Transformation and Aggregation of Web Service Security Requirements

Robert Warschofsky, Michael Menzel, Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
{*robert.warschofsky, michael.menzel, meinel*}*@hpi.uni-potsdam.de*

***Abstract*—Service-oriented Architectures support the provision, discovery, and usage of services in different application contexts. The Web Service specifications provide a technical foundation to implement this paradigm and provide mechanisms to face the new security challenges raised by SOA. To enable the seamless usage of services, security requirements can be expressed as security policies (e.g. WS-Policy and WS-SecurityPolicy) that enable the negotiation of these requirements between clients and services. However, the concept of policy negotiation has not been applicable in the scope of service compositions so far. Since each orchestrated Web Service in a service composition might demand the provision of specific user information and requires a particular security mechanism, the security policy of a service composition depends on the aggregated requirements of the orchestrated services. Current Web Service frameworks are not capable of resolving such policy dependencies.**

In this paper we present our solution to enable an automated creation of security policies from orchestrated services. Therefore, we present a policy model that is capable of capturing Web Service security requirements. Based on this model, we introduce an algorithm that performs the aggregation of security requirements stated by the orchestrated services and mappings to transform WS-SecurityPolicy instances and the security model instances into each other.

***Keywords*-Service-oriented Architectures; SOA Security; Policy Generation; WS-SecurityPolicy**

I. Introduction

Service-oriented Architectures (SOA) enables the provision of business logic as independent services. These services are loosely coupled, reusable, and can be discovered and bound on demand using appropriate service descriptions, which enables the flexibility of SOA. The Web Service specifications such as SOAP [1] and WSDL [2] provide the technical foundation to implement these concepts. In addition, specifications such as WS-Security can be used to ensure the confidentiality, integrity and authenticity of exchanged messages.

To sustain the flexibility and interoperability regarding the integration of security mechanisms, security requirements of services (e.g. confidentiality) have to be exposed as policies next to the services' interface descriptions. Clients can retrieve interface descriptions enhanced with security policies from a service and use the required security mechanisms to secure the service request. This approach enables interoperability at run-time between service consumer and service provider.

Next to simple service invocations, SOA fosters the creation of complex service compositions. Such a composition is exposed as a service itself and leverages independent services to provide its functionality. Kanneganti et al. [3] differentiate between simple compositions that redirect incoming requests by invoking the lower level services and more sophisticated compositions that run complex business processes. In any case, the orchestrated services may have their own security requirements expressed as security policies. These requirements have to be fulfilled by the composed service or clients using the composed service. In the second case security requirements (e.g. the provision of security tokens) have to be fulfilled by a Web Service client across a service composition.

This means that a client has to fulfill the security policy of a service composition that depends on the security requirements of the invoked services. Therefore, the security policy of the service composition must represent an aggregation of the security requirements of the orchestrated services, as shown in Figure 1. However current frameworks and process engines that can be used to create composed services do not provide support to generate the security policy of a service composition automatically. The security policy of a composed service needs to be defined by a developer in a manual manner. In addition, there is a lack of frameworks that can be used to generate and handle WS-SecurityPolicy due to its complexity.

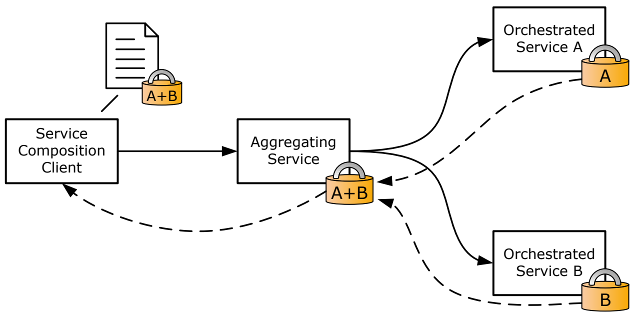


Figure 1. Invocation of a service composition that requires multiple security tokens for the different orchestrated services.

To simplify the creation of policies for composed services, we present in this paper:

- A policy meta-model that serves as an abstraction layer for Web Service security policies and simplifies the generation of these security policies.
- An algorithm that uses our policy model to aggregate and compose security requirements expressed by different security policies.

- A mapping between WS-SecurityPolicy and our security policy model

This paper is structured as follows. Section 2 introduces WS-Policy and WS-SecurityPolicy as the relevant standards to describe Web Service Security Policies. Our platform-independent policy model that is used to provide an abstraction layer for WS-Security is introduced in Section 3. Section 4 describes the mapping between our model and WS-SecurityPolicy. The aggregation of security policies expressed in our model is introduced in Section 5. Section 6 describes related work, while Section 7 concludes this paper.

II. Web Service security policy standards

There are many standards and approaches to express security policies for Web Services. In this paper we focus on the most widely used standards for this purpose, namely WS-Policy and WS-SecurityPolicy.

A. WS-Policy

WS-Policy [4] is an XML based framework, which specifies a model to express general policy requirements for Web Services. These policy requirements are expressed using policy assertions that contain a policy expression and optionally several assertion parameters. To comply to a policy all policy assertions in a policy have to be fulfilled. To express the requirement for more complex policy requirements, policy assertions can be nested.

The WS-Policy framework further defines two policy operators. The *All* operator specifies that all policy assertions inside the operator have to be fulfilled to fulfil that *All* operator. The *ExactlyOne* operator specifies that only one of the policy assertions inside the operator has to be fulfilled to fulfil the operator. Since these operators can also be nested, it is possible to describe alternative sets of policy assertions that have to be fulfilled to comply to a policy. There are no policy assertions specified by the WS-Policy standard, this has to be done by other standards.

WS-Policy also specifies a normalization algorithm to transform a policy into an equivalent normalized policy. Several WS-Policy specifications can be attached to a WSDL to declare polices for the different parts of a service definition. The rules for such an attachment are defined in the WS-PolicyAttachment standard [5].

B. WS-SecurityPolicy

The WS-SecurityPolicy standard [6] is based on the WS-Policy standard and specifies several policy assertions to express security requirements for Web Services.

The standard specifies assertions for several security requirements. Among them are assertions to express requirements about which parts of a message has to be encrypted or signed. There are also assertions to define which key has to be used to encrypt or sign message parts. Additionally, the standard specifies assertions to define which SOAP header or other message parts are required inside a message. Finally, there are assertions specified to express which security tokens are required in a SOAP message to comply to the policy.

The assertions are grouped into four categories, namely *Protection Assertions*, *Token Assertions*, *Security Binding Assertions*, and *Supporting Tokens*. *Protection Assertions* are describing which parts of a secured message have to be encrypted, signed, or simply present in the message.

The *Token Assertions* describe security tokens. A security token can be requested to have a specific issuer and a set of required claims.

The *Security Binding Assertions* are used to describe the security mechanisms used to secure messages between a sender and the receiver.

Supporting Tokens describe additional security tokens that have to be attached to a secured message. These tokens can be used to encrypt or sign parts of the message, in addition to the encryption and signature created with the security tokens in the binding.

III. A platform-independent model for security policies

In the scope of WS-Security [7], it must be considered that

1) WS-Policy and WS-SecurityPolicy provide a syntax, but do not define a semantic. In fact, WS-Policy enables the negotiation and intersection of requirements between client and service without the need to know what is actually expressed by an policy option. However, to enable a mapping from security intentions to security policies, the meaning of the different requirements, their relation to security goals and their dependencies must be well known.
2) requirements that are semantically equal (for instance the encryption of message parts using different keys) have to be expressed in different ways in WS-SecurityPolicy.

Our policy meta-model supports the expression of security requirements concerning communication related security goals, serves as an abstraction layer for security policy languages, simplifies the handling of security policies, and enables the generation of security configurations in different policy languages.

The meta-model consists of several components. The *Security Policy Base Meta Model* describes the correlation of the basic components of the model. The *Digital Identity Meta Model* is used to express information about a subject issued by a trusted party. Different *Security Constraints* are used to describe the requirements to fulfil a *Security Goal*.

A. Security Policy Base Meta Model and Digital Identity Meta Model

A policy in our policy meta-model consists of several *Policy Alternatives* (see Figure 2) and each alternative contains a list of *Security Constraints*, ordered in correspondence to the sequence in which they are added to the alternative.

A *Credential* is the main element of the *Digital Identity Meta Model* (see Figure 3) and is used to describe parts of a digital identity. Therefore, a *Credential* contains a

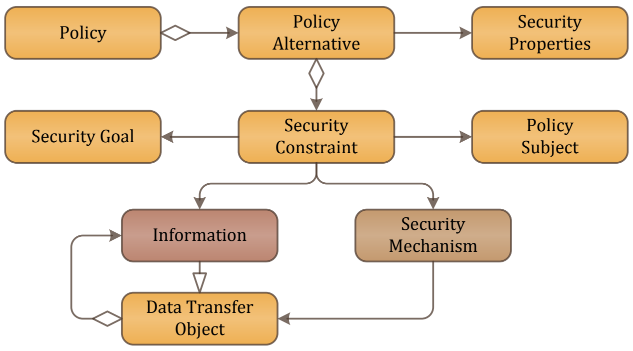


Figure 2. Security Policy Base Meta Model

set of *Claims*. A *Claim* is "a statement made about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.)." [8] It is required that for each *Claim* the corresponding information is added to the *Credential* in the secured message. Additionally, a *Credential* can contain *Authentication Information* that can be used to verify the authenticity of the *Credential*.

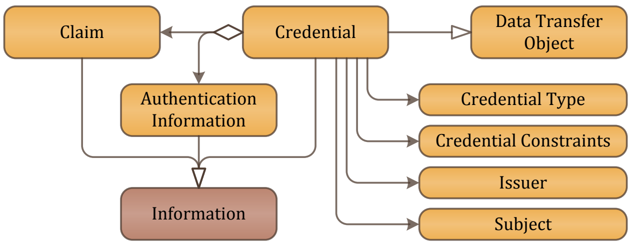


Figure 3. Digital Identity Meta Model

Each *Credential* also has a *Credential Type* that indicates its type (e.g *Username Token*, *SAML Assertions*, or *X.509 Certificate*). In addition, *Credentials* can have optional *Credential Constraints* that are used to further define the credential depending on its type. For example, it can be defined that a *Credential* of the type *Username Token* has to contain the users' password in a hashed form instead of plain text. Finally, each *Credential* has an issuer, identified by a unique URI. Depending on that issuer a receiver of the credential can decide whether the credential is trustworthy. The *Subject* element represents the subject of the *Credential*. For instance, the subject of a *Credential* of the type *Username Token* would be the user identified by the username.

B. Security Constraints Meta Model

In general, a *Security Constraint* describes a requirement to fulfil a *Security Goal*, such as confidentiality, integrity, authentication, or authorization. Moreover, a *Security Constraint* can contain information about what has to be secured and which security mechanism has to be used. To express which information has to be secured a *Data Transfer Object (DTO)* is used. A DTO is any part of information in a message, including the message body, any message header, the message itself, or a *Credential* as defined in the *Digital Identity Meta Model*. Finally, a DTO can contain several information objects, which again are DTOs.

The *Security Policy Base Meta Model* defines that a *Policy Alternative* contains a set of *Security Constraints*. There can be several *Security Constraints* in order to request the different *Security Goals* to be fulfilled. These *Security Constraints* are described in the following.

1) Data Security Constraint: Confidentiality and *Integrity* are two *Security Goals* that can be fulfilled using encryption and signatures. These cryptographic security mechanisms can be described using similar properties. Both *Security Mechanisms* are using a cryptographic algorithm with a cryptographic key to secure transferred data. The *Data Security Constraint* contains all information required to express the fulfilment of both *Security Goals*.

As shown in Figure 4, a *Data Security Constraint* contains *Data Transfer Objects* to identify the message or parts of the message. These *Data Transfer Objects* have to be secured using the requirements specified in the *Data Security Constraint*. In addition, the constraint specifies the *Security Protocol* (such as WS-Security or SSL), as well as, the algorithm type that has to be used to secure the data. The algorithm is determined using an algorithm suite defined by the WS-SecurityPolicy specification.

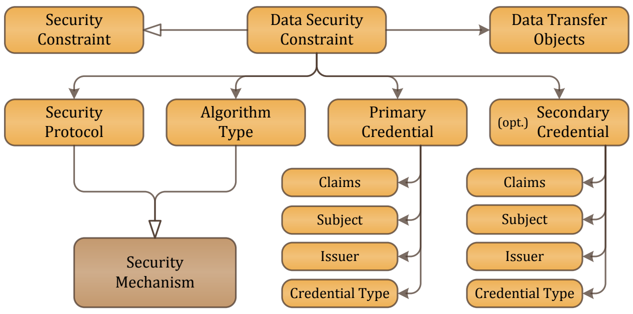


Figure 4. Data Security Constraint Meta-Model.

The cryptographic key that has to be used to secure the data is described as a *Credential* from the *Digital Identity Meta Model*. In this case, the *Credential* should be of a type that can be used to secure data, such as an *X.509 Certificate*. To determine, whether a *Data Security Constraint* describes a symmetric or an asymmetric algorithm, an optional second credential can be given. If a *Data Security Constraint* contains only one *Credential* a symmetric algorithm is described. An asymmetric algorithm is described with both *Credentials* set. The first *Credential* then describes the key for the sender and the second *Credential* describes the key for the receiver of a message.

2) Client Authentication Constraint: The *Authentication* of the sender of a message can be confirmed based on a set of statements about the identity of client that is conveyed within the message. These statements that are expressed as *Claims* have to be issued by a trustworthy party and are represented as a *Credential*. In addition, a *Credential* contains information about the issuer of the information. Therefore, a *Client Authentication Constraint* that is used to describe the requirements for the *Authentication Security Goal* contains a *Credential* (see Figure 5).

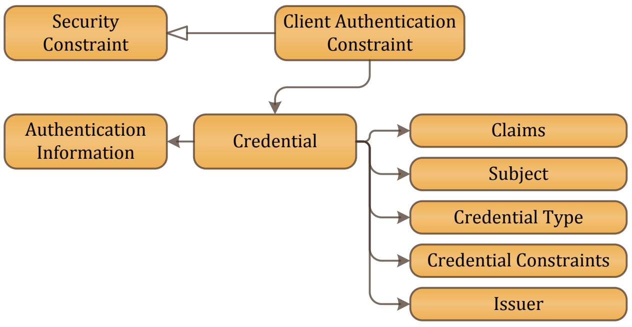


Figure 5. Client Authentication Constraint Meta-Model.

A *Client Authentication Constraint* requires that the sender of a message adds the identity information described in the credential of this constraint to that message. Therefore, the sender has to add a security token of that type that is defined in the *Credential* to the message. The token has to contain the information defined in the *Claims* of the *Credential* and it has to be issued by the party defined as *Issuer* in the *Credential*.

3) Required Information Constraint: A *Data Security Constraint* does not require that the information defined by the *Data Transfer Object* of the constraint has to be present in a secured message, but only that if the information defined by the *Data Transfer Objects* are present, the information has to be secured in the corresponding way. To define that a specific information is required in a message, a *Required Information Constraint* can be used.

As shown in Figure 6, a *Required Information Constraint* only contains a *Data Transfer Object* that describes the message part or information that has to be present in a secured message. The *Data Transfer Object* can describe the same information as the *Data Transfer Object* of a *Data Security Constraint*. It can also describe any other information or message parts, such as additional message header or security tokens.

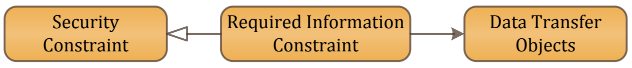


Figure 6. Required Information Constraint Meta-Model.

Since a *Client Authentication Constraint* already requires that the defined *Credential* has to be present in a secured message as a security token, the *Data Transfer Object* of an additional *Required Information Constraint* describing the same *Credential* can be omitted.

IV. Transformations of the policy meta-model

To make use of the advantages of the policy meta-model for the aggregation of WS-SecurityPolicy instances, a transformation between WS-SecurityPolicy and the policy meta-model is necessary. In this Section we present such transformations.

A. Transforming a WS-SecurityPolicy to the policy meta-model

Both, the WS-SecurityPolicy standard and the policy meta-model, provide the notion of a policy alternative which allows a separated transformation of these alternatives. The requirements of a policy alternative are transformed into a set of *Security Constraints* using three phases, as shown in Figure 7. The first phase covers the WS-SecurityPolicy binding assertions, the second phase covers the security expressions, and the third phase covers the supporting tokens.

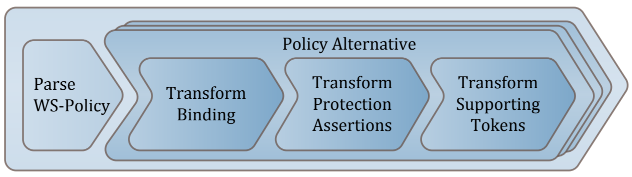


Figure 7. Transformation of a WS-SecurityPolicy to the model.

A WS-SecurityPoilcy *Binding Assertion* contains security token descriptions defining how a SOAP message has to be secured (e.g. signing and/or encrypting a part of a message). If a security token has to be used to sign a part of a SOAP message, an *Integrity Constraint* is created and for an encrypting security token a *Confidentiality Constraint* is created.

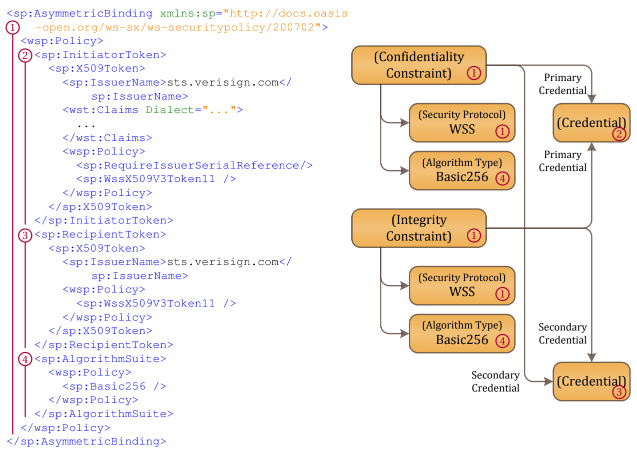
The security tokens stated in the binding can be used and referenced in other expressions as well, such as *EncryptedElements* assertions or *SignedSupportingTokens* assertions. Therefore, the *Confidentiality Constraint* and the *Integrity Constraint* created from these tokens have to be used later and are referred to as *Binding-Confidentiality Constraint* and *Binding-Integrity Constraint* in the scope of this Section.

The description of the token type of each security token is transformed into a *Credential* and this *Credential* is referenced in the *Security Constraints*. Depending on the type of the binding either one or two *Credentials* are used to describe the security tokens of the binding. Both *Security Constraints* resulting from a *TransportBinding* reference the same *Credential*, since the *TransportToken* is used to sign and encrypt the message. The constraints from a *Symmetric Binding* also contain only one *Credential*. Depending on the security token type, the *Credential* can be the same in both constraints (*ProtectionToken*) or it can be two different *Credentials* (*EncryptionToken* and *SignatureToken*). An *AsymmetricBinding* describes security tokens for the sender and the receiver of a message. Therefore, the constraints also contain two *Credentials*, the first for the sender and the second for the receiver.

The security tokens described in the binding are required to secure a message and the information to identify these tokens have to be sent within the message. To express this requirement in the model, the *Credentials* of the binding constraints are attached to an additional constraint. This additional constraint is either a *Client Authentication Constraint* or a *Required Information Con-*

straint. A *Required Information Constraint* expresses the requirement that the *Data Transfer Object* of the constraint has to be present in a message. A *Client Authentication Constraint* indicates that a *Credential* has to be used to authenticate the consumer of a service. In addition, this constraint indicates that the authenticating *Credential* has to be attached to a secured message.

Figure 8 illustrates the transformation of a WS-SecurityPolicy *AsymmetricBinding* into a set of *Security Constraints*. The tokens in the *InitiatorToken* and the *RecipientToken* of the *AsymmetricBinding* are transformed into the *Binding-Confidentiality Constraint* and the *Binding-Integrity Constraint*.



```
<sp:AsymmetricBinding xmlns:sp="http://docs.oasis
①  -open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:Policy>
② <sp:InitiatorToken>
      <sp:X509Token>
        <sp:IssuerName>sts.verisign.com</
            sp:IssuerName>
        <wst:Claims Dialect="...">
          ...
        </wst:Claims>
        <wsp:Policy>
          <sp:RequireIssuerSerialReference/>
          <sp:WssX509V3Token11 />
        </wsp:Policy>
      </sp:X509Token>
    </sp:InitiatorToken>
③ <sp:RecipientToken>
      <sp:X509Token>
        <sp:IssuerName>sts.verisign.com</
            sp:IssuerName>
        <wsp:Policy>
          <sp:WssX509V3Token11 />
        </wsp:Policy>
      </sp:X509Token>
    </sp:RecipientToken>
④ <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256 />
      </wsp:Policy>
    </sp:AlgorithmSuite>
  </wsp:Policy>
</sp:AsymmetricBinding>
```

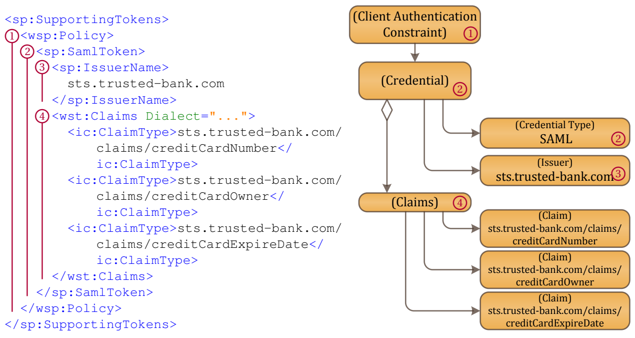
(Confidentiality Constraint) ①
(Security Protocol) WSS ①
(Algorithm Type) Basic256 ④
Primary Credential
(Credential) ②
Primary Credential
(Integrity Constraint) ①
(Security Protocol) WSS ①
(Algorithm Type) Basic256 ④
Secondary Credential
Secondary Credential
(Credential) ③

Figure 8. Asymmetric binding expressed as WS-SecurityPolicy XML on the left and as model elements on the right.

WS-SecurityPolicy *Protection Assertions* (e.g. encrypted and signed Parts) specify the parts of a message that have to be encrypted or signed using the binding security tokens. In the policy meta-model these requirements are described using *Data Transfer Objects* attached to the *Data Security Constraints* for the binding security tokens. The encrypted and signed *Parts* that can be included in a WS-SecurityPolicy alternative, describe the parts of a message that have to be secured using the security token described in the binding. In the policy meta-model this is described by *Data Transfer Objects* in the *Binding-Confidentiality Constraint* and the *Binding-Integrity Constraint* that refer to the signed and encrypted parts. As aforementioned, there are *Data Transfer Objects* to describe the message, the message body, a specific message header, and message attachments. To transform the requirements about the parts of a message that have be secured into the model, a corresponding *Data Transfer Object* has to be added to the list of *Data Transfer Objects* of the *Binding-Confidentiality Constraint* respectively the *Binding-Integrity Constraint* for each part that has to be encrypted or signed.

Supporting Tokens represent requirements for security tokens that can be used to encrypt and sign parts of a message in addition to the security tokens described in the binding. Therefore, these security tokens are also transformed into *Credentials*. A *Credential* created from a signed supporting token is added to the *Data Transfer Object* list of the *Binding-Integrity Constraint* and a *Credential* created from an encrypted supporting token is added to the list of the *Binding-Confidentiality Constraint*. For an endorsing supporting token *Integrity Constrains* are created whose *Data Transfer Object* list references the signature created with the binding credentials.

Figure 9 shows the transformation of a *SupportingToken* into the model. The *SupportingToken* assertion contains only one *Security Token Assertion*, namely a *SamlToken* assertion. This assertion states that the re-



```
<sp:SupportingTokens>
①<wsp:Policy>
 ②<sp:SamlToken>
  ③<sp:IssuerName>
      sts.trusted-bank.com
    </sp:IssuerName>
  ④<wst:Claims Dialect="...">
      <ic:ClaimType>sts.trusted-bank.com/
          claims/creditCardNumber</
          ic:ClaimType>
      <ic:ClaimType>sts.trusted-bank.com/
          claims/creditCardOwner</
          ic:ClaimType>
      <ic:ClaimType>sts.trusted-bank.com/
          claims/creditCardExpireDate</
          ic:ClaimType>
    </wst:Claims>
  </sp:SamlToken>
 </wsp:Policy>
</sp:SupportingTokens>
```

(Client Authentication Constraint) ①
(Credential) ②
(Credential Type) SAML ②
(Issuer) sts.trusted-bank.com ③
(Claims) ④
(Claim) sts.trusted-bank.com/claims/creditCardNumber
(Claim) sts.trusted-bank.com/claims/creditCardOwner
(Claim) sts.trusted-bank.com/claims/creditCardExpireDate

Figure 9. Transformation of a *SupportingToken* into the model.

quired *SamlToken* has to be issued by the STS named `sts.trusted-bank.com` and that it contains the credit card information of the user. The *Security Token Assertion* is transformed into *Credentials* containing the corresponding information of the token and this *Credential* is attached to a *Client Authentication Constraint*. Since the *SupportingToken* is neither encrypted nor signed, the created *Client Authentication Constraint* can be simply added to the *Policy Alternatives* list of *Security Constraints* and no further steps are required.

B. *Transforming a Policy Meta-Model Instance to a WS-SecurityPolicy*

The transformation of a policy meta-model instance into a WS-SecurityPolicy can be performed for each *Policy Alternative* independently and can be divided into three phases, as shown in Figure 10. In the first phase, a *Binding* is created, in the second phase the *Protection Assertions* are created, and in the third phase the *Supporting Tokens* are created.

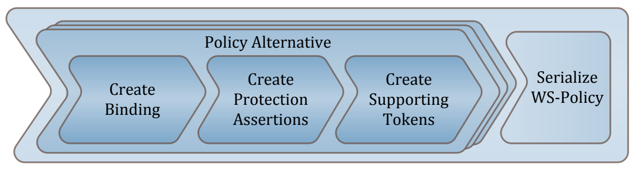


Figure 10. Transformation of the model into a WS-SecurityPolicy.

The first *Confidentiality Constraint* and the first *Integrity Constraint* of a *Policy Alternative* are evaluated to create either a *Transport Binding* assertion or a *Symmetric Binding* assertion or *Asymmetric Binding* assertion. If the *Security Protocol* indicates the use of SSL a *Transport*

Binding assertion is created, otherwise an *Asymmetric Binding* assertion or a *Symmetric Binding* assertion is created, depending on whether the constraints contain a secondary *Credential*, or not.

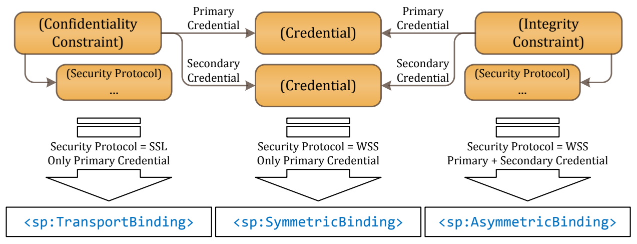


Figure 11. Creation of the binding depending on the properties of the *Binding-Confidentiality Constraint* and the *Binding-Integrity Constraint*.

The first *Confidentiality Constraint* and the first *Integrity Constraint* have to be used in the other transformation phases and are referred to as *Binding-Confidentiality Constraint* and *Binding-Integrity Constraint*. The *Credentials* in the constraints are transformed into *Security Token Assertions* corresponding to the *Credential Type*. The *Security Token Assertions* for the binding assertion of the WS-SecurityPolicy are created depending on the equality of the primary and the secondary *Credentials* of the binding constraints (see Table I).

Table I
Security Token Assertion USAGE DEPENDING ON THE TYPE OF THE *Binding* AND THE *Credentials* OF THE BINDING CONSTRAINTS.

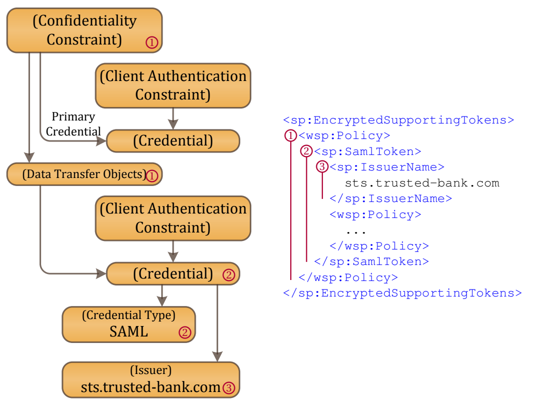
	Same *Primary Credential*	Different *Primary Credential*
Asymmetric Binding	InitiatorToken	InitiatorEncryptionToken InitiatorSignatureToken
Symmetric Binding	ProtectionToken	EncryptionToken SignatureToken
Transport Binding	TransportToken	—

	Same *Secondary Credential*	Different *Secondary Credential*
Asymmetric Binding	RecipientToken	RecipientEncryptionToken RecipientSignatureToken
Symmetric Binding	—	—
Transport Binding	—	—

Based on the *Data Transfer Objects* of the *Binding-Integrity Constraint* and the *Binding-Confidentiality Constraint* the *Protection Assertions* (e.g. encrypted Parts, signed Elements) are created.

Supporting Tokens are created from *Credentials* that are required to be present in a message. Therefore, all *Required Information Constraints* and *User Authentication Constraints* containing a *Credential* as *Data Transfer Object* are evaluated. If these Credentials are not used as primary or secondary *Credential* of the *Binding-Integrity Constraint* or the *Binding-Confidentiality Constraint* a new *Security Token Assertion* is create and added to a *Supporting Token Assertion*. In addition, if the *Credential* is also present in the *Data Transfer Object* list of a binding *Security Constraint* it is marked as *signed* (*Integrity Constraint*) or as *encrypted* (*Confidentiality Constraint*).

An example for the creation of a *Supporting Token* is shown in Figure 12. The example contains a *Confidentiality Constraint*, which is a binding constraint. The primary *Credential* of that constraint is also attached to a *Client Authentication Constraint*, as usual. A second *Client Authentication Constraint* contains the *Credential* that is transformed into a *Supporting Token*. The resulting *Supporting Token* also has to be encrypted, since this second *Credential* is attached to the list of *Data Transfer Objects* of the *Binding-Confidentiality Constraint*, too. Therefore, the *X509Token* assertion created from the second *Credential* is added to the created policy alternative inside an *EncryptedSupportingToken* assertion, as shown on the right of Figure 12.



(Confidentiality Constraint) 1
(Client Authentication Constraint)
Primary Credential
(Credential)
(Data Transfer Objects) 1
(Client Authentication Constraint)
(Credential) 2
(Credential Type) SAML 2
(Issuer) sts.trusted-bank.com 3

```
<sp:EncryptedSupportingTokens>
①<wsp:Policy>
  ②<sp:SamlToken>
    ③<sp:IssuerName>
        sts.trusted-bank.com
      </sp:IssuerName>
      <wsp:Policy>
        ...
      </wsp:Policy>
    </sp:SamlToken>
  </wsp:Policy>
</sp:EncryptedSupportingTokens>
```

Figure 12. Example for the creation of an *EncryptedSupportingTokens* assertion.

V. Policy Meta-Model Aggregation

One main benefit of the policy meta-model is the possibility to easily aggregate multiple policy models into one policy model. The algorithm to enable this posibility is described in this section. The policy meta-model uses *Security Constraints* to express security requirements that correspond to the *Security Goals*. These *Security Constraints* are attached to the *Policy Alternatives* of a policy model *Security Policy*. To express additional security requirements in an alternative, only the corresponding *Security Constraints* have to be added to that alternative. This advantage can be utilised in the process of the aggregation of two *Security Policies*.

For the creation of an aggregated security policy of a service composition the initial security policy of the service composition is extended with the security requirements stated in the security policies of the invoked services. Thereby, the service composition policy can be extended stepwise with the policy of one orchestrated service at a time.

The aggregation of two security policies equals the aggregation of each alternative from the first policy with each alternative from the second policy. This covers all possible combinations of alternatives of the two policies that might be chosen by a service consumer. An algorithm for this aggregation procedure is shown in Listing 1. The resulting alternatives of the aggregation are attached to a new policy that represents the aggregated policy.

```
result = new Policy

for policy1.allAlternatives() as
    alternative1 {
    for policy2.allAlternatives() as
        alternative2 {
        mergedAlternative =
            mergeAlternatives(
            alternative1 , alternative2)
        result.add(mergedAlternative)
    }
}
```

Listing 1. Aggregation algorithm for two *Security Policies* of the policy model

The aggregation of two alternatives is a new alternative that contains all *Security Constraints* of these two alternatives. Therefore, the *Security Constraints* of the first alternative and subsequently the *Security Constraints* of the second alternative are added to the new alternative.

Figure 13 illustrates an example of the aggregation of two security policies. The first of the two policy contains only one alternative, while the second policy contains two alternatives. Therefore, the alternative of the first policy is combined with each alternative of the second policy. As a result, the aggregated policy has two alternatives. Thereby, the first resulting alternative contains the *Security Constraints* of the alternative of the first policy and the *Security Constraints* of the first alternative of the second policy. The second resulting alternative contains the *Security Constraints* of the alternative of the first policy and the *Security Constraints* of the second alternative of the second policy.

VI. Related Work

Modelling security requirements in Service-oriented Architectures is an emerging topic.

Satoh and Tokuda [9] present a mechanism of creating a security policy of a composite service automatically based on predicate logic. Their mechanism "makes it possible to validate the consistency of policies by inference." The work of Satoh and Tokuda goes in the same direction as this paper. However, using the model defined in the scope of this paper to represent the semantics of security policies allows more possibilities, such as semantic validation of security policies and automated creation of security policies.

Satoh and Yamaguchi [10] introduce an intermediate model to transform WS-SecurityPolicy instances into platform-specific configuration files for WS-Security based Model Driven Security (MDS). This model is defined to represent the WS-Security message structure and the meanings of signatures and encryption specified in a WS-SecurityPolicy instance. Although this basic idea is

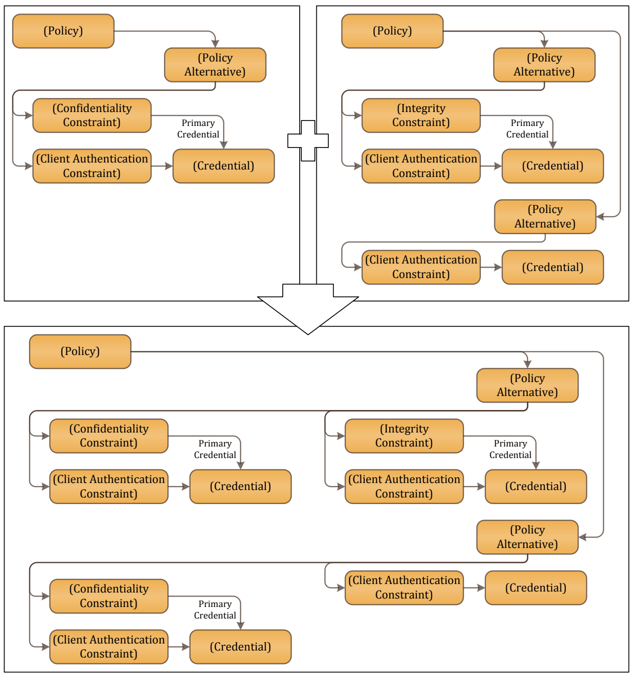


Figure 13. Example for the aggregation of two security policies expressed as policy meta-model.

similar to the approach taken in the scope of this paper, their target is a different one.

Phan et al. are comparing several policy frameworks in their "survey of policy-based management approaches for Service Oriented Systems" [11]. They summarize WS-policy as "being widely accepted and supported by major SOA vendors" and "a good choice for low-level policy representation for Web Services-based implementations of SOA." Further they state that it "would be useful if a policy refinement mechanism can be developed for the translation of high-level business oriented policies [. . .] into low-level WS-Policy statements for runtime execution." Using our policy meta-model this objective can be achieved.

Chang et al. present "a solution for managing security policies in a collaborative Web Services environment." [12] This solution allows to establish security policies dynamically for individual interoperation based on a global policy registry. It also "addresses the software, message, and policy versioning and interoperability issues."

Bartoletti et al. describe networks using a λ-calculus and present a static approach "that determines how to compose services while guaranteeing that their execution is always secure, without resorting to any dynamic check." [13] Therefore, a framework is provided to enable a client to secure its sensitive information in a way that prevents a service of unintentional access.

Charfi and Mezini present an aspect-oriented programming (AOP) extension to BPEL. This extension is used in a framework to secure Web Services and service compositions. They also "introduce the notion of policy-based process deployment to check the compatibility of the security policies of the composition and its partners at deployment time." [14] However, they provide no automated process for the aggregation of security policies and the policy of the composed service has to be defined manually.

VII. Conclusion and Future Work

Specifications for security policies in SOA such as WS-Policy and WS-SecurityPolicy provide a language to enable a declarative configuration of security requirements. These languages enable interoperability at run-time, since service consumers can secure service calls in accordance to a service's policy. In the scope of service composition, security policies might depend on the policies of orchestrated services. So far, there is no tool support available that supports the generation of security policies for composed Web Services.

In this paper, we presented an approach to generate Web Service security policies using an abstraction layer. Our policy model is designed to simplify the handling and generation of Web Service security policies. A mapping to WS-SecurityPolicy is presented to import and export Web Service security policies. However, our model enables the generation or conversion of other Web Service Security languages (e.g. Axis 2 security configuration language) as well. Finally, we presented an algorithm that is capable to merge security policies expressed in our model. Due to the design of our model this algorithm can be hold very simple.

To extend our work, we will provide a validation of aggregated policies in the next step. For example, aggregated policies may contain security requirements that contradict each other. Therefore, it is necessary to develop strategies to deal with such situations.

References

[1] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," Specification, April 2007. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[2] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0," Specification, June 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[3] R. Kanneganti and P. Chodavarapu, *Soa Security in Action*. Greenwich, CT, USA: Manning Publications Co., 2007.

[4] *Web Services Policy 1.5 - Framework*, W3C Std., 4 th Sep. 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-ws-policy-20070904

[5] *Web Services Policy 1.5 - Attachment*, W3C Std., 4 th Sep. 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904

[6] *WS-SecurityPolicy 1.3*, OASIS Std., 29 th Nov. 2008. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/cd/ws-securitypolicy-1.3-spec-cs-01.html

[7] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1," OASIS Standard Specification, February 2006. [Online]. Available: "http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf"

[8] *WS-Trust 1.3*, OASIS Std., 19 th Mar. 2007. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf

[9] F. Satoh and T. Tokuda, "Security Policy Composition for Composite Services," in *ICWE 2008 International Conference on Web Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 86–97.

[10] F. Satoh and Y. Yamaguchi, "Generic Security Policy Transformation Framework for WS-Security," in *IEEE International Conference on Web Services (ICWS 2007)*. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 513–520.

[11] T. Phan, J. Han, J.-G. Schneider, T. Ebringer, and T. Rogers, "A survey of policy-based management approaches for Service Oriented Systems," in *Proceedings of the 19th Australian Conference on Software Engineering (ASWEC '08)*, 2008, pp. 392–401.

[12] S. Chang, Q. Chen, and M. Hsu, "Managing Security Policy in a Large Distributed Web Services Environment," in *COMPSAC '03: Proceedings of the 27th Annual International Conference on Computer Software and Applications*. Washington, DC, USA: IEEE Computer Society, 2003, p. 610.

[13] M. Bartoletti, P. Degano, and G. L. Ferrari, "Security Issues in Service Composition," in *In Proceedings of FMOODS 2006, 8th IFIP Internat. Conf. on Formal Methods for Open Object-Based Distributed Systems, volume 4037 of LNCS*. Springer, 2006, pp. 1–16.

[14] A. Charfi and M. Mezini, "Using Aspects for Security Engineering of Web Service Compositions," in *ICWS '05: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 59–66.