept and all their child properties are identical (have isIdenticalTo relation).
Case 2. *isIdenticalTo* relation is defined at the level of *ProtectionScope* and *SecurityBinding* assertions. A provider security assertion is identical to a requestor security assertion, if they point to the same security concept and all their properties are identical.

isEquivalentTo relation. This relation allows to specify that a security concept specified by the provider is equivalent to a security concept specified by the requestor. There are three ways this relation can occur:

Case 1. It is defined between an encrypted part and an encrypted element, similarly between a signed part and a signed element, and between a required part and a required element. This is because a message part can also be referenced through an XPath expression. For example, the XPath expression /S:Enveloppe/S:Body references the body part of a SOAP message. The following is one of the semantic rules representing this equivalence case:
If there exists, in the requestor policy, a signed element *reqSignedElt*, which has its XPathValue property having the value "/S:Enveloppe/S:Body", and **if** there exists, in the provider policy, a signed part *PSignedPart*, which has its PartType property, which has the value "Body", **then** create an *isEquivalentTo* relation between *PSignedPart* and *RSignedElt*.
Case 2. *isEquivalentTo* relation is defined between two algorithm suites. An algorithm suite denes which algorithms must be used when cryptographic operations such as signing, encryption, and generating message digests are involved. The algorithm suites dened in WS-SP are not disjoint and two algorithm suites can include many common elements. For example, Basic256 and Basic256Sha256 algorithm suites differ only by the digest algorithm. Therefore, if one of these two algorithm suites is specied in the requestor SP and the other is specied in the provider SP and neither of the two SPs involves digest generation, then the two algorithm suites can be considered as equivalent in this context.
Case 3. This case is a consequence of cases 1 and 2. In fact, if the equivalence relation actually exists at the level of the aforementioned concepts of cases 1 and 2, it spreads to the upper concepts of the security ontology. It can so exist between two encryption scopes, two signature scopes, and two required scopes. Then, it can exist at the level of *SecurityBinding* and *ProtectionScope* assertions. For example, an assertion *PProtecScope* specified by the provider is equivalent to an assertion *RProtecScope* specified by the requestor if it has at least one property (encryption, signature, or required scope) that is equivalent to the corresponding property of RProtecScope, and the other properties of *PProtecScope* and *RProtecScope* are identical.

isLargerThan relation. This relation only concerns the protection scope concept. There are three cases for this relation:

Case 1. Since the body part of a SOAP message is larger than any element that belongs to it, also a header part is larger than any of its sub-elements, we dene *isLargerThan* relation between an encrypted part and an encrypted element, between a signed part and a signed element, as well as between a required part and a required element. Similarly, since any XML element is larger than any of its sub-elements, we furthermore dene isLargerThan relation between two encrypted elements, two signed elements, and between two required elements.
Case 2. *isLargerThan* relation can also exist at the level of encryption scopes, signature scopes, and required scopes. For example an encryption scope *PEncScope* specied by the provider is larger than an encryption scope *REncScope* specied by the requestor if: 1) it has at least an encrypted part or an encrypted element which is larger than an encrypted element of *REncScope*, the other encrypted parts and encrypted elements of *PEncScope* and *REncScope* are identical or equivalent, or 2) it has at least one extra encrypted part or encrypted element than *REncScope*, the other encrypted parts and encrypted elements of *PEncScope* and *REncScope* are identical, equivalent, or have *isLargerThan* relations.
Case 3. At the level of the protection scope concept, a protection scope *PProtecScope* specied by the provider is larger than a protection scope *RProtecScope* specied by the requestor if it has at least one property (encryption, signature, or required scope) that is larger than a property of *RProtecScope*, and the other properties of *PProtecScope* and *RProtecScope* are identical or equivalent. Note that in order to obtain correct *isLargerThan* relations, any syntactic heterogeneity between XML data referenced in the protection scopes must be resolved [10] before executing the involved semantic rules.

isSmallerThan relation. This relation is to specify that the protection scope (in the SOAP message) specified by the provider is smaller than the protection scope specified by the requestor. It is the opposite of *isLargerThan* relation and occurs in three cases just in an opposite manner to *isLargerThan* relation.

isStrongerThan relation. This relation is to specify that a security concept specified by the provider is stronger, in the security perspective, than a security concept specified by the requestor. There are three ways this relation can occur:

Case 1. Since some security binding properties such as algorithm suite, timestamp, signature protection, and token protection have influence on the security strength, we add *isStrongerThan* relation at the level of these properties (see Fig. 3). For instance, the following semantic rule defines when a timestamp property specified in the provider security binding assertion is stronger than a timestamp property specified in the requestor security binding assertion:
If there exists, in the requestor policy, a security binding *RSecB*, and **if** there exists, in the provider policy, a security binding *PSecB*, and if *PSecB* and *RSecB* have the same class of binding (transport, symmetric, or asymmetric), and if *RSecB* has a timestamp *RTs*, which has its *IncludeTimestamp* property which has the value "false", and **if** *PSecB* has a timestamp *PTs*, which has its

IncludeTimestamp property which has the value "true", **then** create an *isStrongerThan* relation between *PTs* and *RTs*.
Case 2. At the level of a security binding assertion, an assertion *PSecB* specified by the provider is stronger than an assertion *RSecB* specified by the requestor if it points to the same type of security binding as *RSecB* but it has at least one property that is stronger than the corresponding property of *RSecB*, and the other properties of *PSecB* and *RSecB* are identical or equivalent.
Case 3. *isStrongerThan* relation is dened between two protection scopes. A protection scope *PProtecScope* specified by the provider is stronger than a protection scope *RProtecScope* specified by the requestor if it has at least one property (encryption, signature, or required scope) that isn't specied in *RProtecScope*, and the other properties of *PProtecScope* and *RProtecScope* are identical, equivalent, or have *isLargerThan* relations.

isWeakerThan relation. This relation is to specify that a security concept specified by the provider is weaker, in the security perspective, than a security concept specified by the requestor. It is the opposite of *isStrongerThan* relation and occurs in three cases just in an opposite manner to *isStrongerThan* relation.

hasTechDiffWith relation. In addition to concepts that allow to specify how a SOAP message is to be secured (confidentiality, integrity,), WS-SP-based ontology also includes concepts to describe technical aspects concerning how adding and referencing the security features in the message. At the level of these technical concepts, we define *hasTechDiffWith* relation to state that any mismatch between the provider concept properties and the requestor concept properties must be considered as a technical mismatch rather than a security level mismatch. There are three cases for this relation:

Case 1. Since the security header layout property simply defines which layout rules to apply when adding security items to the security header and has no influence on the security services ensured by a SP, we added *hasTechDiffWith* relation at the level of the *SecurityHeaderLayout* concept (see Fig. 3). The following semantic rule controls the instantiation of *hasTechDiffWith* relation at the level of the *SecurityHeaderLayout* concept:
If there exists, in the requestor policy, a security binding *RSecB*, and if there exists, in the provider policy, a security binding *PSecB*, and **if** *PSecB* and *RSecB* have the same class of binding, and if *RSecB* has a securityHeaderLayout *RSHL*, which has a LayoutValue property *RHLValue*, and **if** *PSecB* has a SecurityHeaderLayout *PSHL*, which has a LayoutValue property *PHLValue*, and if *RHLValue* and *PHLValue* are not equal, **then** create a *hasTechDiffWith* relation between *PSHL* and *RSHL*.
Case 2. Similarly some child properties of the security token property have no influence on the security services ensured by a SP. So we also add *hasTechDiffWith* relation at the level of these subproperties as well as at the level of the *SecurityToken* concept. For instance, we defined *hasTechDiffWith* relation at the level of *DerivedKeys* property since this property just specifies which mechanism must be applied in order to reference, in the SOAP message, derived keys used

in cryptographic operations and it has no influence on which cryptographic operations to use or with which strength they must be applied.
Case 3. At the level of a security binding assertion, an assertion *PSecB* specified by the provider has technical difference with an assertion *RSecB* specified by the requestor if it points to the same type of security binding as *RSecB* but it has at least one property (*SecurityHeaderLayout* or *SecurityToken*) that has technical difference with the corresponding property of *RSecB*, and the other properties of *PSecB* and *RSecB* are identical or equivalent.

isMoreSpecificThan relation. According to WS-SP standard, many security properties are optional to specify in a SP and WS-SP doesn't attribute default values for them. Therefore we define *isMoreSpecificThan* relation that occurs when a security concept specified by the provider is more specific (i.e., described in more detail) than a security concept specified by the requestor. There are two cases for this relation:

Case 1. It occurs between two *SecurityToken* properties that point to the same concept but the security token specified by the provider is described in more detail than the security token specified by the requestor. For example, the security token specified by the provider is an X509 token that has its property *Version&Profile* having the value "WssX509V3Token10" (which means that an X509 Version 3 token should be used as defined in WSS:X509TokenProfile1.0 [5]), while the security token specified by the requestor is an X509 token with no properties.
Case 2. *isMoreSpecificThan* relation is defined at the level of a security binding assertion, an assertion *PSecB* specified by the provider is more specific than an assertion *RSecB* specified by the requestor if: 1) it points to the same type of security binding as *RSecB* but it has a security token property that is more specific than the corresponding security token property of *RSecB*, and the other properties of *PSecB* and *RSecB* are identical or equivalent, or 2) it points to the same type of security binding as *RSecB* and has a *SecurityHeaderLayout* property (which is an optional property without a default value), while *RSecB* hasn't a *SecurityHeaderLayout* property, and the other properties of *PSecB* and *RSecB* are identical or equivalent.

isMoreGeneralThan relation. This relation occurs when a security concept specified by the provider is more general (i.e., described in less detail) than a security concept specified by the requestor. It is the opposite of isMoreSpecificThan relation and oc-curs in two main cases just in an opposite manner to isMoreSpecificThan relation.

isDifferentFrom relation. This relation occurs when the security concepts specified by the requestor and the provider are semantically disparate. There are two main cases for this relation:

Case 1. It is defined at the level of *AlgorithmSuite* and *SecurityToken* properties of *SecurityBinding* assertions. For example a security token specified by the provider as a username token is considered different from a security token

specified by the requestor as an X509 token. This relation is also defined at the level of all properties of *ProtectionScope* assertion. For example an encrypted part specified by the provider as a Body part is considered different from an encrypted part specified by the requestor as a Header part.
Case 2. *isDifferentFrom* relation is defined at the level of *ProtectionScope* and *SecurityBinding* assertions. A provider security assertion is different from a requestor security assertion if: 1) they point to different security concepts, or 2) they point to the same security concept, and they have at least one *isDifferentFrom* relation at the level of their properties, and their remaining properties are linked with semantic relations of type *isIdenticalTo* or *isEquivalentTo*.

4 Semantic Matching of Security Assertions

In this section, we propose an algorithm for matching provider and requestor security assertions. The matching process consists in checking to what extent each security assertion *RAss* specified in the requestor SP is satisfied by a security assertion *PAss* specified in the provider SP. The matchmaker has to perform two main tasks. Firstly, it must create all possible semantic relations at the level of each pair of provider and requestor assertions. This is done through the execution of the semantic rules presented in the previous section. Secondly, based on the created semantic relations, it must decide the appropriate matching degree for each *PAss-RAss* pair. The final matching degree for a requestor assertion is the highest level of match it has against all of the checked provider assertions. There are four possible matching degrees for a *PAss-RAss* pair: perfect match, close match, possible match, and no match in decreasing order of matching.

Perfect match. A perfect match occurs when *PAss* and *RAss* are connected through *isIdenticalTo* or *isEquivalentTo* relations.

Close match. A close match occurs when *PAss* and *RAss* are connected through *isMoreSpecificThan* relation. For example, suppose that *PAss* and *RAss* point to the *TransportBinding* concept, and *PAss* has a *SecurityHeaderLayout* property having the value "Lax", while *RAss* hasn't a *SecurityHeaderLayout* property, and the other properties of both assertions are identical.

The transport binding specified by the provider is described in more detail than the transport binding specified by the requestor. We assume that the requestor omits specifying the *SecurityHeaderLayout* property because he doesn't care which specific value is used for this property. Therefore, close match is an appropriate matching degree for this example.

Possible match. A possible match is decided in three main cases:

Case 1. *PAss* and *RAss* are connected through *isMoreGeneralThan* relation. This means that the information available can not ensure that *PAss* can perfectly match *RAss*. We assume that a potential partnership between the requestor and the provider can take place if the requestor can obtain additional information or negotiate with the provider.

Case 2. *PAss* is connected to *RAss* through *isLargerThan*, *isStrongerThan*, or *hasTechDiffWith* relations. This means that the incompatibility between the two assertions doesn't negatively affect the security services and levels required in RAss. We assume that a potential partnership between the requestor and the provider can take place if the requestor can strengthen his policy assertion or change some technical properties of his assertion.
Case 3. *PAss* and *RAss* point to the same security concept and have at least one *isMoreGeneralThan*, *isLargerThan*, *isStrongerThan*, or *hasTechDiffWith* relation at the level of their properties, and their remaining properties are linked with semantic relations of type *isIdenticalTo*, *isEquivalentTo*, or *isMoreSpecificThan*. For example, suppose that *PAss* and *RAss* point to the SymmetricBinding concept, but *PAss* has a protection token that is more general than the protection token specified in *RAss*. In addition, *PAss* has an algorithm suite that is identical to the algorithm suite specified in *RAss*. And finally, *PAss* has a Timestamp property that is stronger than the Timestamp property of *RAss*. The two assertions have two heterogeneities that don't rule out the possibility of a match, so it is a possible match case.

No match. No match is decided in two main cases:

Case 1. *PAss* and *RAss* are connected through *isDifferentFrom*, *isSmallerThan*, or *isWeakerThan* relations. For example, suppose that *PAss* points to the *SymmetricBinding* concept, while *RAss* points to the *ProtectionScope* concept. These two assertions specify two semantically unrelated concepts. Therefore, they must be linked by *isDifferentFrom* relation and then a no match must be decided for them.
Case 2. *PAss* and *RAss* point to the same security concept, and they have at least one *isDifferentFrom*, *isSmallerThan*, or *isWeakerThan* relation at the level of their properties.

5 Implementation and Application

In this section, we present implementation details and show how we use our approach to handle the flight reservation use case that was discussed in the second section. We implemented the WS-SP-based ontology WS-SP.owl in OWL-DL [6] semantic language with the Protégé OWL tool [7]. The OWL file can be found at `http://www.redcad.org/members/monia.benbrahim/WS-SP.owl`. Then we created *PSP.owl* and *RSP.owl* ontologies as specializations of WS-SP.owl ontology. *PSP.owl* and *RSP.owl* ontologies must be imported respectively by the provider and the requestor in order to specify their SPs. In addition, we created another ontology called *AssertionMatching.owl* that imports two OWL files representing the provider SP and the requestor SP. This ontology contains the concepts of *PSP.owl* with the prefix `provider` and the concepts of *RSP.owl* with the prefix requestor. In *AssertionMatching.owl* ontology, we added all the new semantic relations previously detailed in section 3.2 as well as all the necessary SWRL rules for their dynamic instantiation. For example the following is the SWRL rule corresponding to the first semantic rule presented in section 3.2:

```
requestor: SignedElement(?RSignedElt) ∧
XPathValue(?RSignedElt, "/S:Enveloppe/S:Body") ∧
provider: SignedPart(?PSignedPart) ∧
PartType(?PSignedPart, "Body")→
isEquivalentTo(?PSignedPart, ?RSignedElt)
```

We also implemented the matching algorithm described in the previous section as a set of SWRL rules and added them to *AssertionMatching.owl* ontology.

The assertions contained in a concrete (requestor/provider) SP are mapped to in-stances of *AssertionMatching.owl* ontology. We use the Jess rule engine [8] to exe-cute the SWRL rules and automatically generate, in a first step, the involved semantic relations between each pair of provider assertion-requestor assertion; and generate, in a second step, the appropriate matching degrees between the assertions.

In order to apply our approach to the use case presented in section 2, we used the *RSP.owl* ontology to transform the requestor SP that was written in WS-SP language into the semantic SP *RSPEx1.owl*[2] . Similarly, based on the *PSP.owl* ontology, the provider SP is semantically described in *PSPEx1.owl*[3]. Then an *AssertionMatching.owl* file that imports *RSPEx1.owl* and *PSPEx1.owl* is generated. After running the Jess engine, a set of semantic relations and matching degrees are created between the provider and the requestor assertions. This is an extract from the new knowledge that was added in the *AssertionMatching.owl* file after the execution of the SWRL rules:

```
<rdf:Description rdf:about="http://.../PSPEx1.owl#PSignedPart">
   <isEquivalentTo rdf:resource="
    "http://.../RSPEx1.owl#RSignedElt"/>
</rdf:Description>
<rdf:Description rdf:about="http://.../PSPEx1.owl#PSymBinding">
    <isStrongerThan rdf:resource=
    "http://.../RSPEx1.owl#RSymBinding"/>
</rdf:Description>
<rdf:Description rdf:about="http://.../PSPEx1.owl#PAss1">
    <PerfectMatch rdf:resource=
    "http://.../RSPEx1.owl#RAss1"/>
</rdf:Description>
```

The application of our semantic approach to match the security assertions specified in the flight reservation use case allows to obtain the expected matching degree at the level of each requestor assertion-provider assertion pair as well as the expected final matching degree at the level of each requestor assertion.

[2] Available at
http://www.redcad.org/members/monia.benbrahim/useCase/RSPEx1.owl
[3] Available at
http://www.redcad.org/members/monia.benbrahim/useCase/PSPEx1.owl

6 Related Work

Several works dealt with the enrichment of WS-Policy with semantics using OWL ontologies. The authors of [18] propose to create the policy assertions based on terms from ontologies instead of XML schema based vocabularies. In [17], Speiser proposes a technique for annotating policy assertions with semantic references as it is done for SAWSDL [9]. Chaari et al. [11] redesign WS-Policy with an ontological representation of concepts and relations. They also define a general QoS ontology and integrate it with the WS-policy ontology. All these contributions just give an idea on how to augment WS-Policy with semantics, but they did not indicate how to define specific domain ontologies such as security ontology. The work described in [15] consists in mapping WS-Policy and related specifications such as WS-SP to OWL-DL. In this work, the authors systematically map each policy assertion to an ontology class. In our approach, we rather take into account the semantic meaning of security policy assertions and make several semantic classifications of security assertions. In addition, we transform some assertions into data type and object properties and not into classes in order to explicit the semantic relationships between assertions.

In the area of Web service security, works such as [12], [14], and [13] propose a security ontology to describe the security requirements and capabilities of Web service providers and requestors. Although these ontologies include concepts related to SOAP message security, they consider neither the message security model defined by WS-Security standard nor the security assertions defined in WS-SP standard. Besides these approaches have the drawback of not being compatible even to WS-Policy. The work proposed by Garcia et al [19] mainly deals with integrity and confidentiality of SOAP messages. A security ontology that considers the WS-Security message security model is proposed and constitutes a foundation to support WS-Policy with semantics. However, this ontology doesn't include concepts to describe many security as-pects such as transport level security and supporting tokens. Moreover, the security concepts used in the ontology are not equivalent to WS-SP assertions.

In addition to being compatible to WS-SP, our approach is distinguished from all pre-vious approaches by the fact that we extend the WS-SP-based ontology with additional semantic relations and semantic rules that support more correct and more pre-cise semantic matching of Web service security policies.

7 Conclusion and Future Work

In this paper, we presented an approach to provide a semantic extension to security assertions of Web services. The approach is based on the transformation of WS-SP into an OWL-DL ontology. We showed, through a top down methodology, how we transform WS-SP assertions into classes and properties in the ontology. Besides, in order to support semantic matching of these assertions,

we extended the domain knowledge with semantic relations that can exist between requestor and provider security concepts. These relations are dynamically instantiated through the execution of SWRL rules. Moreover, we proposed an algorithm for matching security assertions. Our semantic approach supports more correct and more flexible security assertion matching compared to syntactic matching as well as to previous works that combined ontologies and Web service security properties. The on going work is the completion of our matching algorithm in order to decide about the final matching degree between complex security policies containing several security alternatives and assertions. Besides, we plan to develop a tool that automatically transforms a WS-SP policy into our ontological representation.

References

1. OASIS: WS-Security 1.1, http://www.oasis-open.org/specs/
2. OASIS: WS-SecurityPolicy 1.3, http://www.oasis-open.org/specs/
3. WS-Policy 1.5, http://www.w3.org/TR/ws-policy/
4. W3C: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/
5. OASIS: Web Services Security X.509 Certificate Token Profile, http://www.oasis-open.org/specs/
6. W3C: OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/
7. OWL protege Web page, http://protege.stanford.edu/overview/protege-owl.html
8. The Jess engine Web page, http://www.jessrules.com/
9. W3C: Semantic Annotations for WSDL and XML Schema, http://www.w3.org/TR/sawsdl/
10. Ben Brahim, M., Ben Jemaa, M., Jmaiel, M.: Security Mapping to Enhance Matching Fine-Grained Security Policies. In: Zavoral, F., Yaghob, J., Pichappan, P., El-Qawasmeh, E. (eds.) NDT 2010, Part I. CCIS, vol. 87, pp. 183–196. Springer, Heidelberg (2010)
11. Chaari, S., Badr, Y., Biennier, F.: Enhancing web service selection by qos-based ontology and ws-policy. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 2426–2431. ACM (2008)
12. Denker, G., Kagal, L., Finin, T.W., Paolucci, M., Sycara, K.: Security for DAML Web Services: Annotation and Matchmaking. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 335–350. Springer, Heidelberg (2003)
13. He, Z., Wu, L., Hong, Z., Lai, H.: Semantic security policy for web service. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 258–262. IEEE Computer Society (2009)
14. Kim, A., Luo, J., Kang, M.: Security Ontology for Annotating Resources. In: Meersman, R., Tari, Z. (eds.) OTM 2005, Part II. LNCS, vol. 3761, pp. 1483–1499. Springer, Heidelberg (2005)
15. Kolovski, V., Parsia, B., Katz, Y., Hendler, J.: Representing Web Service Policies in OWL-DL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 461–475. Springer, Heidelberg (2005)

16. Ono, K., Nakamura, Y., Satoh, F., Tateishi, T.: Verifying the consistency of security policies by abstracting into security types. In: Proceedings of the 2007 IEEE International Conference on Web Services, pp. 497–504. IEEE Computer Society (2007)
17. Speiser, S.: Semantic annotations for ws-policy. In: Proceedings of the 2010 IEEE International Conference on Web Services, pp. 449–456. IEEE Computer Society (2010)
18. Verma, K., Akkiraju, R., Goodwin, R.: Semantic matching of web service policies. In: Proceedings of the Second Workshop on Semantic and Dynamic Web Processes, pp. 79–90 (2005)
19. Zuquim Guimaraes Garcia, D., Beatriz Felgar de Toledo, M.: Ontology-based security policies for supporting the management of web service business processes. In: Proceedings of the 2th IEEE International Conference on Semantic Computing, pp. 331–338. IEEE Computer Society (2008)