Expressiveness Considerations of XML Signatures

Meiko Jensen, Christopher Meyer
Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
{Meiko.Jensen,Christopher.Meyer}@ruhr-uni-bochum.de

***Abstract*—XML Signatures are used to protect XML-based Web Service communication against a broad range of attacks related to man-in-the-middle scenarios. However, due to the complexity of the Web Services specification landscape, the task of applying XML Signatures in a robust and reliable manner becomes more and more challenging.**

In this paper, we investigate this issue, describing how an attacker can still interfere with Web Services communication even in the presence of XML Signatures. Additionally, we discuss the interrelation of XML Signatures and XML Encryption, focussing on their security properties and expressiveness in different application scenarios.

I. Introduction

The rapid development in e-commerce applications has brought up a large set of technologies and standards for network-based business communication. Among these technologies, the XML-based specifications of the so-called Web Services platform [1] received particular attention, mostly due to their widespread use in industry and their capabilities in terms of flexibility and manageability.

However, with new Web Services specifications emerging rapidly, and with ongoing modification of the existing ones, the overall landscape of Web Services standards has changed. Today's Web Services world has seen lots of issues related to missing interoperability of implementations and incompatibility among complementary standards (e.g. [2]).

For the particular task of security for Web Services communication, this high degree of complexity causes severe problems for Web Service developers. This is due to the issue that the correct and secure use of the corresponding Web Services specifications requires an in-depth knowledge on their concepts and especially their pitfalls for proper use.

In this paper, we will examine the Web Services specifications involved in applying digital signatures to Web Services communication. Namely, the particular specifications are XML Signature [3], WS-Security [4], SOAP [5], XML Encryption [6], and WS-Addressing [7]. We will discuss the purposes of digital signatures in network-based communication, and focus on their implications to the proper use of the XML Signature specification in interrelation to other existing Web Service specifications.

The paper is organized as follows: The next section describes the evolution of digital signatures in communication networks, and reasons about their necessity and purposes. Section III then discusses some basic issues and security considerations for different application styles of XML Signatures to SOAP messages, and Section IV goes into detail on the interrelation of XML Signature and XML Encryption, also giving advice on how to reach a maximum of security here. The paper is concluded with future research directions in Section V.

This work was partially funded by the Sec2 project of the German Federal Ministry of Education and Research (BMBF, FKZ: 01BY1030).

II. Digital Signatures and XML

A. Digital Signature Basics

The cryptographic primitive of a digital signature can be used for enforcing a set of security-related requirements. Among these, the properties of *integrity*, *authenticity*, and *non-repudiation* are commonly considered the most important ones. Further, digital signatures can play key roles in authentication protocols or on providing application-specific proofs of authorization (e.g. in the sense of a hand-written signature beneath a contract document).

The use of digital signatures typically consists of two separate tasks. The $sign(input, key_{priv})$ operation uses a *private key* to produce a check value for a given *input*, the *signature value*, which can later-on be verified by the corresponding $verify(input, key_{public})$ operation. If the inputs are identical in both tasks, and if the private key used in $sign()$ belongs to the public key used in $verify()$, the $verify()$ operation will return $true$, in all other cases it will return $false$.

When applying this cryptographic primitive to network-based communication, the cryptographic properties ensure that the requirements listed above can be enforced. For instance, integrity directly correlates to the property that the $input$ parameter of $sign()$ and $verify()$ have to be identical in both operation executions in order to result in a successful signature verification. Authenticity relies on the fact that every entity holds an unique private/public key pair. Obviously, the private key has to be kept secret here.

B. Digital Signature Standards Evolution

In order to make the mathematical foundations of digital signatures accessible to a world-wide set of adopters, the major players of both industry and academia fostered the specification of a set of standards that explain how digital signatures are to be used in digital systems. Several specifications emerged, each specifying protocols and notations (or subsets of them) for interoperable use of digital signatures. For instance, the PGP toolset [8] provides a full standard with open source implementations for applying and verifying digital signatures of files. X.509 [9] provides means to identify the entities behind public keys, S/MIME [10] enables integrity-protected e-mail communication, and TLS ([11], formerly known as SSL) provides an authenticated, secure and integrity-protected data exchange channel.

However, as all of these standards had their limitations in capabilities, the W3C decided to pose a new specification for the use of digital signatures for the (that time new) eXtensible Markup Language (XML). This specification, today named XML Signature [3], provides a lot of new features in terms of flexibility and interoperability support for the use of digital signatures. For instance, it supports digital signatures covering only selective parts of an XML document (in contrast to e.g. PGP, which always requires a full document to be signed). Further options are embedding signature metadata like X.509 certificates along with the signature value in an arbitrary location within the XML document (or even elsewhere, by using an URI-based referencing scheme) or support for an unlimited number of digital signatures within the same XML document.

The large gain in flexibility provided by the XML Signature specification enabled a broad scope of possible application scenarios. As a result the effort to standardize these application scenarios ended up in its own adaptation of XML Signature, namely the WS-Security specification [4]. It defines the particular conditions and parameters on how to apply XML Signatures to SOAP messages in order to enable the security properties it provides for use in Web Service communication.

C. XML Signatures for SOAP Messages

According to the WS-Security specification, a typical XML Signature applied to the body of a common SOAP message looks like shown in Figure 1. It can be seen that the signature metadata is placed within a designated new SOAP header element named `Security`, and that it contains a reference to the contents that actually have been signed (here the SOAP body) along with the resulting hash value and signature value. Here, one has to notice that the calculation of XML Signatures uses a two-step calculation: first, the hash value over the referenced parts (here the body) is calculated and added to the designated `DigestValue` element of the `Reference` block. Then, the whole `SignedInfo` subtree (including the *DigestValue*) is hashed, the resulting hash value is taken as $input$ to the particular $sign()$ operation, and the resulting signature value is then placed in the `SignatureValue` element of the `Signature` block.

On verification, these steps have to be reproduced identically, just taking the public key and the $verify()$ operation instead of private key and $sign()$. If either the referenced contents or the `SignedInfo` block have been modified since the signature application, the verification will fail.

D. Threat Model

Discussing security always also requires a discussion on the threats that cause the particular security requirements. In the scenario of SOAP-based communication, the threat to be prevented with XML Signatures consists of an attacker modifying the (application-critical) contents of the Web Service communication while the message is in transit. Thus, the underlying threat model consists of a Dolev-Yao attacker [12] between the entity applying digital signatures and the entity verifying them. If the system is designed in a way that no such attacker exists, the use of digital signatures can be omitted, as data integrity and authenticity is provided automatically. For real-world scenarios instead, such an attacker might be instantiated by malicious hackers, but also angry employees, a service's Cloud provider, or even an Internet Service Provider.

Thus, for the remainder of this paper, we will assume a Dolev-Yao attacker to be present and to have full access to all SOAP communication between a Web Service client and server.

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:op="http://my.operation">
   <soap:Header>
      <wsse:Security
         xmlns:wsse="http://docs.oasis-open.org/wss/...">
         <ds:Signature
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod
                     Algorithm="http://.../xml-exc-c14n#"/>
               <ds:SignatureMethod
                     Algorithm="http://.../xmldsig#rsa-sha1"/>
               <ds:Reference URI="#myID">
                  <ds:Transforms>
                     <ds:Transform
                        Algorithm="http://.../xml-exc-c14n#"/>
                  </ds:Transforms>
                  <ds:DigestMethod
                      Algorithm="http://...xmldsig#sha1"/>
                  <ds:DigestValue>JVxbSj...</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
              d4UfGRmS199IyZp3TaDi...
            </ds:SignatureValue>
         </ds:Signature>
      </wsse:Security>
   </soap:Header>
   <soap:Body Id="myID">
      <op:greet>
         <name>John Doe</name>
      </op:greet>
   </soap:Body>
</soap:Envelope>
```

Fig. 1. Example of a signed SOAP message

III. The Interrelation of XML Signatures and other WS-* Specifications

The most easy way of protecting SOAP-based communication against modification consists in applying a digital signature on the whole SOAP message document. This way, a single signature value protects the whole message's integrity, and given that only one signature is required, the owner of its private key can directly be identified as the message's originator.

However, for the world of Web Services this approach is not always satisfactory. For instance, one of the major advances of using SOAP messages consists in its ability to have a single message bypass a lot of intermediary processing entities before the message reaches the ultimate recipient. Each of these intermediaries may add new SOAP headers, containing particular metadata for additional functionality or non-functional parameters. After all, also XML Signatures are nothing but an example for such additional metadata. Thus, if the whole SOAP document was protected by a single digital signature, it would be impossible to add such metadata without invalidating the signature.

Thus, instead of having a single XML Signature covering the whole document, practice turned out to favorize an approach of having several XML Signatures (or one XML Signature with several references) to protect the critical parts of a SOAP message only. The big question that arises from this approach is: what are the critical parts of a SOAP message?

In order to determine an answer to this basic question, it becomes necessary to investigate the threat model discussed earlier, and to correlate it to the typical structure of today's SOAP messages.

A. WS-AddressingSpoofing and Similar Threats

Today's most common pattern for applying XML Signature to SOAP messages actually consists in applying it to the whole SOAP body only. This way, all functional data within the body is integrity-protected, and new SOAP headers can be added without invalidating the signature. If necessary, authenticity can be ensured from that XML Signature as well, thus providing a sound basis for access control decisions.

However, latest advancements in the way SOAP message headers are used for additional functionality render this approach to be vulnerable to a list of security-related issues. An example would be the WS-AddressingSpoofing attack [13]. Here, the attacker modifies a SOAP request message (with signed body) by adding a WS-Addressing header that contains a Web Service endpoint of the attacker's choice as `ReplyTo` address. As a result, the Web Service invoked will perform its operation, then deliver the SOAP reply message to the Web Service endpoint denoted in the malicious `ReplyTo` element. This way, the attacker can intercept all reply messages, even if he only has access to the request messages—an obvious security breach.

The tricky part about this class of attack patterns is that they rely on adding new functionalities to SOAP messages that were not anticipated (or not even known about) at the message originator's side. Nevertheless the receiving Web Service server framework knows how to interpret that new functionality, and performs its task accordingly. Thus, the message originator has no possibility to prevent such kinds of attacks by protecting additional headers by means of XML Signature. There is no way to sign the *absence* of a WS-Addressing header (or any other token from any other WS-* specification) apart from protecting the whole SOAP header block with an XML Signature. Protecting the whole SOAP Header quarries issues concerning flexibility and convenience. Legitimate intermediaries can not alter the SOAP header any longer and thus are not able to introduce further means of protection (e.g. additional signatures, encryption blocks) or additional header data.

```
<SupportedSpecs soap:mustUnderstand="true"
   xmlns="http://...specNegotiation/"
   xmlns:soap="http://...soap-envelope">
  <SpecificationSuite
    URI="http://...specNeg.../XML+NS+SOAP+WSDLSuite" />
  <Specification
    URI="http://.../XPath" Version="1.0" />
  <Specification
    URI="http://.../WS-Addressing" Version="1.0" />
  <Specification
    URI="http://.../WS-Security" Version="1.1" />
  <Specification
    URI="http://.../SAML" Version="2.0" />
</SupportedSpecs>
```

Fig. 2. Example of a specification negotiation SOAP header

Hence, in order to cope with such threats, one viable approach consists in the following: a message originator explicitly negotiates a list of tokens that he added to the SOAP header himself or expects to be added by the legitimate intermediaries (for an example see Figure 2). This list of headers can be subsequently protected by an XML Signature, along with all headers that are present at message creation time. Note that this approach does not prevent an attacker from adding new fields, but it enables the message recipient to detect such modifications. This way, a SOAP message recipient can determine which of the SOAP headers contained in the message have been added regularly, and which may be part of a WS-AddressingSpoofing attack or similar. Additionally, the recipient knows which tokens have been present on message creation (as these are part of the XML Signature as well), which were expected to be applied by intermediaries, and which happened to be added without the message originator's expectation. Accordingly, the message recipient can decide by himself on whether he wants to process the unexpected headers or not.

The general approach of adding such creation-time message metadata as separate SOAP header is not new (cf. [14], [15]), however, it is crucial to note that the approach proposed here consists in specifying the set of *specifications* understood and used by a message's originator rather than fixing certain *structural parameters* of the XML elements forming a SOAP message. Hence, a `ReplyTo` header that may be added to a SOAP message without violating the fixed structure metadata will nevertheless rely on the use of the WS-Addressing specification. If this specification is not contained in the given list, the bogus header is not processed.

B. Multiple References

Another security-related problem in terms of XML Signature application consists in the decision on whether to use n single XML Signatures, one signature for each of the n references, one single XML Signature with n references, or anything in mixture. At first glance, it seems unimportant which of these approaches is used, as all of them result in that the critical parts are protected by a digital signature in the end.

However, in terms of security considerations for Web Service communication this difference becomes of critical importance.

Consider the scenario of a SOAP request message that contains a SOAP operation request as SOAP body and a timestamp token as SOAP header. The use of the timestamp token is to guarantee *freshness*, i.e. to ensure that the message expires after a certain amount of time. Typically, such an approach is used to counter replay attacks [16] that consist in copying and re-sending the very same SOAP message again in order to trigger the same Web Service operation execution more often. By adding a timestamp and a nonce value, the Web Service server can detect whether the message already expired or whether the nonce value has been already used within the validity timespan.

As both SOAP body and timestamp header must be considered as critical tokens, each of them has to be integrity-protected. Hence, the Web Service developer has two options: either each token gets its very own XML Signature header with a single reference, or the SOAP message will contain one XML Signature with two references. Investigating the former approach more deeply, incorporating the threat model of Section II-D, that approach turns out to contain a major security risk. If the timestamp is protected by an XML Signature of its own, a Dolev-Yao attacker can copy the whole timestamp token *and* its XML Signature, and can move it to another SOAP message (either one intercepted or one created by himself). This way, he can re-use the intercepted timestamp token for other SOAP requests, e.g. against other Web Service endpoints that require appropriately signed timestamp tokens. As other Web Services will not know that the nonce has already been used, they will only see a valid timestamp token with a valid XML Signature applied. Thus, the attacker manages to bypass replay attack countermeasures using stolen timestamp tokens.

In the other case where a single XML Signature protects both, timestamp token and SOAP body at once, the attacker can only re-use the timestamp token if he also copies the SOAP body at the same time. Though this is not impossible (cf. *XML Signature Wrapping attacks* [17], [18]), it poses severe restrictions to an attacker for properly crafting an attack message that can make use of the XML Signature over timestamp *and* SOAP body. Hence, having one XML Signature with n references tends to be the preferable approach.

On the other hand, consider the scenario of a SOAP request message containing a transaction request, e.g. advising a stock order. Part of the critical data within the SOAP body is a credit card information block. However, the SOAP body as a whole must also be considered as a critical token of its own. Here again, one has to investigate several approaches. The first and most obvious approach consists in applying a single XML Signature with a single reference to the SOAP body. This way, both the SOAP body and the contained credit card data are integrity-protected. The issue with this approach is that the credit card data might become part of another SOAP request that is forwarded to another Web Service endpoint. If the XML Signature covers the whole SOAP body only, it gets useless for the purpose of protecting and authenticating the credit card information in the second SOAP request as well. The same applies to the approach of having a single XML Signature with two references, one to the SOAP body and an additonal one to the credit card information block. But if the message creator applied two separate XML Signatures, one for the SOAP body and a separate one for the credit card information block, the latter one can be re-used in the second SOAP message to enable the credit card provider to validate both integrity and authenticity of the credit card information. Hence, the approach of separate XML Signatures here provides a slight advantage compared to the other two approaches.

C. XML Signature Best Practices

Resuming both cases, the proper application of XML Signatures largely depends on the application scenario. It is necessary to determine the pathways that critical data items have to pass, and to adapt XML Signatures in a way that their reuse is supported if it is legitimate, but that it is prevented for unauthorized purposes. In doubt, the safer option is to have a single XML Signature with n references to all critical tokens of a SOAP message.

IV. The Interrelation of XML Signatures and XML Encryption

Another field of interest in terms of proper application of XML Signatures consists in its correlation to the use of the XML Encryption specification [6] for enforcing data confidentiality in SOAP messages. The main issue here is that XML Encryption changes the actual XML contents, thus it is likely to invalidate XML Signatures if applied naïvely.

A first major stumbling stone to bypass when dealing with XML Signature and XML Encryption at once reveals to be the order of their application. At first glance, there is no difference between the two approaches (sign-then-encrypt and encrypt-then-sign). In both cases, the critical contents are both confidential and integrity-protected (and also authenticatable). However, some eminent differences have to be taken into consideration.

A. Aimed Security Goals

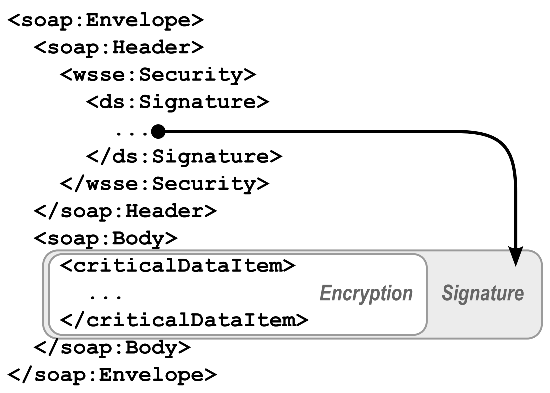
Before we discuss the different approaches for securing messages it is indispensable to define the desireable security goals.

Confidentiality. The security goal of confidentiality confirms that only authorized people are able to access or disclose concerned data. Usually this security goal is achieved by applying encryption algorithms on data worthy of protection.

Integrity. Integrity ensures that data is not being altered. In practical environments, hash values or HMACs are used to guarantee data integrity.

Authenticity. Trustworthiness of entities is addressed by authenticity. In typical environments one may use digital signatures for ensuring compliance with this goal.

Non-repudiation. This term ensures validity of statements issued by an individual in a way that it is not possible for an



```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <ds:Signature>
        ...
      </ds:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <criticalDataItem>
      ...
    </criticalDataItem>
  </soap:Body>
</soap:Envelope>
```

Fig. 3. Example of an encrypted-then-signed SOAP message

entity to repudiate the issuance. As a cryptographic primitive achieving this goal one may consider the usage of digital signatures and timestamps.

B. *Sign-then-Encrypt*

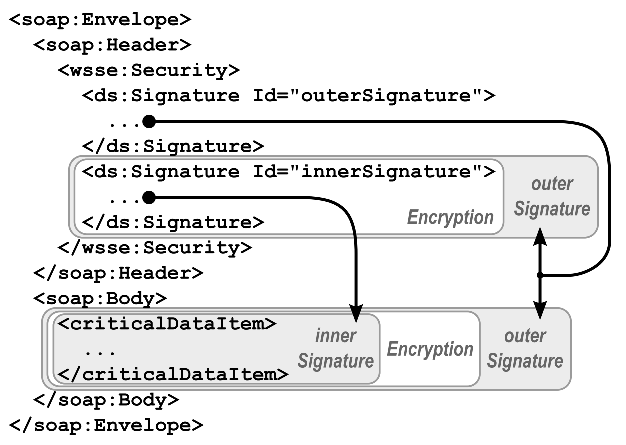
Considering the setting of sign-then-encrypt according to the WS-Security specification, a first critical issue in terms of security consists of the XML Signature metadata block of the SOAP security header. If the critical contents are signed then encrypted, the XML Signature nevertheless provides the attacker some information on the encrypted contents: the XML Signature discloses their hash value. Though it is a cryptographically hard problem to reverse a hash value back to the original contents, it nevertheless can be used for breaking the required security property of confidentiality: identical contents have the same hash value. This way, an attacker is able to determine whether a message's critical contents are identical to those of another message he has seen before. For instance, an attacker might be able to determine a certain limited set of typical messages with identical hash values, and correlate them to a certain type of operation invocation. Hence, he knows the Web Service operation that is to be performed even though all indicators of the operation (here the SOAP body contents) have been encrypted for confidentiality reasons.

A common countermeasure to this kind of threat is to encrypt the XML Signature metadata as well so that the attacker cannot learn from it. Nevertheless, the Web Service developer has to be aware of this fact and take it into account.

C. *Encrypt-then-Sign*

If the XML Signature is to be applied to the encrypted contents instead of the original contents, a first and obvious problem consists in the proper element addressing. If the critical contents to be signed happen to be a subtree of the contents to be encrypted, and if in that scenario (cf. Figure 3) the encryption takes place first, the XML Signature application implementation will not be able to find the referenced contents due to the fact of being encrypted.

Nevertheless, the calculation of XML Signatures over encrypted contents raises issues. First of all, the signer might not even know about the contents that were signed, thus



```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <ds:Signature Id="outerSignature">
        ...
      </ds:Signature>
      <ds:Signature Id="innerSignature">
        ...
      </ds:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <criticalDataItem>
      ...
    </criticalDataItem>
  </soap:Body>
</soap:Envelope>
```

Fig. 4. Example of a signed-then-encrypted-then-signed SOAP message

the property of *non-repudiation* is no longer as strong as a digital signature applied to plain data. The signer may claim to have had no clue about the real contents when applying his signature, so the legitimation properties of a digital signature may get weakened. All in all, this approach somewhat violates the paradigm of *see-what-is-signed* [3, Section 8.1.3], and thus should be prevented.

However, the application of XML Signatures to encrypted XML contents may also provide some advantages. Taking a closer look, the implications of an XML Signature covering an encrypted XML fragment is a claim that the Signer in the moment of XML Signature application has seen the encrypted contents. This implies that the encryption of these contents can not have happened after XML Signature application. For the threat model of Section II-D this means that an attacker is no longer capable of replacing the whole SOAP message body with some contents he encrypted himself for the Web Service server endpoint. The XML Signature here does not protect the critical contents themselves, but merely provides a guarantee that the encryption happened *prior* to XML Signature application. This also gives a proof to the SOAP message recipient that the SOAP message has been encrypted when leaving the message originator.

D. *Sign-then-Encrypt-then-Sign*

Taking these implications as input, the best approach for the interrelation of XML Encryption and XML Signature might be the following (cf. Figure 4 and [19]):

If a message originator is about to send a SOAP message containing critical tokens in terms of confidentiality and integrity to a SOAP recipient, he applies an XML Signature, then an XML Encryption, then again an XML Signature. The reasons for this are the following.

The first, "inner" XML Signature provides all security properties of an XML Signature. It ensures the original data's integrity and provides both authenticity and non-repudiation in the sense intended by the XML Signature application.

The following XML Encryption ensures confidentiality and must cover both the critical contents *and* the XML Signature metadata (of course, only of the inner XML Signature). Thus, it provides the same properties as in all other scenarios.

The last, "outer" XML Signature has the sole purpose of proving to the SOAP message recipient that the message originator himself applied the XML Encryption to the critical contents. This way, both the sender and the recipient of the SOAP message can be sure that the message encryption was applied correctly while the message was in transfer. Hence, even if the attacker is capable of crafting malicious requests with signed tokens using techniques as described above, and even if he is capable of encrypting those contents for the target of the Web Service server correctly, he nevertheless will ultimately fail to create the outer XML Signature, rendering all his efforts unsuccessful.

Depending on the application scenario, this sign-then-encrypt-then-sign approach might even be enhanced by having inner and outer signature become two references of the very same XML Signature. The concept introduces a chained process of four steps resulting in an highly interlocked signature. The necessary steps are discussed below:

1) Compute a hash value over the data that should be protected and store it temporarily
2) Encrypt the already processed block
3) Hash again the relevant block (which is now encrypted)
4) Take both pre-computed hash values and sign them

It is crucial to be aware of the fact that the used encryption algorithm must not use static initialization vectors due to the fact that an attacker may correlate again the given output of the encryption (which would be always the same in case of static IVs and the same message) as described in subsection Sign-then-Encrypt.

Processing as described above offers a major advantage that has to be taken into account: It becomes obvious that the overall task of signing, encrypting, and signing again was performed in one processing step, thus originating all at the very same entity. There is no way for an attacker to remove or forge any of the signatures, nor copy them to other requests in a reasonable scenario. Each removal/modification will immediately invalidate one of the two references, hence the overall XML Signature.

The downside of this approach is that there is no way to "recycle" any of the XML Signatures as described in Section III-B. Additionally, the recipient must take care for properly processing the XML Signature in correct interrelation to decryption, which is not covered by today's versions of the standards.

V. Conclusion and Future Work

We have shown that minimal differences in the way XML Signatures are used can cause large differences in the resulting security implications. Based on this observation, we have illustrated some basic best practice rules to consider when applying XML Signatures to SOAP messages according to the WS-Security specification. Additionally, we investigated the interplay of XML Signatures and XML Encryption, again discussing impacts and best practices.

Future work will consist in elaborating a complete guidance ruleset for all major Web Services specifications to clarify how they relate to XML Signature and what approach is best to be used in order to enforce security for them in a best-possible manner. Additionally, a proof-of-concept implementation of the `SupportedSpecs` approach of Figure 2 is work in progress.

References

[1] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[2] M. Jensen, L. Liao, and J. Schwenk, "The curse of namespaces in the domain of xml signature," in *SWS '09: Proceedings of the 2009 ACM workshop on Secure web services*. New York, NY, USA: ACM, 2009, pp. 29–36.

[3] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML-Signature Syntax and Processing (Second Edition)*, W3C Recommendation, Jun. 2008.

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

[5] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, *SOAP Version 1.2 Part 1: Messaging Framework*, W3C Recommendation, Jun. 2003. [Online]. Available: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[6] T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing*, W3C Recommendation, Dec. 2002. [Online]. Available: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[7] M. Gudgin, M. Hadley, and T. Rogers, "Web Services Addressing 1.0 - SOAP Binding," *W3C Recommendation*, May 2006.

[8] M. Elkins, D. D. Torto, R. Levien, and T. Roessler, *MIME Security with OpenPGP*, IETF RFC 3156, Aug. 2001.

[9] *ITU-T Recommendation X.509, Version 3 (1997)*, ITU-T Information Technology - Open Systems Interconnection - The Directory Authentication Framework, ISO/IEC 9594-8:1997, 1997.

[10] B. Ramsdell (Editor), *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1*, IETF RFC 3851, Jul. 2004.

[11] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol, version 1.1," RFC 4346. http://www.ietf.org/rfc/rfc4346.txt, 2006.

[12] D. Dolev and A. C.-C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[13] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on web services," *Computer Science - R&D*, vol. 24, no. 4, pp. 185–197, 2009.

[14] M. A. Rahaman, A. Schaad, and M. Rits, "Towards secure SOAP message exchange in a soa," in *Workshop on Secure Web Services*, 2006.

[15] M. A. Rahaman and A. Schaad, "SOAP-based secure conversation and collaboration," in *IEEE International Conference on Web Services (ICWS 2007)*. IEEE CS, 2007, pp. 471–480.

[16] P. Syverson, "A taxonomy of replay attacks," in *Proceedings of the 7th Computer Security Foundations Workshop (CSFW)*, 1994.

[17] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," in *SWS '05*. New York, NY, USA: ACM Press, 2005, pp. 20–27.

[18] S. Gajek, M. Jensen, L. Liao, and J. Schwenk, "Analysis of signature wrapping attacks and countermeasures," in *IEEE International Conference on Web Services (ICWS 2009)*. IEEE CS, Jul. 2009.

[19] D. Davis, "Defective sign & encrypt in s/mime, pkcs#7, moss, pem, pgp, and xml," *http://world.std.com/∼dtd/sign_encrypt/sign_encrypt7.html*, 2001.