Web Services Security: a preliminary study using Casper and FDR

E. Kleiner and A.W. Roscoe*
Oxford University Computing Laboratory

June 7, 2004

Abstract

Web Services is an important new XML-based architecture in which security is increasingly important. The WS-Security specification defines mechanisms for securing the SOAP messages. We show how those messages can be mapped to Casper notation and therefore be analysed with FDR. We show two attacks on proposed protocols and lastly discuss informally some ramifications of the use of the WS-Security specification.

1 Introduction

Web Services is an XML-based architecture that has been developed in order to make the coupling between distributed components looser. In the last few years, with the growth of the popularity and importance of the Web Services architecture, more and more standards have been defined for extending the functionality and for dealing with different concerns. Due to its growing importance, Web Services requires rigorous security.

A common way for achieving it is relying on a secure transport layer, typically SSL as was studied and analysed in [3]. Apart from the fact that this technique provides security only in a secure channel (and not in files or databases), it does not correspond with the WS architecture in which the intermediaries can manipulate the message on its way. Once using a secure transport layer intermediaries are not able to control the messages.

A more suitable way is using the WS-Security specification [1] that by using the XML-signature [7] and XML-encryption [8] specifications, deals with and defines standards and ways of securing SOAP messages [6] without relying on a secure transport layer. In effect it creates a new sphere for cryptographic protocols in terms of design and implementation.

The theoretical community has been very successful in the last decade in developing methods for analysing cryptographic protocols. One of these, based upon Hoare's CSP [10] is Casper [15], supported by the FDR refinement checker [17]. This approach has proved to be very successful for modelling security protocols: firstly in finding attacks (e.g. [14] and [16]) and more recently providing general proofs (e.g. [4].) This paper

*{Eldar.Kleiner, Bill.Roscoe}@comlab.ox.ac.uk

reports the preliminary results of an exercise in applying Casper to check Web services protocols secured by the WS-Security specifications. The only comparable work we are aware of is that of [2].

Our paper is structured as follows. In Section 2 we give short overview of Web Services. In Section 3 we indicate how the syntax of a SOAP message that uses the WS-Security may be transformed into Casper input. In Section 4 we show some results on the examples from [13], in particular, demonstrating two related attacks. In section 5 we discuss some reflections of using WS-Security. Finally we conclude and give the outline of our planned work in this area.

We recognise that our work is at an early stage, but given our success (see 4) in finding attacks on a proposed standard protocols, we feel it is appropriate to publish this version.

2 Web Services Background

2.1 Motivation

In IT today one often faces the need to integrate different computing systems within an organization, perhaps even running on different platforms, for consolidated decision making or central monitoring.

The integration task is frequently challenging, largely due to the fact that traditional application are statically bound, namely the parameters, objects' types and the programming language are agreed upon during their individual designs. *Service Oriented Architecture* is a concept which addresses this issue.

2.2 SOA - Service Oriented Architecture

> "SOA is an approach to build distributed systems that deliver application functionality as services to end user applications or to build other services"(IBM).

A service is a package of functions the do not depend on the context or state of other services. Each service can publish its functionality, while other services are capable of discovering and binding dynamically to this functionality.

Web Services adopt the SOA concept using XML-based message layer (called SOAP) and can use any transport layer such as HTTP [18] and SMTP [12]. The main significance of Web Services technology is that is has been embraced by the entire industry.

2.3 SOAP - Simple Object Access Protocol

SOAP was proposed originally by Microsoft and DevelopMentor to provide a way to package information using XML for exchange between different computing systems. Today it is a W3C recommendation.

A SOAP message consists of two main parts:

- `Header` is an open element for extra application requirement. It is used by other specifications for expanding SOAP. For example, it can contain information about routing, context or security.

- `Body` contains the main applicative payload and can also include an optional fault element.

In addition, the SOAP specification defines how to exercise intermediaries and RPC conventions for allowing a client to call a remote function using the SOAP mechanism.

3 Modelling WS-Security protocols

In common with most modern work on cryptographic protocols, we study the WSS ones using the Dolev-Yao model [9]. In this model the network is controlled by the intruder who has the following abilities when attacking a set T of trusted agents. (i) overhearing all the messages flowing through the network, (ii) intercepting messages, (iii) faking messages based on what he knows limited only by cryptography, and (iv) behaving as any agent outside of T. This model is particularly appropriate in the Internet context of Web Services, since there is no sense in which users have any control over the medium which connects them. We claim that in this model the syntax of the SOAP message has relatively[1] little effect on the security of the protocol and therefore an abstracted view of the protocol (in the way we will present here) taken that it encapsulates all the security elements, provides an accurate model.

We construct a mapping ϕ from SOAP messages to Casper input, such that if a WS-security protocol contains the messages $m_1, m_2..., m_n$ then,

1. If an attack is found on $\phi(m_1), \phi(m_2), \ldots, \phi(m_n)$ then a corresponding attack can be reproduced on $m_1, m_2, \ldots, m_n$.

2. If an attack exists on $m_1, m_2, \ldots, m_n$ then it also exists on $\phi(m_1), \phi(m_2), \ldots, \phi(m_n)$

We suggest that if the intruder possesses $\phi(m)$, then he can create (perhaps with some reasonable guesses) a message m' which has the same security behaviour as m. Furthermore, the security checks performed during the execution of the original protocol are equivalent to those in the abstracted protocol.

The first property can be achieved by defining ϕ^{-1}. In case an attack is found in the model then ϕ^{-1} can be used to map it back to the original protocol space. We indicate briefly how ϕ^{-1} can be constructed.

More important of the above properties is (2), since we definitely do not want to generate a false "proof" of correctness using the translation.

3.1 Constructing ϕ

In this preliminary study we address relatively simple WS-Security protocols; in particular we will assume for now that the messages contain only a security header and a body. Furthermore, we will also assume that encryption can be performed only in the body. In future work we will extend ϕ such that it will support more complex messages.

For clarity, we will use an example taken from [13] to demonstrate the way ϕ works. For convenience, we have removed the `Namespaces` from the message (creates no ambiguity since we are dealing here only with security related information items). In addition,

[1] An interesting way in which SOAP can help will be discussed later, but in fact this just increases the faithfulness of the translation described below.

since the `TimeStamp` element is ignored in the proposed protocol we have also removed it. The following message is the first of a protocol. We will see the rest of the protocol, and its purpose, later.

In the following the values `BV1`, ..., `BV6` denote boolean strings holding data (signatures, encryptions, etc).

```
<Envelope>
  <Header>
    <Security mustUnderstand="1">
      <BinarySecurityToken ValueType="x509v3" Id="myCert"> BV1
      </BinarySecurityToken>
      <Signature>
        <SignedInfo>
          <CanonicalizationMethod Algorithm=.... />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig\#rsa-sha1"/>
          <Reference URI="#body">
            <Transforms>
              <Transform Algorithm=.... />
            </Transforms>
            <DigestMethod Algorithm=... />
            <DigestValue> BV2 </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue> BV3 </SignatureValue>
        <KeyInfo>
          <SecurityTokenReference>
            <Reference URI="#myCert" />
          </SecurityTokenReference>
        </KeyInfo>
      </Signature>
      <EncryptedKey>
        <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <KeyInfo>
          <SecurityTokenReference>
            <KeyIdentifier ValueType="X509v3"> BV4
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue> BV5 </CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#enc" />
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </Header>
  <Body  Id="body">
    <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#content">
      <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue> BV6 </CipherValue>
```

```
      </CipherData>
    </EncryptedData>
  </Body>
</Envelope>
```

Message: M

The syntax and content of message M are built from BV1-BV6 using the structuring mechanisms of XML complying to the Web Services Security specification.

The crucial observation we make is that anyone who possesses the values BV1...BV6 in fact possesses the entire message from the point of view of security, since all of the structuring mechanisms and other values such as URL's are essentially public and guessable. In particular an intruder who possesses these values can create the whole message. We conclude that, from the point of view of maintaining security, it is only these values that can matter.

Furthermore, careful inspection reveals that BV1 and BV4 are not really secret at all and we would expect any attacker to possess them since they are public certificates. So in fact the only "novel" parts of M are (BV2, BV3, BV5, BV6). The rest of the syntax simply puts them in context and indicates what they are.

What we will therefore do is demonstrate a mapping ϕ which reduces M to a representation of this quadruple, presenting them in the form used by Casper.

We now define the function ϕ. However the definition of ϕ we present in this paper does not cover all structures defined by [7, 1, 8] (for example we do not deal here with all the Security and KeyInfo optional children defined by [7]). Nevertheless, our definition is adequate for the protocols presented in this paper. We will provide a complete definition in a later paper.

We adopt the following general principles:

- ϕ can be applied to a valid, parsed XML document (we use the XML infoset recommendation [5])
- ϕ is a mapping that works on an element information item as a function of its children.

Note that ϕ might ignore children when they are not important security-wise.

Since ϕ cannot comprehend some of the elements' content (e.g. binary data) it uses the function ψ that "knows" the protocol specification to retrieve the necessary information that ϕ needs. The automated version of ϕ will have to get as an input some information about the protocol specification so that ψ will be able to work as described later. In addition, since ϕ might return different outputs for the same input depending on its context, ψ is also used to retrieve the context. More details are given later in this paper.

3.1.1 Envelope element

The result of applying ϕ to the Envelope element is the Envelope element's children list where ϕ is applied to every element in the list. In the Example.

$$\phi(M) = \phi(\langle Header\rangle ... \langle /Header\rangle), \phi(\langle Body\rangle ... \langle /Body\rangle).$$

The inverse of ϕ in this case is obvious. The original message can be constructed by putting the elements from the sequence one after the other inside an envelope. Note that different messages can be semantically equivalent because the order of the element does not necessarily change the semantic of the message. One can observe that the abstracted messages will be semantically equivalent as well.

3.1.2 `Header` element

In a similar way ϕ creates a sequence of the children of the `Header` element.

$$\phi(\langle Header\rangle ... \langle /Header\rangle) = \phi(Child_1), \phi(Child_2), ..., \phi(Child_n)$$

3.1.3 `Security` element

ϕ is defined very similarly when it is applied to the `Security` element, except for a slight change which is swapping the signature and the encryption elements. The reason for this swap is that [1] specifies that if the designer of the message wants to sign an element after encrypting it, the `Signature` element should be followed by the `Encryption` element and vice versa (i.e. the order of the operation is right to left.) Since ϕ works from left to right we need to swap their places to preserve the semantics.

$$\phi(\langle Security\rangle ... \langle /Security\rangle) = \phi(\langle BinarySecurityToken\rangle ... \langle /BinarySecurityToken\rangle),$$
$$\phi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle), \phi(\langle Signature\rangle ... \langle /Signature\rangle)$$

Again, the inverse of ϕ can be constructed similarly to the way it was constructed in 3.1.1.

3.1.4 `BinarySecurityToken` element

The `BinarySecurityToken` contains binary data. The `ValueType` attribute indicates what is encoded (e.g. X.509 certificate or Kerberos ticket.) In the example the `BinarySecurityToken` contains X.509 certificate. In this case, since the certificate is public, there is no need to model it. Therefore, the ϕ function will not return a value in this case.

When the X.509 `BinarySecurityToken` is referred to by another element, ϕ returns the public or the secret key corresponding to the certificate.

$$\phi(Ref) = \{A, PK(A)\}_{SK(CA)}$$
$$\phi(Ref, ENC) = PK(A)$$
$$\phi(Ref, SIG) = SK(A)$$

Here, *Ref* is the value of the `Id` attribute of the `BinarySecurityToken` and $A = \psi(\langle BinarySecurityToken...\rangle ... \langle /BinarySecurityToken\rangle)$.

ψ is responsible for returning the identity of the holder of the certificate. When automating ϕ and ψ, ψ will not be designed to extrapolate the identity of the certificate holder out of the binary data. Instead will create (with user assistance if necessary) a symbolic name for a typical agent in this role of the protocol.

3.1.5 `Signature` element

The ϕ function creates an abstracted signature of the `Signature` element in the following way.

$$\phi(\langle Signature\rangle ... \langle /Signature\rangle) = \{\phi(\langle Reference ...\rangle ... \langle /Reference\rangle),$$
$$\phi(\langle Reference ...\rangle ... \langle /Reference\rangle) ...\}_{\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, SIG)}$$

3.1.6 `Reference` element

We are distinguishing between two cases. The first is when the `Reference` is obliged to contain the `DigestMethod` element (when it appears in the signature element by [7])

$$\phi(\langle Reference\ \ URI = "Ref"\rangle ... \langle /Reference\rangle) = \phi(DigestMethod)(\phi(Ref))$$

In the rest of the cases,

$$\phi(\langle Reference\ \ URI = "Ref"/\rangle) = \phi(Ref)$$
$$\phi(\langle Reference\ \ URI = "Ref"/\rangle, ENC) = \phi(Ref, ENC)$$
$$\phi(\langle Reference\ \ URI = "Ref"/\rangle, SIG) = \phi(Ref, SIG)$$

3.1.7 `DigestMethod` element

In this case ϕ returns the name of the abstracted hash function. See [15] for details of abstracting hash functions. In our example:

$$\phi(\langle DigestMethod\ \ algorithm = \text{“}http://www.w3.org/2000/09/xmldsig\#sha1"/\rangle) = sha1$$

3.1.8 `KeyInfo` element

The ϕ function returns the abstracted key that the `KeyInfo` represents. Here are two examples of possible scenarios:

$\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, T) = \phi(\langle SecurityTokenReference\rangle ... \langle /SecurityTokenReference\rangle, T)$
or
$\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, T) = \phi(\langle KeyName\rangle ... \langle /KeyName\rangle, T)$

3.1.9 `SecurityTokenReference` element

The `SecurityTokenReference` provides a way for referencing a security token by direct reference, key identifier, key names and embedded references. ϕ works as follows:

$$\phi(\langle SecurityTokenReference\rangle ... \langle /SecurityTokenReference\rangle, T) = \phi(Child, T)$$

3.1.10 `KeyName` element

`KeyName` contains a string that uniquely defines a cryptographic key. ϕ returns an abstracted version of the key that the `KeyName` element stands for. Once ϕ is automated, the user will have to supply the abstracted key. For example, if the `KeyName` contains a “distinguished name” of an X.509 certificate, the user will have to supply the identity of the certificate holder.

- If the string defines a symmetric key, then

$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, T) = K$$

  Where $K = \psi(\langle KeyName\rangle ... \langle /KeyName\rangle)$

- If the string defines an asymmetric key then,

$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, SIG) = SK(A)$$
$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, ENC) = PK(A)$$

  Where $A = \psi(\langle KeyName\rangle ... \langle /KeyName\rangle)$ is the identity of the certificate holder.

3.1.11 `KeyIdentifier` element

The `KeyIdentifier` is a value that uniquely identifies a security element (cryptographic key in our example). The value type and the way to process it can vary and is defined by the designer and not by [1]. In our example the `KeyIdentifier` contains an X.509 certificate of the responder.

$$\phi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle, SIG) = SK(A)$$
$$\phi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle, ENC) = PK(A)$$

where $A = \psi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle)$ is the identity of the certificate holder.

3.1.12 `ReferenceList` element

ϕ is defined over `ReferenceList` element in the following way,

$$\phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle) = \phi(\langle DataReference .../\rangle) ... \phi(\langle DataReference .../\rangle)$$
$$\phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle, A) = \phi(\langle DataReference .../\rangle, A) ... \phi(\langle DataReference .../\rangle, A)$$

3.1.13 `DataReference` element

When ϕ gets the `DataReference` element as an input, it does not return a value, instead it creates a context for ϕ that can be retrieved later by ψ (see 3.1.14 for example.)

$$\phi(\langle DataReference\ \ URI = Ref/\rangle) = Context(Ref)$$
$$\phi(\langle DataReference\ \ URI = Ref/\rangle, A) = Context(Ref, A)$$

where *Ref* is the context name and A is the set of values of the context.

3.1.14 `EncryptedKey` element

The `EncryptedKey` element provides a way to encrypt a symmetric key with the recipient's public key. Once again, when automating the mapping, the user will need to supply the abstraction of the key that this element holds. We will mark this key K. In this case,

$$\phi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle) = \phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle, \{K\}),$$
$$\{K\}_{\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, ENC)}$$

where $K = \psi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle)$.

3.1.15 `EncryptedData` element

The value of ϕ of the `EncryptedData` element depends on the context of ϕ. The context is retrieved using ψ and affect ϕ in the following way:

- If $\psi(Ref)$ is not defined (in case the context Ref does not exist), then $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle)$ is also not defined.
- If $\psi(Ref)$ is defined and the `EncryptedData` element has a `KeyInfo` element child then, $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle) = \phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, ENC)$
- If $\psi(Ref)$ is defined and the EncryptedData element does not have a `KeyInfo` element child then, $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle) = A$ where A is the set of values of the context.

3.1.16 `Body` element

The body is abstracted in the following way:

- If the `Body` element has no `EncryptedData` element child then, $\phi(Body) = Body$
- If ϕ of the `EncryptedData` element is defined and it is the only child of the `Body` element then, $\phi(Body) = \{Body\}\{\phi(\langle EncryptedData...\rangle ... \langle /EncryptedData\rangle)\}$
- If ϕ of the `EncryptedData` element is not defined and it is the only child of the `Body` element then, $\phi(Body) = Body$
- If `EncryptedData` is one of the elements in the body then the designer will have to supply the applicative structure of the body and only the relevant part will be dealt with. More details will be given in future work.

3.2 Example

We will now demonstrate the complete derivation of $\phi(M)$.

$\phi(M)$
$\Rightarrow \phi(\langle$Header$\rangle ... \langle /$Header$\rangle)$,$\phi(\langle$Body$\rangle ... \langle /$Body$\rangle)$

$\Rightarrow \phi(\langle$Security$\rangle ... \langle /$Security$\rangle)$,$\phi(\langle$Body$\rangle ... \langle /$Body$\rangle)$

$\Rightarrow \phi(\langle$BinarySecurityToken$\rangle ... \langle /$BinarySecurityToken$\rangle)$, $\phi(\langle$EncryptedKey$\rangle ... \langle /$EncryptedKey$\rangle)$, $\phi(\langle$Signature$\rangle ... \langle /$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle /$Body$\rangle)$

$\Rightarrow \phi(\langle$EncryptedKey$\rangle ... \langle /$EncryptedKey$\rangle)$, $\phi(\langle$Signature$\rangle ... \langle /$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle /$Body$\rangle)$

$\Rightarrow$ According to the protocol specification [13], ψ of the EncryptedKey element is a random generated symmetric key. We will give it the conventional name K. $\Rightarrow$
$\phi(\langle$ReferenceList$\rangle ... \langle/$ReferenceList$\rangle$,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ $\phi(\langle$DataReference URI=#enc $/\rangle$,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$, $\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{SecurityTokenReference} \rangle ... \langle/\text{SecurityTokenReference}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{KeyIdentifier} \rangle ... \langle/\text{KeyIdentifier}\rangle,\text{ENC})}$, $\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ According to the designer's specification, ψ of the KeyIdentifier element is the receiver's X.509 certificate. We will give the receiver the conventional name B. $\Rightarrow$
Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\phi(\langle$Reference URI=#body$\rangle ... \langle/$Reference$\rangle)\}$,
$\{\phi(\langle$KeyInfo$\rangle ... \langle/$KeyInfo$\rangle)$,SIG$\}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\phi(\langle$DigestMethod.../$\rangle)(\phi(\text{body}))\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\phi(\text{body}))\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$,
$\{\text{sha1}(\{\text{Body}\}_{\phi(\langle \text{EncryptedData Id=“enc”} \rangle ... \langle/\text{EncryptedData}\rangle)})\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$
Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{SecurityTokenReference} \rangle ... \langle/\text{SecurityTokenReference}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{Reference URI=\#myCert.../} \rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ The Reference element points to the BinarySecurityToken element. According to [13], the holder of this certificate is the sender. We will give him the conventional name A$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{SK(A)}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{SK(A)}$, $\{\text{Body}\}_K$

$\Rightarrow \{K\}_{PK(B)}, \{sha1(\{Body\}_K)\}_{SK(A)}, \{Body\}_K$

4 Results of analyzing example protocols

We now quote message `M'`, taken from [13] with slight changes: namely, we removed the Namespaces and the `TimeStamp` element since it was ignored.

```
<Envelope>
  <Header>
    <Security mustUnderstand="1">
      <Signature>
        <SignedInfo>
          <CanonicalizationMethod Algorithm=.... />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig\#rsa-sha1"/>
          <Reference URI="#body">
            <Transforms>
              <Transform Algorithm=.... />
            </Transforms>
            <DigestMethod Algorithm=... />
            <DigestValue>... </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>...</SignatureValue>
        <KeyInfo>
          <SecurityTokenReference>
            <KeyIdentifier ValueType=X509v3>...
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
      </Signature>
      <BinarySecurityToken ValueType="x509v3" Id="myCert"> ...
      </BinarySecurityToken>
      <EncryptedKey>
        <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <KeyInfo>
          <SecurityTokenReference>
            <Reference URI="#myCert" />
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue>...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#enc" />
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </Header>
  <Body  Id="body">
    <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#content">
```

```
      <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue>...</CipherValue>
      </CipherData>
    </EncryptedData>
  </Body>
</Envelope>
                                    Message: M'
```

In scenario #6 in [13] the following protocol is proposed.

1. A→B: `M`
2. B→A: `M'`

After applying ϕ to both of the messages we get the following protocol.

1. MSG 1 .A→B : $\{\mathrm{K}\}_{\mathrm{PK(B)}}$, $\{\mathrm{sha1}(\{\mathrm{Body}\}_{\mathrm{K}})\}_{\mathrm{SK(A)}}$, $\{\mathrm{Body}\}_{\mathrm{K}}$
2. MSG 2. B→A : $\{\mathrm{K2}\}_{\mathrm{PK(A)}}$, $\{\mathrm{sha1}(\{\mathrm{Body2}\}_{\mathrm{K2}})\}_{\mathrm{SK(B)}}$, $\{\mathrm{Body2}\}_{\mathrm{K2}}$

Here A is an initiator who seeks to transfer some applicative data in Body to B and receive a response in Body2. A sends his certificate, a key K that he freshly generated signed with B's public key, the Body encrypted with K and a signature of the encrypted Body. When B receives the message he checks the certificate in order to make sure that A is an authorized user, he decrypts the second element using his private key to get K, ensures that the signature is correct and then use K to decrypt the last element to get Body. B then applicatively creates Body2 and sends it back in MSG 2.
The author of [13] claim that the response (MSG 2) is authenticated.[2]
We checked this protocol with the following Casper authentication specification:

$$Agreement(B, A, [Body2])$$

This specifies that if A thinks he has successfully completed a run of the protocol with B, then B has previously been running the protocol, apparently with A, and B was the one who sent *Body2* to A. Using FDR the following authentication attack was found.

1. MSG 1. I → Bob : $\{\mathrm{K}\}_{\mathrm{PK(Bob)}}$, $\{\mathrm{sha1}(\{\mathrm{Body}\}_{\mathrm{K}})\}_{\mathrm{SK(I)}}$, $\{\mathrm{Body}\}_{\mathrm{K}}$
2. MSG 2. Bob → I : $\{\mathrm{K2}\}_{\mathrm{PK(I)}}$, $\{\mathrm{sha1}(\{\mathrm{Body2}\}_{\mathrm{K2}})\}_{\mathrm{SK(Bob)}}$, $\{\mathrm{Body2}\}_{\mathrm{K2}}$
3. MSG 1. Alice → I_{Bob} : $\{\mathrm{K3}\}_{\mathrm{PK(Bob)}}$, $\{\mathrm{sha1}(\{\mathrm{Body3}\}_{\mathrm{K3}})\}_{\mathrm{SK(Alice)}}$, $\{\mathrm{Body3}\}_{\mathrm{K3}}$
4. MSG 2. I_{Bob} → Alice : $\{\mathrm{K2}\}_{\mathrm{PK(Alice)}}$, $\{\mathrm{sha1}(\{\mathrm{Body2}\}_{\mathrm{K2}})\}_{\mathrm{SK(Bob)}}$, $\{\mathrm{Body2}\}_{\mathrm{K2}}$

On receiving the final MSG 2 (step 4), Alice believes she has completed a run of the protocol with Bob. The truth is that Bob was never actually running the protocol with her. The Intruder successfully impersonated him: there is therefore no reasonable sense in which Bob's response to Alice is authenticated.

This attack relies on a previous run of the protocol (steps 1,2) which may take place long before steps 3,4. The intruder uses an old MSG 2 to attack the protocol in the

[2]The author emphasizes that the scenarios in the paper have not been extensively vetted for attacks.

second run by encrypting K2 with the public key of Alice, replacing the certificate to the one of Alice and sending it in step 4 as a message from Bob. An inverse attack in which the intruder is the initiator of the protocol in step 3 is also feasible.

We also analysed scenario #7 in [13] in which FDR found a similar attack. The protocol after applying ϕ looks like this:

1. MSG 1. A→B : $\{\text{sha1}(\{\text{A,PK(A)}\}_{\text{SK(CA)}}), \text{sha1(Body)}\}_{\text{SK(A)}}, \{\text{K}\}_{\text{PK(B)}}, \{\text{Body}\}_{\text{K}}$
2. MSG 2. B→A : $\{\text{sha1(Body2)}\}_{\text{SK(B)}}, \{\text{K2}\}_{\text{PK(A)}}, \{\text{Body2}\}_{\text{K2}}$

We again used Casper to create the CSP model of the proposed protocol. FDR then found a similar attack:

1. MSG 1. I→Bob : $\{\text{sha1}(\{\text{I,PK(I)}\}_{\text{SK(CA)}}), \text{sha1(Body1)}\}_{\text{SK(I)}}, \{\text{K1}\}_{\text{PK(Bob)}}, \{\text{Body1}\}_{\text{K1}}$
2. MSG 2. Bob→I : $\{\text{sha1(Body2)}\}_{\text{SK(Bob)}}, \{\text{K2}\}_{\text{PK(I)}}, \{\text{Body2}\}_{\text{K2}}$
3. MSG 1. Alice→I_{Bob} : $\{\text{sha1}(\{\text{Alice,PK(Alice)}\}_{\text{SK(CA)}}), \text{sha1(Body3)}\}_{\text{SK(Alice)}}, \{\text{K3}\}_{\text{PK(Bob)}}, \{\text{Body3}\}_{\text{K3}}$
4. MSG 2. I_{Bob}→Alice : $\{\text{sha1(Body2)}\}_{\text{SK(Bob)}}, \{\text{K2}\}_{\text{PK(Alice)}}, \{\text{Body2}\}_{\text{K2}}$

The vulnerability of both the protocols is caused by the fact that neither of the messages in the first protocol and the second message in the second protocol are bound correctly to the sender. The intruder can use this fact for re-sending those messages, pretending they came from him. A simple solution to correcting the flaw is adding the sender identity to the signature (his certificate for example.) This prevents the above attacks because then the intruder will not be able to produce a valid MSG 2 (in step 4) since Alice will be expecting it to contain Bob's identity signed in the message.

We analysed the fixed protocols and FDR then failed to find an attack. However they are safe only for a single run. When Bob receives a message from Alice he still has no guarantee that the message was recently sent by her. The protocol can be strengthened further by using nonces or timestamps to prevent the intruder re-sending messages.

Those attacks are analogous to many found on protocols outside the WebServices area. For example, the attack on the Needham-Schroeder Public Key protocol described in [14] takes advantage of a similar flaw (the flaw in this protocol is that the identity of the responder is not bound to the message.)

5 Reflections on WS-Security

It appears that at present SOAP Message Security is used for the purpose of setting the parts of messages which convey actual security in context, namely allowing the receiver to see details of what the bit strings constituting signatures, encryptions, hashes etc are meant to be. We have seen that because this formatting is public it is as easily created by an intruder.

We indicated earlier that there is an interesting way in which SOAP can assist security. This is because there are two examples of possible forms of attack that the Web Services Security mechanism provides extra strength against, both of which have been studied in the literature of cryptographic protocols.

- *Type-flaw* attacks, where an intruder persuades a legitimate user that a value is of a different type to what it really is, and exploits this in an attack. So for example the intruder might persuade agent A to sign a value which A thinks is a nonce or a hash value, but which in fact has other structures that the intruder can then use to impersonate A in some run with (say) B. The avoidance of this type of problem (see [11]) for example, depends on the nature of the cryptography involved, how it is used – for example whenever one signs something one adds the signer's interpretation of what it is signing – and protocol design. In XML-signature, the tag name of the element that is signed provides information about the type of the element. Similar things are true about values that are encrypted or hashed. Therefore, SOAP messages that are protected by the Web Services specification are relatively resistant to type-flaw attacks, with two caveats:
  - In order for the above mechanism to work, the signed/hashed/encrypted object must contain a sufficient description of the type so as not to leave any ambiguity.
  - This mechanism only prevents type confusion by trusted agents of data which is received by them protected by a cryptographic device they can understand.
- *Protocol interference*, where information gleaned from running one protocol can be used to attack another. To avoid this type of attack the best approach is to use the principle of *explicitness* within the encryptions and signed data, and ensure hash functions are context dependent. This would mean that instead of encrypting, signing or hashing X, one does the respective thing to (T, X) where the tag T includes information such as which protocol this was done for, and which field in what message. By signing/encrypting/hashing a value in a SOAP context we have the machinery here to create excellent *de facto* tags.

We emphasise that the SOAP mechanism do not in general prevent these forms of attack, but they do provide some protection implicitly. We hope it might be possible to strengthen the SOAP standard so as to guarantee absence of these forms of attack. This will be subject of further work.

6 Conclusion

We have demonstrated that the security of SOAP based protocols is essentially the same problem as the traditional form of cryptographic protocol. We have demonstrated the correspondence for a few protocols and presented preliminary work on the general mapping.

The main advantages of this translation are firstly clarity in the sense that the abstract descriptions are far more concise, and secondly the availability of tools such as Casper to analyse the WS-Security protocols.

The usefulness of our method was immediately shown by the attacks we discovered. It seems clear to us that our methods or something very similar need to be adopted in the Web Services community when security is a concern.

We note that because our methods give a translation to Casper notation where the whole area of protocol verification is well understood, they do not suffer from the restriction to only authentication specifications reported in [2].

7 Future work

In the future we will present the complete formalised mapping between the Web Services notation and Casper input.

We expect that most or all of this process can be automated, so that in effect one can simply input a WS-Security protocol to an extended Casper and obtain a model that we can input to FDR to test its security.

As far as we know, *intermediaries* is a poorly understood area in Web Services Security. We are interested in "internalising" potential intermediaries in the style of [4] and believe we then be able to model and check protocols with arbitrary number of intermediaries.

Of course the question remains of how we can draw inferences about general implementations from checks of small examples. The fact that we have translated WS-Security into equivalent Casper input means we can expect the body results that already exists for standard protocols to apply here as well.

Lastly, we would like to look into SAML and extend ϕ such that we will be able to reason about protocols that contain messages with SAML assertions.

Acknowledgements

We would like to thank Philippa Hopcroft and Gavin Lows for their suggestions and Hal Lockhart for his help and the information he shared with us, in particular [13].

References

[1] A. Nadalin, C. Kaler, P. Hallam-Baker and R. Monzillo. "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)". Oasis, 2004. http://www.oasis-open.org/committees/download.php/5941/oasis-200401-wss-soap-message-security-1.0.pdf.

[2] Karthikeyan Bhargavan, Cedric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 198–209. ACM Press, 2004.

[3] P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *the 16th IEEE Computer Security Foundations Workshop, pages 141-154, IEEE Computer Society Press*, 2003.

[4] P.J Broadfoot and A.W. Roscoe. Internalising agents in CSP protocol models. Workshop on Issues in the Theory of Security (WITS '02), Protland Oregon, USA, 2002.

[5] J. Cowan and R. Tobin. "XML Information Set". W3C Recommendation, 2001. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen S. Thatte and D. Winer. "Simple Object Access Protocol (SOAP) 1.1". W3C Note, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[7] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon. "XML-Signature Syntax and Processing". W3C Recommendation, 2002. http://www.w3.org/TR/xmldsig-core/.

[8] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway and E. Simon. "XML Encryption Syntax and Processing". W3C Recommendation, 2001. http://www.w3.org/TR/xmlenc-core/.

[9] D. Dolev and A.C. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.

[10] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[11] J. Heather, G. Lowe and S. Schneider. How to prevent Type Flaw Attacks on Security Protocols. *Proceeding of 13th IEEE Computer Security Foundations Workshop*, pages 255–268, 2000.

[12] J. Klensin . "Simple Mail Transfer Protocol". IETF, 2001. http://www.ietf.org/rfc/rfc2821.txt.

[13] H. Lockhart. "Web Services Security: Interop 2 Scenarios Working Draft 06". Oasis, 2003. http://lists.oasis-open.org/archives/wss/200310/pdf00000.pdf.

[14] G. Lowe. An attack on the Needham-Schroeder Public-Key authentication protocol. *Information Processing Letters*, 1995.

[15] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[16] G. Lowe and A.W. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE transactions on Sofware Engineering*, 23(10):659–669, 1997.

[17] Formal Systems (Europe) LTD. *Failure-Divergences Refinement FDR2 Manual*, 1997.

[18] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk and T. Berners-Lee. "RFC 2616: Hypertext Transfer Protocol – HTTP/1.1". IETF, 1997. http://www.ietf.org/rfc/rfc2616.txt.