XML and Web Services Security Standards

Nils Agne Nordbotten, Norwegian Defence Research Establishment

***Abstract*—XML and Web services are widely used in current distributed systems. The security of the XML based communication, and the Web services themselves, is of great importance to the overall security of these systems. Furthermore, in order to facilitate interoperability, the security mechanisms should preferably be based on established standards. In this paper we provide a tutorial on current security standards for XML and Web services. The discussed standards include XML Signature, XML Encryption, the XML Key Management Specification (XKMS), WS-Security, WS-Trust, WS-SecureConversation, Web Services Policy, WS-SecurityPolicy, the eXtensible Access Control Markup Language (XACML), and the Security Assertion Markup Language (SAML).**

I. Introduction

A Web service is defined as a software system designed to support interoperable machine-to-machine interaction over a network [1]. Put in another way, Web services provide a framework for system integration, independent of programming language and operating system. Web services are widely deployed in current distributed systems and have become the technology of choice for implementing service-oriented architectures (SOA). In such architectures, loosely coupled services may be located across organizational domains.

The suitability of Web services for integrating heterogeneous systems is largely facilitated through its extensive use of the Extensible Markup Language (XML). The interface of a Web service is for instance described using the XML based Web Services Description Language (WSDL). Furthermore, communication is performed using XML based SOAP messages.[1]

Thus, the security of a Web services based system depends not only on the security of the services themselves, but also on the confidentiality and integrity of the XML based SOAP messages used for communication.

The Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C) have over the last years standardized several specifications related to security in Web services and XML. This paper provides an overview of these security standards. Using these established standards when creating Web services, instead of custom solutions, clearly has the advantages of facilitating both system interoperability and reusability.

The rest of this paper is organized as follows: we start by providing a brief overview of XML and Web services security in the next section. Afterwards, more detailed discussions of each of the security standards are provided in separate sections.

[1]SOAP was originally an acronym for Simple Object Access Protocol, however, as of SOAP 1.2 it is no longer an acronym. Readers who are unfamiliar with SOAP and WSDL may refer to the introduction provided by Curbera et al. [2].

II. An Overview of XML and Web Services Security

XML based SOAP messages form the basis for exchanging information between entities in Web services systems. The information contained within these SOAP messages may be subject to both confidentiality and integrity requirements. Although mechanisms at lower layers may provide end-to-end security, these lower layer mechanisms are often insufficient. This is due to the fact that a SOAP message may be subject to processing and even modification (e.g., removal/insertion of a SOAP header) at intermediary nodes. The result being that the end-to-end security provided by lower layer mechanisms (e.g., SSL/TLS) is broken, as illustrated in Figure 1. Relying on lower layers for end-to-end security may also cause problems if a message is to pass through various networks utilizing different transport protocols. Furthermore, security at the XML level has the advantage of enabling confidentiality and source integrity to be maintained also during storage at the receiving node(s).

XML Signature and XML Encryption are used to provide integrity and confidentiality respectively. Although these two standards are based on digital signatures and encryption, none of them define any new cryptographic algorithms. Instead, XML Signature and XML Encryption define how to apply well established digital signature/encryption algorithms to XML. This includes:

- A standardized way to represent signatures, encrypted data, and information about the associated key(s) in XML, independent of whether the signed/encrypted resource is an XML resource or not.
- The possibility to sign and/or encrypt selected parts of an XML document.
- The means to transform two logically equivalent XML documents, but with syntactic differences, into the same physical representation. This is referred to as canonicalization. In order to be able to verify the signature of an XML resource that has had its representation changed, but still has the same logical meaning, it is essential that canonicalization is performed as part of the XML signature creation and verification processes.

As both XML Signature and XML Encryption rely on the use of cryptographic keys, key management is a prerequisite for their effective use on a larger scale. Therefore, the XML Key Management Specification (XKMS) was created to be suitable for use in combination with XML Signature and XML Encryption. XKMS basically defines simple Web services interfaces for key management, thereby hiding the complexity of traditional public key infrastructures (PKIs) from the clients. XML Signature, XML Encryption, and XKMS are all discussed in more detail in Section III.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

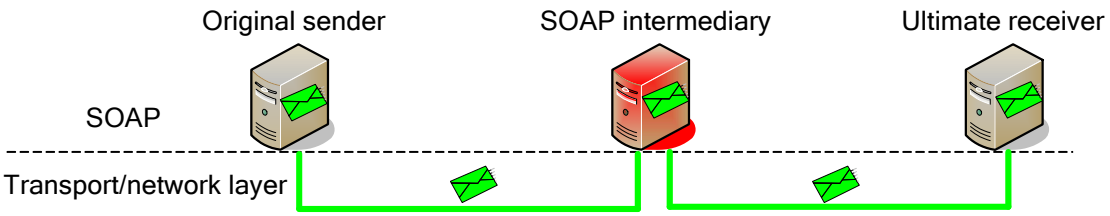

Fig. 1. The transport/network layer security (e.g., SSL/TLS or IPSec) is broken at the intermediary SOAP node. By applying security at the SOAP/XML level, on the other hand, end-to-end security can provided.

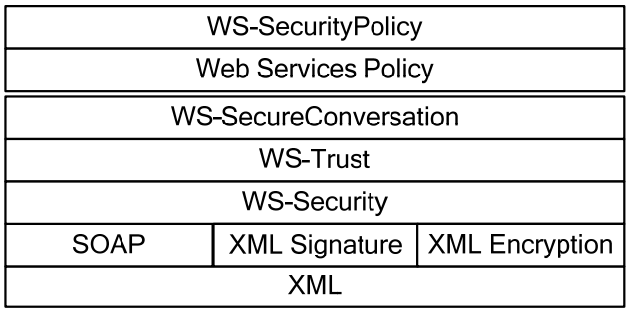

Fig. 2. The conceptual relationship between the XML and Web services security standards. Be aware that Web Services Policy can also be used (independently) for other purposes than security. Also recall that XKMS may provide key management for use with XML Encryption and XML Signature, although this is not shown in the figure.

As noted previously, SOAP is an XML based messaging format. Thus, XML Signature and XML Encryption are obvious candidates for being reused to provide SOAP security as well. As illustrated in Figure 2, this is exactly what WS-Security does. WS-Security specifies how to apply XML Signature and XML Encryption to SOAP messages, effectively providing integrity and confidentiality to SOAP messages (or parts of SOAP messages). As multiple encryptions can be used within the same SOAP message, the different parts of a SOAP message may be encrypted for different receivers (SOAP intermediaries). Likewise, a SOAP intermediary may add an additional signature to a SOAP message, thereby providing integrity protection for a newly added header or supporting separation-of-duty through co-signatures.

In addition to providing confidentiality and integrity for SOAP messages, WS-Security also provides a mechanism to avoid replay attacks (i.e., timestamps) and a way to include security tokens in SOAP messages. Security tokens are typically used to provide authentication and authorization.

WS-Security has no notion of a communication session, that is, it is only concerned with securing a single SOAP message or a single SOAP request/response exchange. In cases where multiple message exchanges are expected, WS-SecureConversation may be used to establish and maintain an authenticated context. The authenticated context is represented by a URI in a context token and consists of a shared secret that can be used for key derivation. WS-SecureConversation relies on WS-Trust to establish the security context.

WS-Trust basically defines a framework for obtaining security tokens (including the context tokens used in WS-SecureConversation) and brokering of trust. WS-Security, WS-Trust, and WS-SecureConversation are all discussed in more detail in Section IV.

With a range of Web services standards, interoperability becomes very difficult unless the communicating parties know what standards to use and how these standards are to be used. Web Services Policy provides the means by which service providers and clients can specify their interoperability requirements and capabilities. WS-SecurityPolicy can be viewed as an extension to Web Services Policy, defining how Web Services Policy can be used to specify requirements and capabilities regarding the use of WS-Security, WS-Trust, and WS-SecureConversation. For instance, a service provider may specify using WS-Policy/WS-SecurityPolicy that it requires certain message parts to be encrypted. WS-Policy and WS-SecurityPolicy are also further discussed in Section IV.

The last two standards covered in this paper are the Security Assertion Markup Language (SAML) and the eXtensible Access Control Markup Language (XACML). SAML may be used to communicate authentication, attribute, and authorization information in a trusted way. SAML is based on XML and although its original motivation was single sign-on for Web browsing, it is also well suited for use in Web services. XACML on the other hand is used to define access control policies in XML, and may be used to define access control policies for any type of resource. Because SAML and XACML are not targeted exclusively at Web services, SAML and XACML were not included in Figure 2. However, this does not imply that there is no interaction between these standards. A XACML implementation may for instance rely on the security tokens of WS-Security for authentication. As to security tokens, there is also a SAML based security token in WS-Security. SAML and XACML are both further discussed in Section V.

III. XML Security

XML Signature and XML Encryption are fundamental to XML and Web services security. Because of this, XML Signature and XML Encryption are both well supported in available products and by development tools. The next two sections describe XML Signature and XML Encryption respectively. Then, in Section III-C, the XML Key Management Specification (XKMS) is presented. XKMS facilitates the use of XML Signatures and XML Encryption by simplifying key management. All three of these specifications are standardized

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

by W3C [3][4][5].[2]

A. XML Signature

The use of digital signatures is a common method for ensuring message integrity, authentication, and non-repudiation. XML Signature [3] defines a standard interoperable format for representing digital signatures in XML and provides mechanisms for efficiently applying digital signatures to XML resources. XML Signature is not limited to signing XML resources, however, as it can also be used to sign binary resources such as a JPEG-file.

A single XML signature may cover several resources, where each resource may be an XML document, a part of an XML document, or a binary resource. The `Signature` element for representing digital signatures in XML is shown in Figure 3. The `SignedInfo` element is used to specify what is being signed. A `Reference` element within `SignedInfo` is associated with each resource, identifying the resource through a URI.[3] The `Reference` element also includes a digest of the referenced resource. The digest is created by first applying any applicable transforms and then calculating the digest value from the result.

Canonicalization is one possible transform that may be applied to an XML resource before calculating the digest. The need for XML canonicalization is due to the fact that two logically equivalent XML resources may differ in physical representation. Such variations in physical representation may for instance be due to the use of different character encodings or insignificant structural differences. Canonicalization methods define a normal form, that is, the canonical form, into which logically equivalent documents can be converted to obtain the same physical representation. There are two main canonicalization methods, Canonical XML [7][8] and Exclusive XML Canonicalization [9]. [4]

Apart from canonicalization, several other transformations may be applied to a resource. These include Base64 decoding, XPath filtering, and Extensible Stylesheet Language Transformations (XSLT). Furthermore, there is a decryption transform [11] that enables XML Signature applications to distinguish between XML structures that were encrypted before the signature was calculated and structures that were encrypted after the signature was calculated. Later in this section we will also discuss another transform, that is, the enveloped signature transform. Independent of which transformations are applied to a resource, each applied transformation is identified by a `Transform` element within the `Reference` element.

The last element within the `Reference` element is the `DigestMethod` element, used for specifying the digest algorithm being used. The only digest algorithm required to be supported is SHA-1. However, there have also been defined identifiers for additional algorithms [12][4] and several implementations support SHA-2 (i.e., SHA-224, SHA-256, SHA-384, and/or SHA-512).

In addition to containing one or more `Reference` elements, the `SignedInfo` element also specifies the signature method used (`SignatureMethod`) and the canonicalization method for canonicalizing the `SignedInfo` element itself (`CanonicalizationMethod`). Because the `SignedInfo` element is what is actually signed in XML Signature, it is required that this element is canonicalized before calculating the signature value. Notice that because the `SignedInfo` element contains the digests of all the resources to be signed, these resources are implicitly signed as well when signing the `SignedInfo` element. Signature verification therefore consists of two steps. The first is to make sure that the `SignedInfo` element has not changed by verifying the signature value stored in the `SignatureValue` element. The second is to make sure that none of the referenced resources have changed, by verifying the digest of each resource.

The specification only requires one signature algorithm to be supported, that is, DSA with SHA-1 (also known as DSS). Furthermore, support for message authentication codes based on secret/shared keys (i.e., HMAC-SHA1) is also mandatory. It is also recommended to support RSA with SHA-1 and many implementations also support some of the additional defined algorithms [12], such as RSA with SHA-512 or the Elliptic Curve Signature Algorithm (ECDSA). Although commonly used algorithms are likely to be supported by most vendors, differences between products with regard to what algorithms are supported may cause interoperability problems.

As mentioned previously, the `SignedInfo` element contains references to the resources being signed. In this regard, an XML Signature may be enveloping, enveloped, or detached with respect to each referenced resource. This is illustrated in Figure 4. An enveloped signature means that the `Signature` element is inside the referenced XML resource. A detached signature on the other hand references a resource that is separate from the `Signature` element. Finally, an enveloping signature references a resource that is contained within the `Signature` element. In the latter case, an instance of the `Object` element is used to contain the resource. Because a single signature can reference/sign multiple resources, a signature may be enveloped, detached, and enveloping at the same time. Furthermore, multiple independent signatures may coexist within the same XML document.

Because the `Signature` element of an enveloped signature is actually located within the XML document being signed, an enveloped signature transform is defined. This transform removes the entire `Signature` element from the digest calculation, so that the signature element is not included in the digest of the XML resource being signed. Otherwise it would not be possible to calculate the correct digest, considering that the resource (from which the digest is to be calculated) would be subject to change when adding the digest to the `Signature` element.

XML Signature also defines a `KeyInfo` element, as shown in Figure 3, that may be used to provide information about the key to be used for verifying the signature. This information

[2]The original XML Signature specification was also published as an IETF RFC [6].

[3]The URI reference may be omitted, in which case the receiving application is assumed to know the identity of the resource.

[4]The UDDI Specification Technical Committee within OASIS has also released a specification for a schema centric XML canonicalization [10].

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

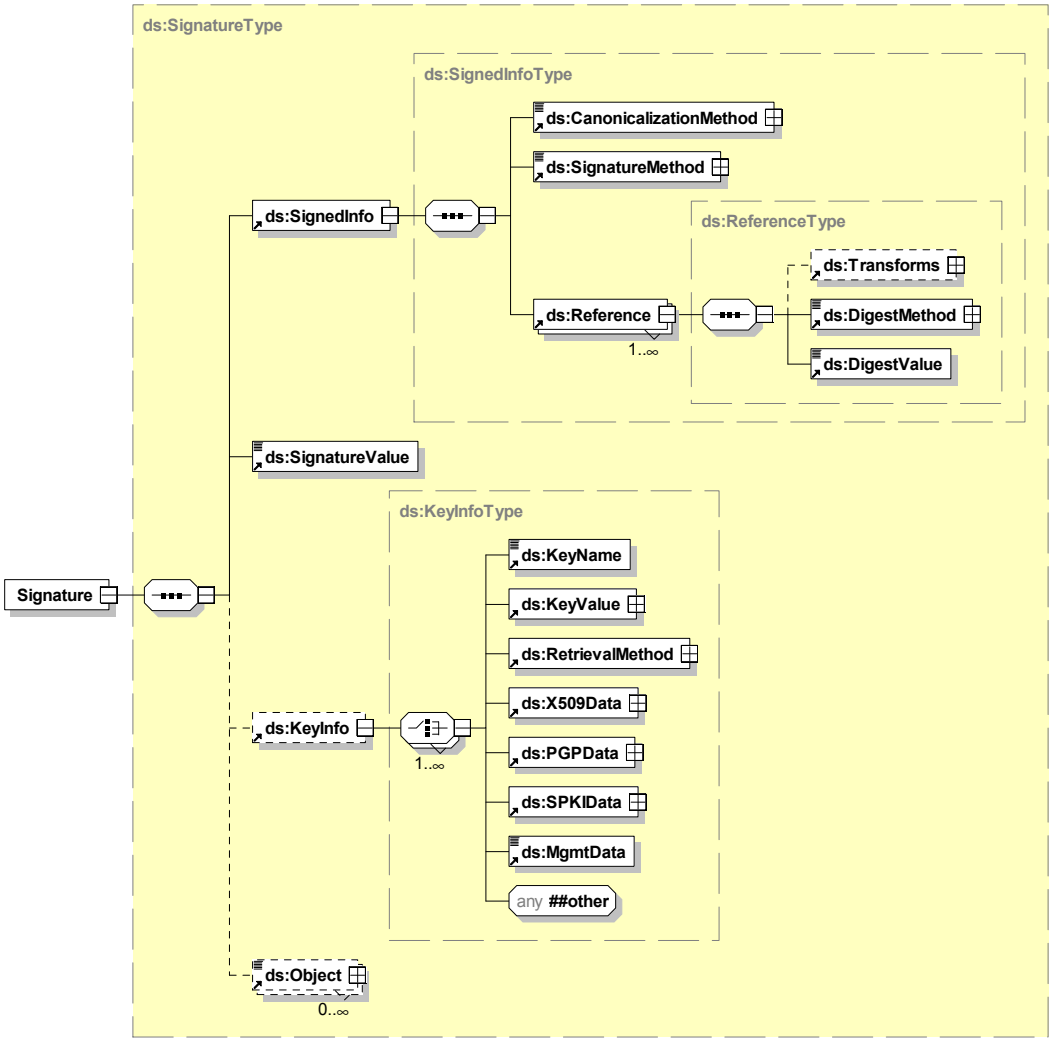

Fig. 3. The Signature element. (XML attributes are not shown.)

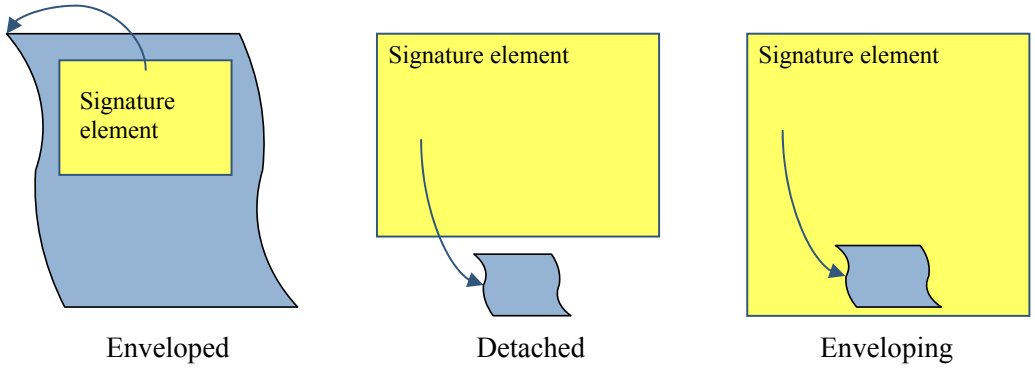

Fig. 4. Enveloped, detached, and enveloping signatures.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

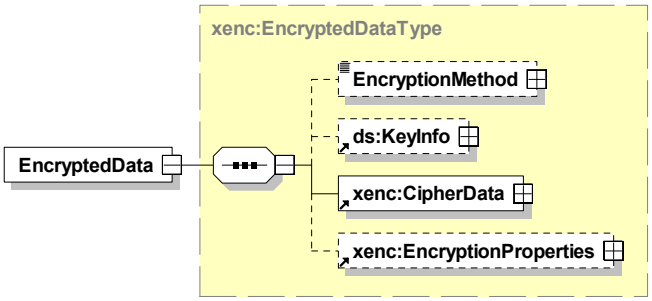

Fig. 5. The EncryptedData element.

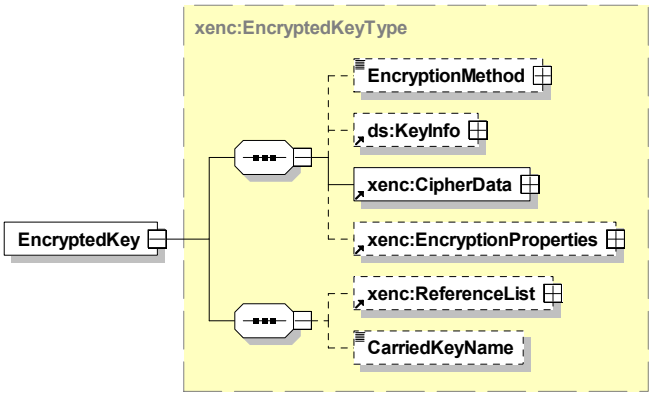

Fig. 6. The EncryptedKey element.

may be provided by identifying the key by name, by including the raw public key itself, and/or by including (or referencing) an X.509 or SPKI certificate corresponding to the key pair being used. Using the `PGPData` element, a PGP key packet can also be included. Furthermore, the `RetrievalMethod` element may be used to reference `KeyInfo` information at another location (i.e., within another `KeyInfo` element).

As we will see in the next section, the `KeyInfo` element defined by XML Signature is also used by XML Encryption. In fact, XML Encryption extends the `KeyInfo` element with an `EncryptedKey` element, which may provide transport for a secret/symmetric key. The `KeyInfo` element is also used by the XML Key Management Specification (to be discussed in Section III-C), facilitating its close integration with XML Signature and XML Encryption.

B. XML Encryption

XML Encryption [4] provides confidentiality by allowing selected parts of, or an entire, XML document to be encrypted. XML Encryption is similar to XML Signature in many ways. For instance, like XML Signature, XML Encryption does not apply only to XML resources as it may be used to encrypt arbitrary binary resources as well.

Data that is encrypted using XML Encryption is represented by an `EncryptedData` element. The `EncryptedData` element is shown in Figure 5. As can be seen, the `CipherData` element is the only mandatory child element of `EncryptedData`. `CipherData` either contains or provides a reference to the ciphertext of the encrypted data. As may be noticed, this is equivalent with the enveloping and detached variations of XML Signature. Contrary to XML Signature, however, a single `EncryptedData` element can only contain or reference one resource. If multiple resources are to be encrypted within the same XML document, multiple `EncryptedData` elements must be used.

When encrypting an XML element, one may choose to encrypt the entire element (including its outmost tags) or only the element's content. In the case where the ciphertext is contained within the `CipherData` (i.e., enveloping), the `EncryptedData` element replaces the XML element (or element content) being encrypted.

The encryption algorithm used may be specified in the `EncryptionMethod` element (or be known by the receiver). The specification requires support for both Triple-DES and AES-128/256. Both are used in cipher block chaining (CBC) mode, with an initialization vector that is prefixed to the ciphertext. Additional information (e.g., specifying the time/date at which the encryption was performed) may be included within the `EncryptionProperties` element.

As mentioned, in Section III-A, XML Encryption defines an `EncryptedKey` element that may be used to provide transport for a secret/symmetric key. As can be seen in Figure 6, the `EncryptedKey` element contains all the child elements of `EncryptedData` plus two additional ones. In fact, `EncryptedKey` and `EncryptedData` are both derived from the same abstract type (i.e., `EncryptedType`). When using the `EncryptedKey` element to provide key transport, it is included as a child element of the `KeyInfo` element of `EncryptedData`. `EncryptedKey's CipherData` is then used to transport the secret key in encrypted form, while the `KeyInfo` element within the `EncryptedKey` element is used to communicate information about the key used for encrypting the secret key. Typically a pre-shared secret key or the public key of the receiver is used for this latter purpose.

In the case that the same key is used to encrypt multiple `EncryptedData` elements, the `ReferenceList` within `EncryptedKey` may be used to identify the `EncryptedData` elements that utilize the key. The `EncryptedKey` can also be given a key name, so that it can be referenced from each respective `EncryptedData` element.

C. The XML Key Management Specification (XKMS)

An appropriate method for key management is essential in order to employ XML Signature and XML Encryption in a scalable manner. The XML Key Management Specification (XKMS) [5] defines simple Web services for retrieving, validating, and registering public keys, thereby shielding clients from the complexity of the potentially underlying public key infrastructure (PKI).

XKMS is divided into two main parts, the XML Key Registration Service Specification (X-KRSS) and the XML Key Information Service Specification (X-KISS). The XML Key Registration Service Specification defines services in order to register, recover, revoke, and reissue keys. In the case of registering a new public key, the key pair generation may

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

either be performed by the client or as part of the offered service. In the case that the key pair is generated by the client, the client is required to prove possession of the private key in order to register the public key. In either case, the XML Key Registration Service Specification provides mechanisms for authenticating clients.

The XML Key Information Service Specification defines two services, namely locate and validate. The locate service enables a client to retrieve a public key, or information about a public key. The data format for communicating key information is provided by the `KeyInfo` element defined by XML Signature (and also used by XML Encryption), thereby facilitating the use of XKMS together with XML Signature and XML Encryption.

A client may for instance receive a signed XML document where the key to be used to verify the signature is identified by some mechanism provided by the `KeyInfo` element (e.g., an included X.509 certificate or a key name). Instead of being required to resolve the key itself, the client may simply include the received `KeyInfo` element within a request to the locate service, which then resolves the required `KeyInfo` elements (e.g., by including the key value) and returns it to the client. The locate service could obtain the resolved information by parsing a certificate included in the `KeyInfo` element, based on a previous registration of the key with an XML key registration service, from an underlying public key infrastructure, or by some other means. In any case, the locate service shields the client from the complexity of having to perform these actions itself. However, the locate service does not validate the returned key information.

This is where the validate service has its role. The validate service provides the same functionality as the locate service, but also assures that the returned information meets specific validation criteria (e.g., by validating the X.509 certificate). In order for such an assurance to be trustworthy, the client is obviously required to have a trust relationship with the validation service. Furthermore, it must be assured that the applied validation criteria are appropriate for the application.

Because validation incurs additional overhead, the locate service is likely to be preferable in scenarios where there is no sufficient trust relationship between the client and the validation service, or where the validation requirements of the application are not fulfilled by the validation service.

Let us consider the scenario in Figure 7, where Alice wants to send an encrypted document to Bob using his public key. However, Alice does not possess the public key of Bob. Furthermore, although Bob has registered his public key with the XKMS service within his own domain, there is no trust relationship between Alice and the XKMS service within Bob's domain. In this case, Alice may contact the validate service within her own domain, specifying that she requires the public key of Bob to be used for encryption (or key exchange). The validate service may then forward this request to the locate service within Bob's domain (which might be located through DNS). Before the response is returned to Alice, it is validated by the validate service within her own domain.

IV. Web Services Security

While the previous section focused on security standards for XML in general, we will now turn our attention to security standards targeted exclusively at Web services. In particular, this section provides an overview of WS-Security, Web Services Policy, WS-SecurityPolicy, WS-Trust, and WS-SecureConversation. All five were originally proposed as part of the Web services security roadmap by IBM and Microsoft [13]. WS-Security, WS-Trust, WS-SecureConversation, and WS-SecurityPolicy have later become standardized within OASIS [14][15][16][17], while Web Services Policy has been standardized within the W3C [18]. Development support for these standards can for instance be found in the Web Services Interoperability Toolkit [19] for Java and in the Windows Communication Foundation [20] for .Net. The standards are also supported by various products, such as XML firewalls [21].

Before we look at each individual standard, let us first briefly repeat the relationships between these standards (as previously illustrated in Figure 2). First of all, WS-Security is concerned with security for SOAP messages, thus, WS-Security clearly builds on top of SOAP. In addition, WS-Security also makes use of XML Signature and XML Encryption. WS-Trust again builds on WS-Security, while at the same time providing functionality that may be utilized by WS-Security. WS-SecureConversation builds on WS-Security and WS-Trust, while at the same time enabling WS-Security to be used in a more efficient way. Finally, WS-SecurityPolicy extends Web Services Policy in order to facilitate the use of WS-Security, WS-SecureConversation, and WS-Trust.

A. WS-Security

The Web Services Security (WSS) specifications aim to provide a framework for building secure Web services using SOAP, and consist of a core specification and several additional profiles. The core specification, the Web Services Security: SOAP Message Security specification [14] (WS-Security for short), defines a security header for use within SOAP messages and defines how this security header can be used to provide confidentiality and integrity to SOAP messages. XML Encryption is utilized to provide confidentiality, while message integrity is provided through the use of XML Signature. Using these mechanisms, SOAP message body elements, selected headers, or any combination thereof may be signed and/or encrypted; potentially using different signatures and encryptions for different SOAP roles (i.e., different intermediaries and ultimate receiver(s)).

Recall (from Section II) that because SOAP message headers may be subject to processing and modification by SOAP intermediaries, lower layer security mechanisms such as SSL/TLS are often insufficient to ensure end-to-end integrity and confidentiality for SOAP messages. For such messages, the functionality provided by WS-Security is essential if confidentiality and/or integrity are required.

In order to ensure that the response received by an initiator has been generated in response to the original request in its unaltered form, WS-Security defines a signature confirmation

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

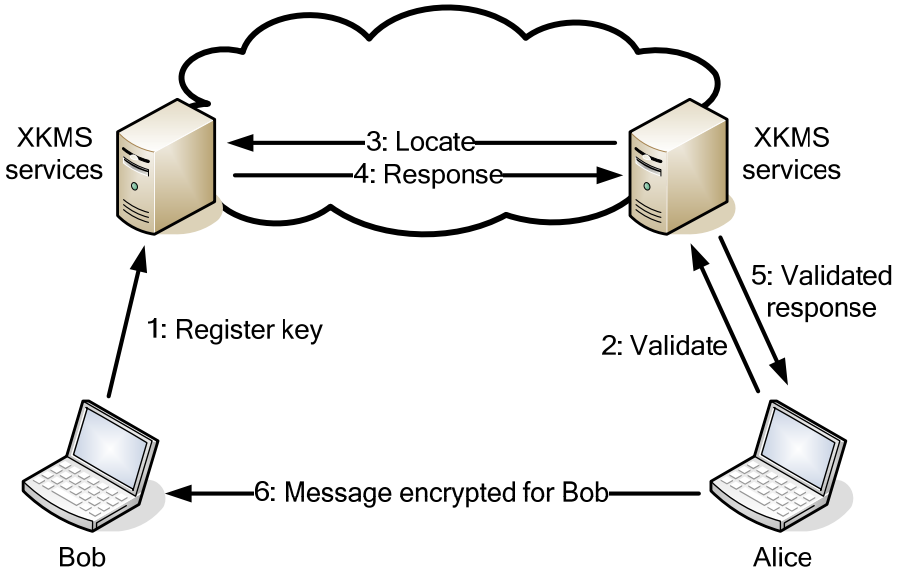

Fig. 7. Alice utilizes XKMS in order to obtain a validated public key for Bob.

attribute to be used for including a copy of the digital signature value of the request message. By including this attribute in the digital signature of the response message, as a signed receipt, the response message is tied to the original request.

The specification also defines a timestamp element that may aid in preventing replay attacks. This element specifies the creation time of the message and optionally an expiration time. As no clock synchronization is provided, it is suggested that recipients take clock skew into consideration when evaluating the freshness of a message, unless clock synchronization is performed out-of-band. It is recommended that the timestamps are cached for a minimum of five minutes (or, if present, until the expiration time) to detect replay of previous messages. If there is a risk that the message could potentially be replayed to another receiver, the recipient should be uniquely identified and bound to the timestamp by means such as a digital signature.

In order to provide extensibility, WS-Security also provides a mechanism to include security tokens within SOAP messages. Security tokens contain a set of claims, and may be in binary or XML representation. An authority may assert the claims contained within a security token by signing the security token. Currently five token types are defined in separate profiles. These are the X.509 certificate token profile [22], the Rights Expression Language (REL) token profile [23], the Kerberos token profile [24], the UsernameToken profile [25], and the SAML token profile [26]. There is also a WSS: SOAP Messages with Attachments (SwA) Profile [27], which is applicable to SOAP 1.1 but not to SOAP 1.2.

1) The UsernameToken Profile: The UsernameToken profile [25] specifies how the `UsernameToken` can be used as a means to identify a requester by username. A password, or some sort of shared secret constituting a password equivalent, may also be included. Passwords may be included in their original form or as a SHA-1 digest. In order to prevent replay attacks, it is also recommended that a nonce (i.e., a random value created by the sender) and a timestamp are included. By combining nonces with timestamps, nonces are not required to be cached beyond their validity period. The SHA-1 password digest is to be calculated over the nonce, timestamp, and password, thus, both the sender and the receiver need to know the plaintext password or password equivalent. Notice though, that if the password equivalent is the digest of the password, the receiver is not required to store the plaintext password.

The UsernameToken profile also defines a way to derive a shared cryptographic key from the password associated with a given username. Key derivation is achieved by specifying a salt (i.e., a random value) and a number of iterations. By hashing the password and salt, and iterating the number of times specified on the result, the shared key can be obtained. This way, only those who know the password are able to derive the shared key (given the salt and the number of iterations). The maximum size of a derived key is 160 bits, although the entropy of keys generated from typical passwords in this way is likely to be much lower. For instance, there are only $(26+26+10)^8$, or about 2^{48}, different eight character passwords consisting of lower- and uppercase letters as well as numbers. Thus, a key derived from such a password would be comparable in strength with a 48-bit key, which is not secure for general use. Additionally, such a scheme may be vulnerable to dictionary attacks unless the passwords have been randomly generated.

The specification does not provide measures to prevent a `UsernameToken` from being replayed to a different receiver. Thus, if the same usernames/passwords are valid with multiple receivers, measures against such replay attacks must be provided by implementers. One potential solution is to require the identity of the receiver to be included in the password digest. Nevertheless, such custom solutions may be susceptible to cause interoperability problems.

2) The X.509 Certificate Token Profile: The X.509 certificate token profile [22] defines how to include X.509 certifi-

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

cates in SOAP messages. Such certificate tokens may be used to validate the public key used for authenticating the message or to specify the public key, which was used to encrypt the message (or more commonly to convey the secret key used to encrypt the message). When the X.509 certificate is used to authenticate the sender, ownership of the certificate token is proved by signing the message using the corresponding private key.

3) The Rights Expression Language (REL) Token Profile: The Rights Expression Language (REL) token profile [23] defines how to include ISO/IEX 21000-5 Rights Expressions in SOAP messages. In the context of XML and Web services, the Rights Expression Language is also known as the XML Rights Management Language (XrML). Although a technical committee was formed within OASIS in order to standardize XrML, this committee was disbanded before reaching an agreement on a standard.

Anyway, in REL/XrML, rights are expressed in the form of licenses. A license grants a key holder some rights and is signed by the issuer. Licenses may for instance be used to convey attributes of the key holder or to provide authorization to perform certain actions (e.g., issuing specific types of licenses to others). Considering that SAML is more widely supported by Web services implementations, and can be used to achieve much of the same things, one may want to consider using SAML instead.

4) The SAML Token Profile: The SAML token profile [26] defines how to include SAML assertions within security headers and how to reference these assertions from within the SOAP message. A binding between a SAML token and the SOAP message (and its sender) can be created by signing the message with a key specified within the SAML assertion. Alternatively, an attesting entity that the receiver trusts may vouch for the message being sent on behalf of the subject for whom the assertion statements apply. In this latter case, the attesting entity must ensure the integrity of the vouched for SOAP message (e.g., by applying a digital signature). SAML is discussed in more detail in Section V-B.

5) The Kerberos Token Profile: The Kerberos token profile [24] defines how to attach Kerberos tickets to SOAP messages. The specification is limited to the Kerberos AP-REQ message [28], allowing a client to authenticate to a service. Like with the X.509 certificate token, ownership of the token is proved by signing the message using the corresponding key. How the AP-REQ is to be obtained is outside the scope of the profile, but such functionality is provided by the Kerberos specification and might also be provided using WS-Trust.

6) The Basic Security Profile: The Web Services Interoperability Organization (WS-I) has also defined another related profile called the Basic Security Profile [29]. This profile provides clarifications, and requirements, on how WS-Security and its associated profiles should be implemented in order to promote interoperability. Because WS-Security makes use of XML Signature and XML Encryption, the Basic Security Profile also applies to XML Signature and XML Encryption when these are used with WS-Security. Because the Basic Security Profile is mostly about implementation details, however, it is not discussed in further detail in this paper.

B. Web Services Policy

The Web Services Policy framework [18] provides for expressing policies in Web services-based systems. Using such policies, interoperability requirements and capabilities can be expressed by both Web service consumers and providers. Before looking at Web Services Policy in more detail, let us first consider a usage scenario. A service provider may for instance require that all messages are encrypted using XML Encryption, using AES-128 (i.e., AES encryption with 128-bit keys) or AES-256. Likewise, a potential service consumer might support XML Encryption using TripleDES or AES-128. Clearly, in order to be interoperable, both parties should make use of XML Encryption with AES-128. Still, the consumer and provider need to be able to make this decision themselves in an automatic way. This is where Web Services Policy has its role, enabling the capabilities and requirements of both service consumers and providers to be expressed through policies.

At the top level, a policy consists of a collection of policy alternatives. Each policy alternative again contains policy assertions corresponding to specific requirements and capabilities associated with that policy alternative. For instance, a Web service consumer may choose to use any single one of the policy alternatives supported by the provider. However, once a policy alternative has been chosen, both parties are required to fulfill all the policy assertions (i.e., requirements/capabilities) within that policy alternative. Returning to our example scenario, the policy alternatives (of the consumer and provider) requiring XML Encryption using AES-128 would be chosen. However, within those same policy alternatives, there might also be other requirements that must be fulfilled (e.g., that all messages are to include a timestamp).

The specific policy assertions are actually not defined within the Web Services Policy specification, as these are to be defined within domain specific specifications such as WS-SecurityPolicy (which will be discussed in Section IV-B2). The use of domain specific assertions makes the Web Services Policy framework highly adaptable to various application areas. Furthermore, because an entity is only required to understand the assertions within the policy alternative being used, incremental deployment of new assertions can easily be achieved. That is, by adding the new assertions within separate policy alternatives, the original policy alternatives may remain unchanged in order to provide backward-compatibility. Guidelines for defining policy assertions are provided in a separate working group note [30].

A policy according to Web Services Policy may be expressed in XML using one of two forms, normal form or compact form. Normal form is a straightforward representation of a policy's XML Infoset, enumerating each of its policy alternatives and their assertions. Alternatively, a policy may be more space efficiently represented using an equivalent compact form. To ensure interoperability, the specification recommends that the normal form is used where practical though.

The Web Services Policy specification also defines a domain-independent policy intersection algorithm. This policy intersection algorithm may be used when two or more communicating parties want to determine their set of compatible

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

policy alternatives. If defining new assertions, one should keep in mind that parameterized assertions may require a domain-specific policy intersection algorithm to be provided. Thus, parameterized assertions should preferably be avoided.

Apart from being exchanged between consumers and providers, Web Services Policy may also be used as a declarative language for configuring a system. This was for instance the case in Microsoft's Web Services Enhancements (WSE) 2.0. However, this approach has been abandoned in WSE 3.0 [31]. Apparently, the policies had a tendency to become too complex for manual editing. This may indicate that Web Services Policy in its current form is best suited as a language used by applications, and is less suitable for use by system developers in order to specify system policies (at least in more complex systems). Still, a compromise may be to utilize Web Services Policy for configuration but to provide preconfigured policies providing common configurations. This is the solution used in for instance the BEA WebLogic Server [32].

1) Web Services Policy - Attachment: Web Services Policy - Attachment [33] defines two general mechanisms for associating policies with the entities to which they apply. The first mechanism enables references to policies to be included within arbitrary XML elements. This way, Web services policies can be referenced within the entities' existing metadata. More specifically, this is done using a `PolicyURIs` attribute, containing a list of IRIs (Internationalized Resource Identifiers), referring to the policies. If more than one IRI is included within the `PolicyURIs` attribute, the referenced policies must be merged to obtain the applicable policy. In the case that policies are merged, conformance to the final applicable policy enforces conformance to all the referenced policies as well.

Alternatively, with the second mechanism, policies may be associated to the entities to which they apply through an external binding. For this purpose, a `PolicyAttachment` element is defined. This element contains a policy scope in addition to defining and/or referencing one or more policies. The policy scope identifies the entities to which the referenced/included policies apply. This way, policies can be associated with arbitrary entities.

Although the Web Services - Attachment specification defines two new mechanisms for attaching policies (i.e., `PolicyURIs` and `PolicyAttachment`), this does not prevent the `Policy` and `PolicyReference` elements (as defined in the Web Services Policy framework specification [18]) from being used directly as child elements within other XML elements. In fact, the Web Services Policy - Attachment specification advocates the use of these original mechanisms within WSDL and UDDI. Even though WSDL 1.1 forbids the use of extensibility elements/attributes within some elements, the WS-I Basic Profile 1.1 overrules this restriction and allows element extensibility everywhere [33].

2) WS-SecurityPolicy: As previously mentioned, the Web Services Policy specification itself does not define any policy assertions for expressing specific requirements and capabilities, as this is left to domain specific specifications. One such assertion specification is WS-SecurityPolicy [17], which defines policy assertions corresponding to the security features provided by WS-Security, WS-Trust, and WS-SecureConversation.

For instance, WS-SecurityPolicy defines two mechanisms for specifying the parts of a message that are to be integrity protected. With the `SignedParts` assertion, QNames are used to specify that the entire SOAP message body and/or selected headers require integrity protection. Alternatively, the XPath based `SignedElements` assertion may be used to specify arbitrary message elements requiring integrity protection. Although the names of these assertions suggest that integrity protection is to be provided through the use of digital signatures, this is not a requirement. Likewise, assertions are also defined for specifying the parts of a message that needs confidentiality protection. The `EncryptedParts` and `EncryptedElements` assertions are equivalent to their integrity counterparts (using QNames and XPaths respectively). Furthermore, the `ContentEncryptedElements` assertion allows XPaths to be used to specify arbitrary elements that require confidentiality protection of their content only. The `RequiredElements` and `RequiredParts` assertions may be used to specify header elements that the message must contain, using XPaths and QNames respectively.

Likewise, token assertions may be used to specify required tokens. The supported token types are those specified for WS-Security (i.e., the Username, X509, Kerberos, SAML, and REL tokens) and security context tokens according to WS-Trust and WS-SecureConversation. The WS-SecurityPolicy specification also defines a `KeyValueToken` assertion. Recall from Section III-A that the `KeyInfo` element of XML Signature provides for identifying a public key pair by including the public key value itself (i.e., in the `KeyValue` element). Hence, the `KeyValueToken` assertion provides a way to specify that the public key value must be included.

As an alternative to the token assertions corresponding to specific tokens, an `IssuedToken` assertion intended for use in combination with WS-Trust is also provided. Although WS-Trust is first to be discussed in the next section, let us say for now that WS-Trust defines a security token service from which security tokens can be requested. Such requests are made using a request security token message, where the requested security token is described by a `RequestSecurityTokenTemplate` element. In this context, the `IssuedToken` assertion can be used to provide/specify the `RequestSecurityTokenTemplate` (and identify the issuing security token service). Thus, the `IssuedToken` assertion serves two important functions: Not only does it provide for identifying tokens of arbitrary type but it also enables the token to be obtained, even when the details of the request message are unknown (and potentially incomprehensible) to the requester.

In addition to simply being able to specify the presence of a security token, assertions are also provided in order to specify that the token must contain specific claims and be issued by a specific issuer. Furthermore, a range of token specific assertions are defined for the different token types, enabling the characteristics of a token to be closely described. Moreover, it can be specified whether a token is required to be present in all messages (i.e., from the initiator, from the

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

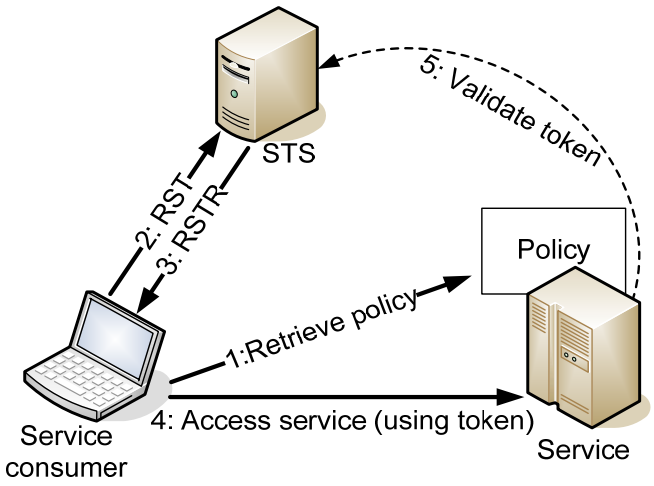

Fig. 8. The policy of the service (which is expressed using Web Services Policy/WS-SecurityPolicy) specifies a security token required to access the service. After retrieving the policy (e.g., from the WSDL file of the service), the service consumer uses the information in the policy to obtain the correct security token from the security token service (STS). The communication with the STS consists of a request security token (RST) element/message and a request security token response (RSTR) element/message. When receiving the security token from the service consumer, the service may have the token validated by the STS or validate the token itself.

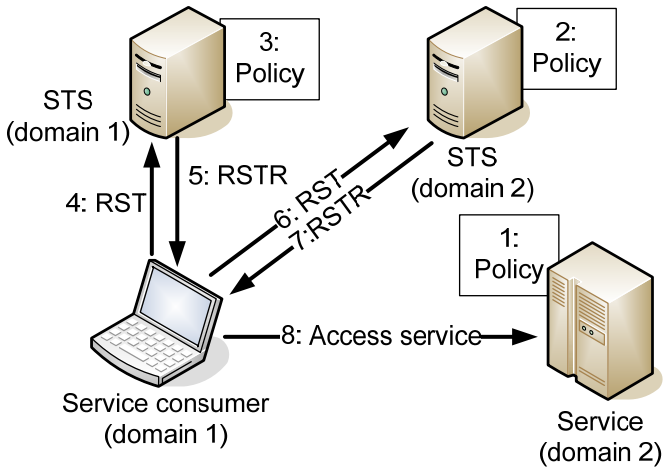

Fig. 9. The policy of the service specifies that a security token from the STS in domain 2 is required. The policy of the STS in domain 2 requires a security token from the STS in domain 1 (i.e., one of its policy alternatives accepts such a security token). The STS in domain 1, for which the service consumer has a valid username/password, requires a username token for authentication. After retrieving the policies in the given order (1, 2, and 3), the service consumer obtains a security token, from the STS in domain 1 (4-5), which is then used to obtain a security token from the STS in domain 2 (6-7). This last security token is then used to access the service (8).

recipient, or both ways), only in the first message, or is not required to be included at all (i.e., being identified through a reference instead).

Apart from the already mentioned assertions, WS-SecurityPolicy also defines assertions for identifying required/supported cryptographic algorithms and transport bindings (e.g., requiring the use of HTTPS), and for specifying the order in which confidentiality and integrity protection is to be applied. Furthermore, additional assertions can be found in the specification (in particular with regard to specifying the options of WS-Security and WS-Trust).

C. WS-Trust

We have seen in the previous sections that WS-Security provides for including security tokens in SOAP messages, while Web Services Policy and WS-SecurityPolicy together provides for specifying what security tokens are required (or supported). Thus, a scenario like the one shown in Figure 8 may easily occur. In this scenario, the service consumer wants to access a service. The service, however, requires a specific security token to be included in all SOAP messages for access to be granted. In order to facilitate interoperability, the service communicates this requirement using Web Services Policy (and the policy assertions provided by WS-SecurityPolicy). However, assuming that the service consumer does not have access to the appropriate security token, being aware of the requirement is not likely to help much apart from being able to generate a sensible error message. This is exactly where WS-Trust has its role, by providing mechanisms for security token management.

WS-Trust [15] augments the functionality of WS-Security and Web Services Policy/WS-SecurityPolicy by defining mechanisms for obtaining/issuing, renewing, cancelling, and validating security tokens. Specifically, a security token service (STS) is defined, providing these mechanisms as Web services. Thus, after discovering what security token is required, the service consumer may use WS-Trust in order to obtain the required token from an STS as illustrated in Figure 8. Then, when the service consumer attempts to access the service after having obtained the required security token, the service may rely on the security token service to validate the token or chose to perform the validation itself.

Although this appears to solve our example scenario, there are some important underlying assumptions. Clearly, in order for the security token to be of value, it must be trusted by the relying party (i.e., in our case the service). This trust may exist because the relying party has a pre-established trust relationship with the STS, implying that the relying party trusts the claims within security tokens issued by that STS. Considering that the relying party may specify the trusted issuer of a security token in its policy, this is not an unlikely scenario. Even when the relying party has no direct trust relationship with the STS, the relying party may sometimes still be able to trust security tokens issued by that service. This may for instance be the case for X.509 certificate tokens, whose trustworthiness is based on whether the certificate chain can be validated or not (relying on a trusted certificate authority). In this latter case it may very well be that the certificate is not really issued by the STS, but that the security token service simply provides an interface to obtain a certificate issued to the client by someone else.

Likewise, the STS also requires trust in the claims for which it vouches in a security token. Thus, the client will usually be required to supply the STS with some evidence of its identity and/or of the claims to be included in the security token. The client may for instance provide this evidence by authenticating using a username/password (token) or by

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

supplying a security token from some other trusted STS. This facilitates trust brokering, where a security token from one domain can be exchanged for another security token for use in another domain. This is illustrated in Figure 9, where the service consumer first obtains a security token from the STS in its own domain (i.e., by authenticating with a username/password). The security token issued by the STS in domain 1 is then used to obtain the security token (from the STS in domain 2) required to access the service. This illustrates the fact that an STS, like any other Web service, may require specific security tokens to be supplied. Also notice that the security token obtained from the STS in domain 2 may be of the same or a different type as the one issued by the STS in domain 1. In the prior case it could potentially be sufficient to have the STS in domain 2 to co-sign the security token issued by the STS in domain 1.

Please be aware that the scenarios shown in Figure 8 and Figure 9 are just examples of possible communication patterns. For instance, in Figure 9, the STS in domain 1 could potentially have obtained the final security token on behalf of the client from the STS in domain 2. Likewise, the service could potentially request the required security token on behalf of the client (even without the client requesting it). In fact, the STS might be collocated with the service in the sense that the security token is generated by the service and transmitted to the client using the message elements defined by WS-Trust. The request-response model may also be extended to include a challenge-response, or include negotiation.

Another important point is that the client should not be required to parse the token. Thus, all parameters required by the client (e.g., the token lifetime) should be included as part of the response message. For security tokens where a private-key is used as proof of possession, this key is also returned to the client (typically encrypted using XML Encryption unless a secure connection is used). In order to ensure that the token contains the required claims, the client may also specify the required claims in the request message.

Considering that security tokens (e.g., the X.509 certificate token or the SAML token) may bind a key with an identity, a security token service issuing, renewing, validating, and cancelling such security tokens offers similar functionality as that provided by the XML Key Management Specification (XKMS). Still, important differences exist. For instance, while WS-Trust can in principle be used to handle any type of security token, XKMS is primarily intended (and better suitable) for use together with XML Signature and XML Encryption, utilizing their `KeyInfo` element. Cancelling a security token is also different from revoking a certificate as we know it from public key infrastructures. Cancelling a security token at the issuer (i.e., the STS issuing the security token) simply means that the issuer will no longer renew or validate the token. Because there are no revocation lists, it is required to have a token validated by the issuer in order to make sure that is has not been cancelled.

D. WS-SecureConversation

In Section IV-A we discussed how WS-Security can be used to secure the integrity and confidentiality of SOAP messages. WS-Security provides no notion of a context for exchanging multiple messages however. WS-SecureConversation [16] therefore builds on WS-Security and WS-Trust to provide mechanisms for establishing and identifying a security context. The security context is shared by the communicating parties for the duration of the communication session, and has the benefit of providing an authenticated state with associated key material.

Consider for instance a scenario where the SOAP messages between a service consumer and provider require integrity protection. By using WS-Security/XML Signature, this may for instance be achieved by signing the messages using the private key of the sender and including the associated X.509 certificate token in the message. If multiple messages are to be exchanged, however, such a solution has several disadvantages. First of all, having to include the same X.509 certificate in multiple messages to the same recipient is clearly a waste of bandwidth. Also notice that such a stateless approach, where the certificate token is resent with each message, may incur the overhead of performing full certificate validation for each received message. Thus, in order to avoid these disadvantages, the service consumer and provider may establish a security context at the beginning of the conversation/session. This way, full key exchange and authentication (e.g., using X.509 certificate tokens) is only required to be performed when establishing the security context, and not for every message.

The security context is represented by a security context token, where the security context is identified by a URI. The specification defines three different ways to obtain the security context token (and thereby establishing the security context). All three methods utilize the WS-Trust framework in order to request/distribute the security context token. One way is for the context initiator to request a security context token from a security token service (STS), which then distributes the security context token to the communicating parties. If not to rely on an STS, the context initiator may instead create a security context token itself and unsolicited distribute this token to the other parties. Alternatively, the initiating party may send a request for a security context token to the other party, which may then return the security context token or initiate negotiations.

WS-SecureConversation also provides for establishing a shared secret among the communicating parties. The shared secret is distributed within a proof-of-possession token that is distributed together with the security context token. This proof-of-possession token contains a secret encrypted for the recipient of the token (e.g., using the public key of the recipient or a TLS connection). Although the shared secret might be used for encrypting and/or authenticating messages directly, the specification recommends that a new key is derived for each message. Key derivation is, by default, performed by hashing the shared secret together with some supplied parameters.[5] A derived key token is used to identify the key being used, by referencing the security context token and providing the (non-secret) parameters used for deriving the key from the

[5]This is similar, but not identical, to the key derivation discussed in Section IV-A1 (for use with the Username token).

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

shared secret or previously derived key. An XML Signature or Encryption element, utilizing the derived key, may refer to the derived key token in order to identify the key being used.

The security context token will typically have a lifetime. Thus, in case the security context expires before the end of the communication session, the token will have to be renewed. Likewise, the security context token may be explicitly cancelled if its lifetime lasts beyond the end of the communication session. It is also possible to amend the security context token with additional associated claims during the communication session. Renewal, cancellation, and amending of security context tokens are all performed using the mechanisms provided by WS-Trust.

V. Security Markup Languages

This section provides an overview of the eXtensible Access Control Markup Language (XACML) and the Security Assertion Markup Language (SAML), which have both been standardized within OASIS [34][35]. While the previous section focused on standards targeted exclusively at Web services, XACML and SAML are applicable to other types of systems as well. In particular, XACML may be used to define access control policies for any type of system, while SAML is also being used for single-sign-on web browsing.

Open-source implementations of both standards are freely available to developers. For instance, the Open SAML project [36] provides Java and C++ implementations of SAML. Likewise, an open source Java implementation of XACML is available from Sun [37]. An extensive list of XACML related work and products is provided by the XACML Technical Committee [38].

A. The eXtensible Access Control Markup Language (XACML)

The eXtensible Access Control Markup Language (XACML) [34] is a specification for defining access control policies using XML. In addition to defining a policy language for expressing policies, XACML also provides an architectural model. The basic architectural model is shown in Figure 10. As illustrated in the figure, policy enforcement is performed by one or more policy enforcement points (PEPs). A policy enforcement point again relies on a policy decision point (PDP) for deciding the outcome of a request, based on the policies applicable to the request.

XACML also relies on policy administration points (PAPs) and Policy Information Points (PIPs). Policy administration points are used to create policies and make them available to the PDP(s), although the exact features of a PAP are implementation dependent. The PAP may store the policies in one or more centralized locations or attach them to the resource(s) to which they apply. In the former case, the location(s) may be referenced by the resource(s). Policy information points, on the other hand, provide attributes of subjects, resources, and the environment (e.g., the role of a subject or the time of day). Such attributes may be required by a PDP in order to evaluate a request against a policy.

A context handler may be used to translate between native formats used by PIPs and the format used by PDPs (referred to as the XACML context), enabling a PDP to interoperate with native PIPs (e.g., LDAP servers). Likewise, a context handler may also be used to translate between a PDP and various PEPs, enabling non-XACML aware PEPs to rely on the same XACML PDP. Furthermore, notice that although the access requester and the resource are depicted as separate computers in the figure, this is not necessarily the case. In fact, the resource, PEP, PDP, PAP, PIP, and access requester could potentially all be located on the same machine.

Let us now turn our attention to the XACML policy language. As shown in Figure 11, the XACML rule constitutes the basic building block for defining policies in XACML. Each rule has an effect, which is either permit or deny. Furthermore, each rule may specify a target. The target of a rule defines the subjects, resources, actions, and/or environments to which the rule applies (i.e., who may, or may not, do what to which resource given the environment).

A rule may also contain a condition, further restricting the applicability of the rule. Such a condition may involve attributes of the subject, resource, action, and/or environment, and can make use of arithmetical, comparative, set, and Boolean operators. Thus, XACML provides high granularity for defining rules and allows rules to be made context sensitive. A condition may for instance involve the role of the subject, the time of day, and/or previous events.

A XACML rule is not to exist on its own, but instead as part of a XACML policy. In case there is more than one rule in a policy, these are interrelated by a rule-combining algorithm. Three different rule combining algorithms are defined: deny-overrides, permit-overrides, and first-applicable. In addition, custom algorithms can be defined.

The target of a policy may be determined from the targets of its rules or be specified explicitly. If no target is specified by a rule, the target of the rule is taken to be the same as the target of the containing policy. Anyway, the target is used by the PDP to determine if the policy/rule is applicable to a given request. Consequently, the effective target of a rule is at least as strict as the target of the containing policy (i.e., PDPs only consider the rules within applicable policies).

A policy may also specify obligations. Such obligations may for instance be that an e-mail should be sent to the resource owner if access is granted or that denied requests for access should be logged. In order to ensure that the obligations are fulfilled, any obligations should be carried out by the PEP before granting access.

Policies may again be combined into a policy set in basically the same way as rules are combined into policies. The algorithms for this are equivalent to the ones for combining rules, with the addition of an only-one-applicable algorithm, where only one policy is to be applicable to a given request/target.

A request may have multiple subjects, but only one action and one resource (some exceptions for multiple resources are specified in a separate profile). A PDP's response to a request is either permit, deny, not applicable (i.e., if no policy/rule was applicable), or indeterminate (i.e., if an error occurred). One or more obligations may also be specified, and the PEP must

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

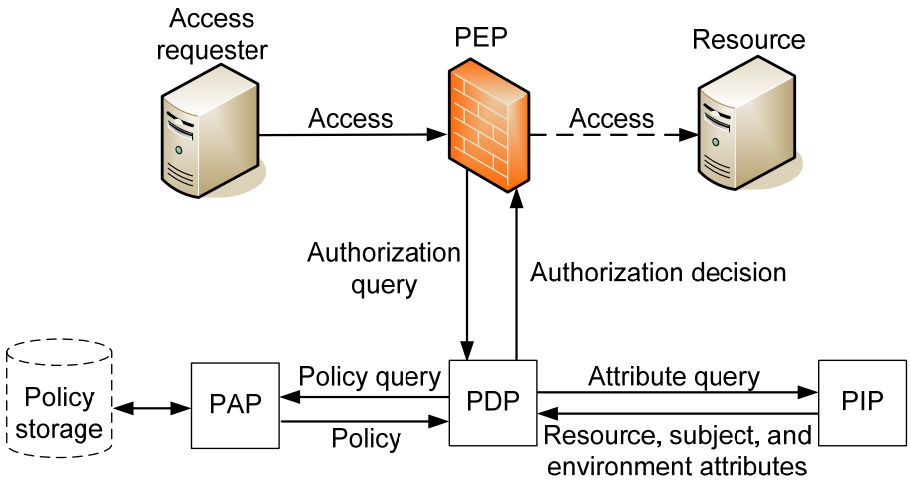

Fig. 10. The architectural/usage model of XACML, containing a policy decision point (PDP), a policy enforcement point (PEP), a policy administration point (PAP), and a policy information point (PIP). The access requester wants to access the resource.

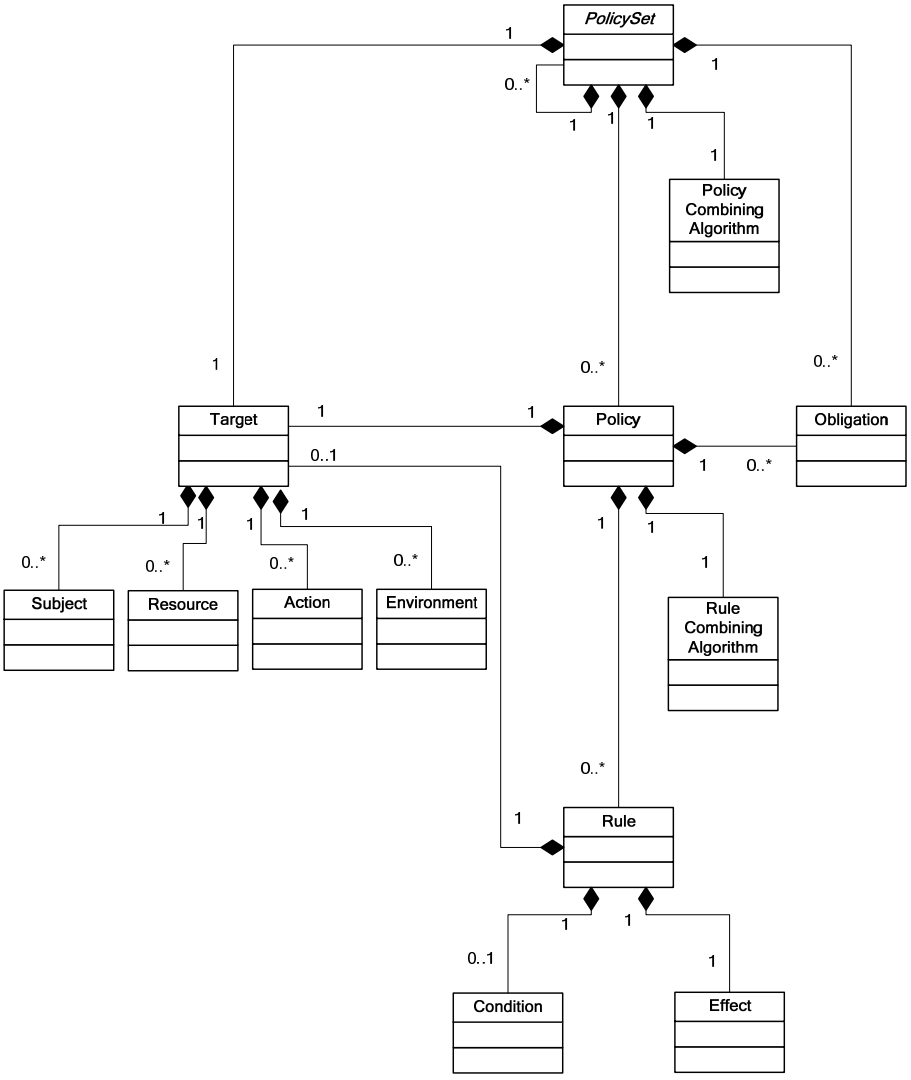

Fig. 11. The XACML policy language model [34].

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

deny access unless it can fulfill all the obligations.

While the description here is based on the current version of XACML (i.e., version 2.0), the XACML technical committee within OASIS is currently working on the specification for XACML 3.0. One of the major changes will be with regard to the `Target` element, which will be based on a generic matching mechanism instead of using the current special target categories (i.e., the subject, resource, action, and environment categories). For anyone wanting to experiment with XACML 3.0, there is a preliminary patch for Sun's XACML implementation available from the Swedish Institute of Computer Science (SICS) [39].

In addition to the core specification, XACML also has several profiles. We will now give a brief overview of each of these profiles.

1) The Privacy Policy Profile: The Privacy policy profile [40] defines two attributes for specifying the purpose for which personal identifiable information is collected. It also defines a rule for enforcing that the information/resource is being used according to the purpose for which it was collected.

2) The SAML Profile: The SAML profile [41] defines how to use SAML to carry XACML policies, queries, and responses. In particular, the profile extends SAML with a `XACMLAuthzDecisionQuery` and a `XACMLAuthzDecisionStatement` which may be used to communicate authorization queries and responses between a policy enforcement point (PEP) and a policy decision point (PDP). The profile also defines a `XACMLPolicyQuery` and a `XACMLPolicyStatement` which may be used to query and distribute policies from a policy administration point (PAP). Furthermore, it is defined how standard SAML attribute requests and responses are to be used for requesting and exchanging XACML attributes, defining a mapping between SAML and XACML attributes.

By defining extensions to SAML, the profile enables the full functionality of XACML to be utilized when used in conjunction with SAML. In addition, the `XACMLAuthzDecisionStatement`, the `XACMLPolicyStatement`, and SAML's attribute statement may be used as part of SAML assertions for storing authorizations, policies, and attributes respectively.

3) The XML Signature Profile: The XML Signature profile [42] recommends that XACML schema instances are embedded in SAML assertions, requests, and responses as defined in the SAML Profile [41]. These SAML objects should then be canonicalized and signed according to the SAML specification [35]. The use of SAML for this purpose has the advantage of providing a format for specifying the identity of the signer and a validity period.

4) The Core and Hierarchical Role Based Access Control (RBAC) Profile: The RBAC profile [43] specifies how XACML can be used to meet the requirements for "core" and "hierarchical" role based access control according to the ANSI/INCITS standard [44]. In order to meet these requirements, the RBAC profile defines four types of policies, out of which the first two are mandatory:

- A role policy set is used to associate a given role with a permission policy set.
- A permission policy set defines the permissions associated with a given role.
- A role assignment policy (or policy set) may be used to define which roles can be enabled or assigned to which subjects. The role assignment policy may also impose restrictions on combinations of roles or the total number of roles held by a subject.
- A `HasPrivilegesOfRole` policy may be included within each permission policy set in order to support queries asking whether a subject has the privileges of a given role. It can not be used to answer questions such as what roles are associated with a given subject though.

By including multiple roles in the target of a role policy set, that policy set only applies to subjects having all the specified roles enabled. In order to support hierarchical roles, the permission policy set for a senior role may refer to the permission policy set(s) of its junior role(s) in order to inherit those privileges as well. It is recommended in the profile that roles are specified as values of a role attribute. However, roles may also be identified by defining separate attributes for each role.

5) The Hierarchical Resource Profile: The Hierarchical resource profile [45] defines how XACML can be efficiently used to provide access control for a resource organized as a hierarchy (e.g., file systems, XML documents, and organizations). The hierarchical resource is required to form a directed acyclic graph (i.e., a tree or a forest) and may be represented as an XML document or in some other way. The individual resources that are part of the hierarchical resource are referred to as nodes. The profile defines how the identities of nodes are to be consistently represented (using XPath if the hierarchical resource is represented as an XML document and URIs otherwise). Furthermore, the profile defines how to request access to a node and suggests how to define policies applying to multiple nodes.

6) The Multiple Resource Profile: The Multiple resource profile [46] defines how access to multiple resources can be requested in a single request to a PDP, and how the response to such a request can be sent in a single response.

7) The Web Services Profile: Although it has not yet reached standardization status, the Web services profile [47] defines XACML related policy assertions for use with Web Services Policy. The `XACMLAuthzAssertion` may for instance be used to communicate required/provided role attributes or to specify the willingness/requirement to fulfill certain obligations (such as encrypting stored data). The `XACMLPrivacyAssertion` may for instance be used to specify the intended/acceptable use, distribution, and time of retention for a resource.

8) XACML 3.0 Administrative Policy: There is also work in progress on an Administrative policy profile [48], defining how administration and delegation policies can be expressed in XACML 3.0. In particular, policy administration policies may be used to define the types of policies that individuals can create and modify. Delegation policies may permit users to dynamically create policies of limited duration in order to delegate capabilities to others.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

B. The Security Assertion Markup Language (SAML)

The Security Assertion Markup Language [35] defines how to express security assertions in XML. Conceptually, an assertion is a set of statements, made by an asserting party (i.e., a SAML authority), that a relying party may trust. To clarify this, we will consider the contents of SAML assertions in more detail.

As indicated in Figure 12, the asserting party issuing the assertion is identified by the `Issuer` element. Although the `Issuer` element is the only required element of an assertion, an assertion without any statements is generally not of much use. Thus, SAML defines three different statement types, namely authentication, authorization, and attribute statements. In order to provide for extensibility, these three statement types are all derived from the same abstract type (i.e., the `Statement` element), from which additional statement types may be derived. As shown in the figure, a SAML assertion can contain any number of statements.

An assertion containing an authentication, authorization, or attribute statement is also required to specify the subject to which the assertion(s) apply, utilizing the `Subject` element. Apart from identifying the subject, the `Subject` element may also specify methods for subject confirmation. Such methods for subject confirmation may be used by the relying party to confirm that a message indeed came from the subject identified in the assertion (or an associated entity of the subject). There are several such methods for subject confirmation [49]. Typically, the subject may authenticate itself by signing the message using a private key associated with the assertion. Alternatively, other application specific procedures may be applied or it may be considered sufficient to be in possession of the assertion (potentially in combination with some other constraint(s), such as a short validity period).

Let us now take a closer look at the three defined statement types, starting with the authentication statement. An authentication statement specifies how and when the subject was authenticated. As a wide range of authentication methods exists, how to specify the authentication context is covered in a separate standards document [50].

An authorization decision statement states whether authorization to perform specific actions on a specific resource has been granted or not (or alternatively that authorization could not be decided). An authorization statement may also refer to or include other assertions, upon which the authorization decision was made. Finally, the attribute statement element may include one or more attributes associated with the subject, e.g., the subject's role or credit limit. Such attributes may be included in plaintext or in encrypted form.

The `Conditions` element allows an asserting party to place restrictions on the valid use of an assertion. Such restrictions may specify the intended audience (i.e., restricting the potential relying parties) or the assertion's validity period. It may also be indicated that the information within the assertion is likely to change soon, and that the assertion should therefore only be relied upon once. In fact, considering the lack of a revocation model in SAML, it is preferable that assertions have a relatively short lifetime. Restrictions may also be placed on the use of the assertion for issuing new assertions (with regard to the audience of the new assertion and the number of indirections).

The `Signature` element provides for protecting the integrity of the assertion by applying a signature. Typically, the asserting party will sign the assertion using its private key, thereby providing proof of the assertion's origin. Finally, the `Advice` element may be used by the asserting party for communicating additional information. This additional information should not affect the validity of the assertion, however, as applications are allowed to ignore this element.

1) SAML Protocols: In addition to defining the syntax and semantics of SAML assertions, SAML also defines various request/response protocols. For instance, an assertion query and request protocol is defined. This protocol may be used for requesting an existing assertion by referring its unique identifier or for querying assertions based on subject and statement type. In the latter case, queries are defined for all of the three statement types. Thus, an authentication query can be used to obtain the assertions containing authentication statements for a given subject, while an attribute query can be used to obtain assertions containing specific attributes of a given subject. Likewise, an authorization decision query may be used to query whether specific actions should be permitted on a specific resource by the given subject. However, as noted in the specification, one may consider using XACML instead for this latter purpose. Considering that XACML provides richer features for authorization decisions and is very well suited for being used in combination with SAML, using XACML is clearly a good option.

SAML also defines an authentication request protocol. The authentication request protocol differs from the authentication query just discussed by being a request for having authentication performed by an identity provider, while the authentication query was for existing authentication assertions. How authentication is performed by the authentication provider is outside the scope of the protocol, although the entity requesting the authentication to be performed may specify requirements on how the authentication is to be performed. One potential usage of this protocol would be a service provider requesting a client to authenticate, providing the client a list of trusted identity providers. The client could then present this authentication request to one of the listed identity providers, which upon successful authentication returns an assertion containing an authentication statement.

In addition to the already discussed protocols, SAML also defines a single logout protocol and protocols for managing the identifiers used, between identity providers and service providers, for identifying principals. Bindings for the SAML protocols (i.e., to SOAP and HTTP) are specified in a separate document [51].

2) Usage Scenarios: Being designed to work without a single centralized authority, SAML has many potential uses in scenarios where having a central authority is challenging from a political or technical point of view. Potential usage scenarios include single sign-on, federated identity, authorization, and SOAP message security (i.e., being used as a security token for WS-Security). As discussed in Section V-A2, SAML is

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

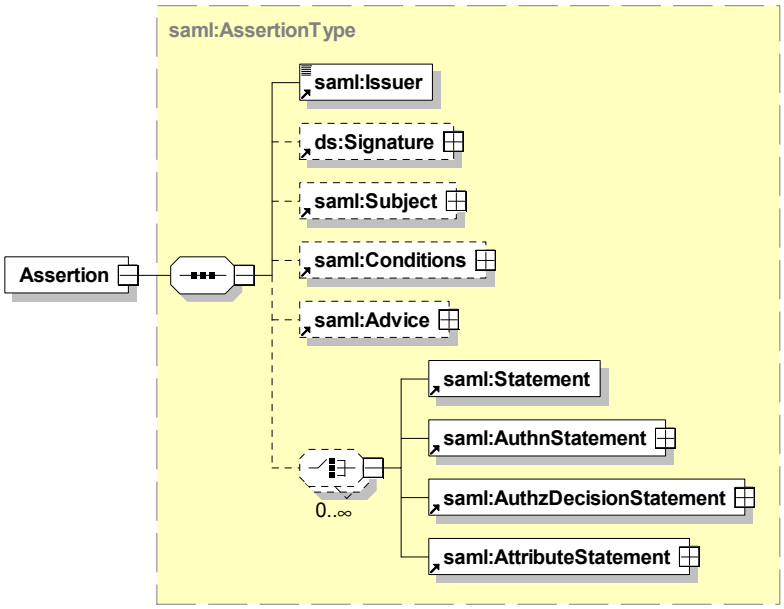

Fig. 12. The SAML assertion element.

also well suited for being used in combination with XACML. For additional and more in-depth usage scenario examples, please refer to the SAML technical overview [52].

VI. Final Remarks

In this paper we have provided an overview of current security standards for XML and Web services. Together these standards provide a flexible framework for fulfilling basic security requirements such as confidentiality, integrity, and authentication, as well as more complex requirements such as non-repudiation, authorization, and federated identities. Furthermore, the standards offer flexibility with respect to the cryptographic algorithms used, facilitating adaptation of stronger algorithms if required.

The flexibility and high number of options does nonetheless come at the cost of an increased risk of erroneous use. For instance, the option to only sign parts of a message may put an implementation at risk if its security in some direct or indirect way depends on message parts that are not signed. The combination of relatively complex policies and subjects operating across organizational boundaries may also require advanced management and auditing tools, and may in some cases make it difficult to determine exactly who has access to a given resource.

Mechanisms such as those provided by Web Services Policy and the Web Services Description Language (WSDL) may also provide valuable sources of information to an attacker trying to find weaknesses in a system. Furthermore, the severity of a single vulnerability may be amplified when federated identities or trust brokering is being used. When relying on trust brokering or other trust relationships, it is also essential to ensure that the level of trust is sufficient for the application at hand.

In addition to more common security issues, there are also some attacks/vulnerabilities that are specific to XML [53][54][55]. Although XML firewalls may be able to detect messages trying to exploit these vulnerabilities, the use of end-to-end encryption may effectively prevent such detection. Consequently, XML parsers and other affected applications should be able to handle such messages in a secure manner. Thus, in summary, although the standards discussed in this paper provide essential mechanisms for successfully deploying secure Web services, they do not provide a complete solution.

We will now conclude this paper by providing some references to related standards and specifications. In particular, the Web services security roadmap [13] from IBM and Microsoft also proposed three other specifications, namely WS-Privacy, WS-Federation, and WS-Authorization, in addition to those discussed in Section IV. Because none of these additional proposals have become standardized, they were not included in this paper. There is, however, a technical committee [56] within OASIS working to standardize WS-Federation.

WS-Federation extends WS-Trust in order to provide federated identities. Recall from our discussion of WS-Trust that a security token service (STS), supporting a range of security token types and with the proper trust relationships, can provide a cornerstone for brokering trust and federating identities between different domains. Although this is similar to what is offered by SAML, a key difference is that WS-Federation is independent of the security token type. Considering that SAML and WS-Federation are both strongly supported, they appear likely to coexist in the imminent future. Because the standardization process for WS-Federation is still in an early stage, however, WS-Federation is not covered in more detail in this paper. The interested reader is referred to the overview provided by Goodner et al. [57].

Another specification of interest is WS-MetadataExchange [58], which may be used to request and exchange metadata, including policies.

In addition to the OASIS standards discussed previously in

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

this paper, there is also an OASIS standard defining how to represent biometric information in XML, that is, the XML Common Biometric Format (XCBF) [59]. Because XCBF has very specific (and relatively narrow) usage, it was not included in this paper.

WS-ReliableMessaging [60] (which is also an OASIS standard) may be used to implement ordered and guaranteed delivery of SOAP messages (without duplicates). Considering that guaranteed and ordered delivery may be fundamental for the security of some applications, WS-ReliableMessaging may beneficially be used together with WS-SecureConversation for securing sequences of messages.

Finally, considering that the security standards discussed in this paper are of recent date, they should be expected to evolve as more experience on their use is gained. Also, as new Web services technologies and usage patterns become established, additional security functionality will likely be required. Consider for instance the Web Services Business Process Execution Language (WS-BPEL) [61]. WS-BPEL allows executable business processes to be defined by specifying the interactions among several Web services, thereby creating a composite service. Such an executable business process may potentially be defined by an external party and involve both internal and external services. Thus, some method to assure that such a processes is in accordance with the local security policy is required [62]. Nevertheless, the security standards discussed in this paper are applicable to such composite services as well, e.g., securing the interactions within the composite service or enforcing access control to the resulting service. It is also worth to notice that WS-BPEL may be used to implement fault tolerance in Web services [63], thereby complementing the standards discussed in this paper.

REFERENCES

[1] D. Booth, H. Haas, F. McCabe, E. Newcomer, C. Ferris, and D. Orchard. "Web Services Architecture," W3C Working Group Note, 2004.

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 1, 2002, pp. 86-93.

[3] D. Eastlake, J. Reagle, D. Solo, F. Hirsch, and T. Roessler, "XML Signature Syntax and Processing (Second Edition)," W3C Recommendation, 2008.

[4] D. Eastlake and J. Reagle, "XML Encryption Syntax and Processing," W3C Recommendation, 2002.

[5] P. Hallam-Baker and S. H. Mysore, "XML Key Management Specification (XKMS 2.0)," W3C Recommendation, 2005.

[6] D. Eastlake 3rd, J. Reagle, and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing," IETF RFC 3275, 2002.

[7] J. Boyer, "Canonical XML Version 1.0," W3C Recommendation, 2001.

[8] J. Boyer and G. Marcy, "Canonical XML 1.1," W3C Recommendation, 2008.

[9] J. Boyer, D. Eastlake 3rd, and J. Reagle, "Exclusive XML Canonicalization Version 1.0," W3C Recommendation, 2002.

[10] S. Aissi, A. Hately, and M. Hondo, "Schema Centric XML Canonicalization Version 1.0," http://www.uddi.org/pubs/SchemaCentricCanonicalization.htm, 2005.

[11] M. Hughes, T. Imamura, and H. Maruyama, "Decryption Transform for XML Signature," W3C Recommendation, 2002.

[12] D. Eastlake 3rd, "Additional XML Security Uniform Resource Identifiers," IETF RFC 4051, 2005.

[13] IBM Corporation and Microsoft Corporation, "Security in a Web Services World: A Proposed Architecture and Roadmap," http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-secmap.pdf, 2002.

[14] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," OASIS Standard, 2006.

[15] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, "WS-Trust 1.3," OASIS Standard, 2007.

[16] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, "WS-SecureConversation 1.3," OASIS Standard, 2007.

[17] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, "WS-Security Policy 1.2," OASIS Standard, 2007.

[18] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü. Yalcinalp, "Web Services Policy 1.5 - Framework," W3C Recommendation, 2007.

[19] Sun Microsystems, "The WSIT Tutorial," http://java.sun.com/webservices/reference/tutorials/wsit/doc/, 2007.

[20] Microsoft Corporation, "Web Services Protocols Supported by System-Provided Interoperability Bindings," http://msdn2.microsoft.com/en-us/library/ms730294.aspx, 2007.

[21] Layer 7 Technologies, "XML Firewall and VPN," http://www.layer7tech.com/products/page.html?id=70, 2007.

[22] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security X.509 Certificate Token Profile 1.1," OASIS Standard, 2006.

[23] T. DeMartini, A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security Rights Expression Language (REL) Token Profile 1.1," OASIS Standard, 2006.

[24] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security Kerberos Token Profile 1.1," OASIS Standard, 2006.

[25] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security UsernameToken Profile 1.1," OASIS Standard, 2006.

[26] R. Monzillo, C. Kaler, A. Nadalin, and P. Hallam-Baker, "Web Services Security: SAML Token Profile 1.1," OASIS Standard, 2006.

[27] F. Hirsch, "Web Services Security SOAP Messages with Attachments (SwA) Profile 1.1," OASIS Standard, 2006.

[28] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," IETF RFC 1510, 1993.

[29] M. McIntosh, M. Gudgin, K. S. Morrison, and A. Barbir, "Basic Security Profile Version 1.0," Web Services Interoperability Organization (WS-I) Final Material, 2007.

[30] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü. Yalcinalp, "Web Services Policy 1.5 - Guidelines for Policy Assertion Authors," W3C Working Group Note, 2007.

[31] A. Skonnard, "Migrating to WSE 3.0," http://msdn.microsoft.com/msdnmag/issues/06/04/ServiceStation/, 2006.

[32] BEA, "WebLogic Web Services: Security - Configuring Message-Level Security," http://edocs.bea.com/wls/docs100/webserv_sec/message.html, 2008.

[33] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü. Yalcinalp, "Web Services Policy 1.5 - Attachment," W3C Recommendation, 2007.

[34] T. Moses, "eXtensible Access Control Markup Language (XACML) version 2.0," OASIS Standard, 2005.

[35] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0," OASIS Standard, 2005.

[36] Internet 2, "OpenSAML - an Open Source Security Assertion Markup Language toolkit," http://www.opensaml.org/, 2007.

[37] Sun Microsystems, "Sun's XACML Implementation," http://sunxacml.sourceforge.net/, 2006.

[38] OASIS Open, "XACML References and Products, Version 1.83," http://docs.oasis-open.org/xacml/xacmlRefs.html, 2007.

[39] Swedish Institute of Computer Science, "SICS's implementation of the XACML 3.0 draft," http://www.sics.se/spot/xacml_3_0, 2007.

[40] T. Moses, "Privacy policy profile of XACML v2.0," OASIS Standard, 2005.

[41] A. Anderson and Hal Lockhart, "SAML 2.0 profile of XACML v2.0," OASIS Standard, 2005.

[42] A. Anderson, "XML Digital Signature profile of XACML v2.0," OASIS Standard, 2005.

[43] A. Anderson, "Core and hierarchical role based access control (RBAC) profile of XACML v2.0," OASIS Standard, 2005.

[44] American National Standards Institute, "ANSI INCITS 359-2004, Role Based Access Control," 2007.

[45] A. Anderson, "Hierarchical resource profile of XACML v2.0," OASIS Standard, 2005.

[46] A. Anderson, "Multiple resource profile of XACML v2.0," OASIS Standard, 2005.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302

[47] A. Anderson, "Web Services Profile of XACML (WS-XACML) Version 1.0," OASIS XACML TC Working Draft, 2007.
[48] E. Rissanen, H. Lockhart, and T. Moses, "XACML v3.0 Administrative Policy Version 1.0," OASIS XACML TC Working Draft, 2007.
[49] J. Hughes, S. Cantor, J. Hodges, F. Hirsch, P. Mishra, R. Philpott, and E. Maler, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard, 2005.
[50] J. Kemp, S. Cantor, P. Mishra, R. Philpott, and E. Maler, "Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard, 2005.
[51] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, and E. Maler, "Bindings for the OASIS Assertion Markup Language (SAML) V2.0," OASIS Standard, 2005.
[52] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo, "Security Assertion Markup Language (SAML) V2.0 Technical Overview," http://www.oasis-open.org/committees/download.php/23920/sstc-saml-tech-overview-2.0-cd-01.pdf, 2007.
[53] E. Moradian and A. Håkansson, "Possible attacks on XML Web Services," *International Journal of Computer Science and Network Security*, vol. 6, no. 1B, pp. 154-170, 2006.
[54] P. Lindstrom, "Attacking and Defending Web Services," http://forumsystems.com/papers/Attacking_and_Defending_WS.pdf, 2004.
[55] B. Hill, "A Taxonomy of Attacks against XML Digital Signatures & Encryption," http://www.isecpartners.com/files/iSEC_HILL_AttackingXMLSecurity_Handout.pdf, 2004.
[56] "OASIS Web Services Federation (WSFED) TC," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsfed, 2008.
[57] M. Goodner, M. Hondo, A. Nadalin, M. McIntosh, and D. Schmidt, "Understanding WS-Federation," http://msdn2.microsoft.com/en-us/library/bb498017.aspx, 2007.
[58] K. Ballinger et al., "Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1," http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-mex/metadataexchange.pdf, 2006.
[59] J. Larmouth, "XML Common Biometric Format," OASIS Standard, 2003.
[60] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and Ü. Yalcinalp, "Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1," OASIS Standard, 2007.
[61] C. Barreto et al., "Web Services Business Process Execution Language Version 2.0 - Primer," http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf, 2007.
[62] K.-P. Fischer, U. Bleimann, W. Fuhrmann, and S. M. Furnell, "Security Policy Enforcement in BPEL-Defined Collaborative Business Processes," *Proc. 23rd International Conference on Data Engineering Worskshops*, 2007, pp. 685-694.
[63] G. Dobson, "Using WS-BPEL to Implement Software Fault Tolerance for Web Services," *Proc. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, 2006, pp. 126-133.

BIOGRAPHY

Nils Agne Nordbotten (nils.nordbotten@ffi.no) has a PhD in computer science from the University of Oslo. He is currently a research scientist at the Norwegian Defence Research Establishment (FFI). Before joining FFI in 2007, he was employed as a research fellow at Simula Research Laboratory. His main research interests are within networks and distributed systems, in particular with regard to security and fault tolerance.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/SURV.2009.090302