On the relationship between Web Services Security and traditional protocols

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

June 30, 2005

Abstract

XML and Web Services security specifications define elements to incorporate security tokens within a SOAP message. We propose a method for mapping such messages to an abstract syntax in the style of Dolev-Yao, and in particular Casper notation. We show that this translation preserves flaws and attacks. Therefore we provide a way for all the methods, and specifically Casper and FDR, that have been developed in the last decade by the theoretical community for the analysis of cryptographic protocols to be used for analysing WS-Security protocols. Finally, we demonstrate how this technique can be used to prove properties and discover attacks upon a proposed Microsoft WS-SecureConversation protocol.

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 and the uses to which it is put, Web Services requires rigorous security.

A common way of achieving security is relying on a secure transport layer, typically SSL as was studied and analysed in [7]. 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 [4] that by using the XML-signature [10] and XML-encryption [11] specifications, deals with and defines standards and ways of securing SOAP messages [9] without relying on a secure transport layer. In effect it creates a new sphere for cryptographic protocols in terms of design and implementation.

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

In [14] we claimed that in the Dolev-Yao [12] model the syntax of the SOAP message has relatively[1] little effect on the security of the protocol and therefore that an abstracted view of the protocol, taken that it encapsulates all the security elements, provides an accurate model. We suggested a mapping function ϕ (see Appendix I of the full paper [15] for details)[2] from SOAP messages to Casper [20] 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 demonstrated how, using property (1) of ϕ, we could use Casper [20] and the FDR refinement checker [22] to find two attacks on WSS protocols proposed by Oasis in [19].

In this paper we prove property (2) of ϕ and confirm our last claim. We then demonstrate how we can use this property together with the *Data Independence* and *internalising* techniques of [8] to provide general proofs of Web Services Security protocols as well as finding vulnerabilities.

The contribution we make is a formal translation of WS-Security protocols to traditional cryptographic protocols, ensuring that flaws and attacks are preserved. Therefore we provide a way for all the methods, and specifically Casper and FDR, that have been developed in the last decade by the theoretical community for the analysis of cryptographic protocols to be used for analysing WS-Security. We also prove (Lemma 2) that, in the case of any addition of deductive rules to the traditional model for encapsulating additional capabilities of the intruder, ϕ continues to satisfy property (2). Finally, we create a framework in which the addition of rules for manipulating XML elements can be proven to not detract from the correctness of property (2) of ϕ (see section 5 for example). To the best of our knowledge, ours is the first work proving the relationship between WS-Security protocols and traditional security protocols.

Our paper is structured as follows. In Section 2 we give a short overview of the CSP model of traditional security protocols. In Section 3 we indicate how the traditional model can be extended to encapsulate WS-Security protocols. In Section 4 we prove that ϕ's property (2) holds. In Section 5 we show how to produce a general proof of correctness of WS-Security protocol using our technique and in particular we prove some properties of a Microsoft protocol based on the WS-SecureConversation [2] specification, we then reveal some flaws letting an intruder exercise different sort of attacks. Finally we conclude and give the outline of our planned future work in this area.

This paper is an outline of [15], where many details omitted here can be found.

2 Modelling traditional security protocols in CSP

In this section we describe how a security protocol is modelled using CSP and how the model allows us to reason about it.

[1] We showed a case in which SOAP can help, but in fact this just increases the faithfulness of the translation.

[2] The definition of ϕ in the full paper is more elaborated than the one in the original paper and it aims to support any WSS protocol

2.1 The *Message* datatype

The datatype *Message* represents the messages exchanged between the different agents. It is based on a set atoms called *Atom* where the sets *Key* (contains all the cryptographic keys), *Nonce*, *Text* and *Password* are defined to be subsets of the atom set ($Key \subseteq Atom$, $Nonce \subseteq Atom$ and $Password \subseteq Atom$). In addition, we define *HashFn* to be the set that contains all the available cryptographic hash functions. The datatype *Message* is composed of sequencing, referencing, encrypting and hashing the atomic value in *Atom* and defined by:

$$\begin{array}{lll} Message & ::= & \textsc{Encrypt}.Key.Message \mid \\ & & \textsc{Hash}.HashFn.Message \mid \\ & & \textsc{Reference}.Key \mid \\ & & \textsc{Sq}.Message^{*} \mid \\ & & \textsc{Atom}.Atom \end{array}$$

In this paper we will use the Casper notation of writing $\{\mathrm{m}\}_k$ for $\textsc{Encrypt}$.k.m. We will use g(|m|) for $\textsc{Hash}$.g.m, R(K) for modelling a reference to a key K and finally will abbreviate $\textsc{Sq}.\langle\, \mathrm{m}_1,\ldots,\mathrm{m}_n\rangle$ to $\langle\, \mathrm{m}_1,\ldots,\mathrm{m}_n\rangle$. For example, we will denote the construct $\textsc{Encrypt}.\mathrm{k}.(\textsc{Sq}.\langle \mathrm{a},\mathrm{n_a}\rangle)$ by $\{\mathrm{a},\mathrm{n_a}\}_{\mathrm{k}}$.

2.2 Trustworthy agents

Every agent taking a part in the protocol is modelled as a CSP process (An agent can also be internalised in the intruder deduction set [8] but for now we will assume that all the honest agents are implemented as CSP processes.) We define the process P_A denoting agent A, using the following events:

- *send*.A.B.M - symbolises agent A sending message M to agent B.
- *receive*.A.B.M - symbolises agent B receiving message M apparently from A.

In addition we define the following events for delineating specifications for the protocol we want to analyse. See [20] for more details about how these events are used to express properties of security protocols.

- *claimSecret*.A.B.M symbolises that A thinks that message M is a secret shared only with agent B.
- *running*.A.B.$\{\mathrm{M}_1,\ldots,\mathrm{M}_n\}$ symbolises that A thinks he started a new run of the protocol with B where $\{\mathrm{M}_1,\ldots,\mathrm{M}_n\}$ represent some details of this run.
- *finish*.A.B.$\{\mathrm{M}_1,\ldots,\mathrm{M}_n\}$ symbolises that A thinks he has just finished a run of the protocol with B where $\{\mathrm{M}_1,\ldots,\mathrm{M}_n\}$ represent some details of this run.

For more information regarding the translation of a protocol description to a CSP representation see [25].

2.3 Modelling the intruder and putting the network together

Based on the Dolev-Yao model [12] we allow the intruder to have 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 would any agent outside of T. We first define the rules that allow the intruder to construct new messages. The definition is based on the relation $\vdash$ which characterises deduction rules by which the intruder can deduce new messages. We say that $B \vdash M$ if message M can be deduced from the set of messages B.

member $B \in M \Rightarrow B \vdash M$

sequencing $B \vdash \{M_1, \ldots, M_n\} \Rightarrow \langle M_1, \ldots, M_n \rangle$

splitting $B \vdash \langle \ldots, M, \ldots \rangle \Rightarrow B \vdash M$

encrypting $B \vdash M \wedge B \vdash \textsc{Atom}\ K \wedge K \in Key \Rightarrow B \vdash \{M\}_K$

decrypting $B \vdash \{M\}_K \wedge B \vdash \textsc{Atom}\ K^{-1} \Rightarrow B \vdash M$

hashing $B \vdash M \wedge g \in \mathit{HashFn} \Rightarrow B \vdash g(|M|)$

referencing $B \vdash K \wedge K \in Key \Rightarrow B \vdash R(K)$

Informally, the intruder can form an encryption when he knows the message and the key. He can decipher an encryption for which he knows the inverse of the key, create a reference to a key that he knows, hash every message he knows and can both break up and form sequences.

Since by the Dolev-Yao model the intruder should be able to overhear, intercept and block each message, the intruder process also models the communication medium.

The process representing the intruder is parameterised by X, which ranges over subsets of *Message*, and represents all the facts the intruder has learned. In this model the intruder gets every message sent by the honest agents or by the server via the *send* channel. He then can pass it to the agents via the appropriate *receive* channel unless he decides to block it or fake a new message instead.

$$
\begin{aligned}
Intruder(X) \mathrel{\widehat{=}}\ & \Box_{M \in Message}\, send?A?B!M \rightarrow Intruder(X \cup \{M\}) \\
& \Box \\
& \Box_{M \in Message, X \vdash M}\, receive?A?B!M \rightarrow Intruder(X) \\
& \Box \\
& \Box_{M \in Message, X \vdash M}\, leak.M \rightarrow Intruder(X)
\end{aligned}
$$

The initial state of the intruder is $Intruder(IIK)$ where *IIK* is the *Intruder Initial Knowledge*. The complete system is then[3]:

[3] For clarity this model is abstracted. In the model generated by Casper each fact is modelled as a process parallelled with the entire fact space. This technique reduced dramatically the state space that FDR needs to explore (see [25] for more details).

$$SYSTEM \mathrel{\widehat{=}} (|||_{A \in Honest} P_A) \parallel INTRUDER(IIK)$$

2.4 Specifying Protocol requirements

The requirements of the protocols are encapsulated by *trace specifications.*

Secrecy As mentioned before, when agent A performs the event $claimSecret.A.B.Secs$ it means that we believe, at this point in the protocol run, that the values in the set $Secs$ are secret and shared only with agent B. It expresses the expectation that the intruder cannot be in possession of values from $Secs$, i.e. the intruder should not be able to perform *leak*.M where M$\in$ $Secs$.

Authentication We first introduce the *finish* and *running* events (see [25] for more details)[4]. The *finish* event is performed by the honest agents when they complete a protocol run and the *running* event should be performed before the last *send* event.

We will use the following definition [21] which is one of the more common forms of authentication:

> If A thinks he has completed a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, and both agents agreed as to which roles they took, and both agreed as to the values of the variables $v_1, \ldots, v_n$, and there is a one-one relationship between the runs of B and the runs of A.

The following specification corresponds to this definition[5]:

$$\begin{array}{l} Agreement_{AgreementSet}(tr) \mathrel{\widehat{=}} \\ \quad \forall A \in Agent; B \in Honest; Ms \in AgreementSet \bullet \\ \quad\quad tr \downarrow finished.A.B.Ms \leq tr \downarrow running.B.A.Ms \end{array}$$

3 Creating a CSP model of WSS Protocols

In order to analyse Web Services Security protocols we have to extend the model that was described in Section 2. We will later use the new extended WSS model to prove Property (2) of ϕ.

The WSS model is similar to the one in Section 2. There are essentially has differences. Firstly, we need to replace the *Message* datatype with a new datatype which encapsulates SOAP messages. Secondly, a new set of deductions must be added to the intruder's deduction set to enable him to "understand" the XML tagging system underlying the SOAP messages.

[4] notice that [25] refers to these events as *Running* and *Commit*

[5] The binary operator $\downarrow$ (tr$\downarrow$ e) represents the number of occurrences of event e in a trace tr

3.1 The *Envelope* datatype

The *Envelope* datatype represents the SOAP messages that are sent and received by the agents. Similarly to the *Message* datatype, *Envelope* will be based on the *Atom* set where *Key*$\subseteq$*Atom*, *Nonce*$\subseteq$*Atom*, *Agent*$\subseteq$*Atom*, *TimeStamp*$\subseteq$*Atom*, *Password*$\subseteq$*Atom* and on *HashFn*, the set that contains all the cryptographic hash functions. In addition, we define *HeaderApp* to be the set of all the applicative SOAP headers, *BodyApp* to be the set of the applicative sub-element of the `Body` element, *Id* is the set of all the unique ids and *Element* to enclose all the valid SOAP elements. Some sample clauses from the definition of *Envelope* follow. The complete definition of *Envelope* is presented in the full paper.

Envelope	::=	<Envelope.*Attributes.*>.*EnvelopeContent.*</Envelope>
EnvelopeContent	::=	*Header* | *Body* | $\langle$*Header*, *Body*$\rangle$ | $\langle$*Header*, *EncryptedData*$\rangle$

Header	::=	<Header.*Attributes.*>.*HeaderContent.*</Header>
HeaderAtom	::=	*Security* | *HeaderApp* | *EncryptedData*
HeaderContent	::=	Sq.*HeaderAtom**

Security	::=	<Security.*Attributes.*>.*SecurityTokens.*</Security>
SecurityToken	::=	*UsernameToken* | *BinarySecurityToken* | *Signature* | *ReferenceList* | *EncryptedKey* | *EncryptedData* | *SecurityTokenReference*
SecurityTokens	::=	Sq.*SecurityToken**

The *Envelope* datatype illustrates the SOAP messages almost down to the last detail. However, we make some abstractions in order to make it more concise, and clearer: details may be found in the full paper.

3.2 The intruder

The Intruder's definition in the WSS model is very similar to the one in the traditional model and is still based on the basic seven deduction rules presented in Section 2. The following is an some example of the added deductions (called WS*) that provide the intruder the ability to manipulate the SOAP's XML-based elements (for the complete set see the full paper [15]).

WSEncryptedKey1 $B \vdash$ <EncryptedKey.*Attributes.*>.
$\langle$<KeyInfo>.R(K_1).</KeyInfo>,
<CipherData>.$\{K_2\}_{K_1}$.</CipherData>,
Sq.*Element**
$\rangle$.
</EncryptedKey> $\Rightarrow$
$B \vdash \{K_2\}_{K_1} \wedge B \vdash R(K_1)$

The complete WSS system is then similar to the one presented in Section 2:

$$SYSTEM_{WSS} \mathrel{\widehat{=}} (|||_{A \in Honest} P_A) \parallel INTRUDER(IIK_{WSS})$$

We remark that our reason for building a full CSP model of WSS is primarily to allow the abstract analysis rather than to check it directly with FDR.

4 Proving property (2) of ϕ

We can now give a formal definition of the function $\phi : Envelope \rightarrow Message$. We are aiming to use he result of [17]. Therefore ϕ is designed as close as possible to the idea of renaming transformation defined there.

Note that since the *Envelope* datatype definition uses the *Message* datatype definition, ϕ might be applied to *Message*. In that case according to the definition of ϕ (see full paper), ϕ behaves like the identical function, i.e. returns its input.

ϕ is defined over the events of $SYSTEM_{WSS}$ as follows:

$$\begin{array}{lcl}
\phi(send.A.B.Envelope) & \widehat{=} & send.A.B.\phi(Envelope) \\
\phi(receive.A.B.Envelope) & \widehat{=} & receive.A.B.\phi(Envelope) \\
\phi(claimSecret.A.B.Envelope) & \widehat{=} & claimSecret.A.B.\phi(Envelope) \\
\phi(running.A.B.Envelope) & \widehat{=} & running.A.B.\phi(Envelope) \\
\phi(finished.A.B.Envelope) & \widehat{=} & finished.A.B.\phi(Envelope)
\end{array}$$

and over traces by:

$$\phi(tr) \quad \widehat{=} \quad \langle \phi(e) \mid e \leftarrow tr \rangle$$

Applying ϕ to the trustworthy agent processes $(\phi(P_A) \mid \forall A \in Honest)$ is implicitly defined since in CSP, the *renamed* process $f(P)$ is the process that behaves like P except the fact that every event e in P is renamed to $f(e)$.

Finally, we apply ϕ to the whole system:

$$\phi(SYSTEM_{wss}) \mathrel{\widehat{=}} (|||_{A \in Honest} \phi(P_A)) \parallel INTRUDER(\phi(IIK_{wss}))$$

4.1 Fault-Preserving renaming transformations

In [17] Hui and Lowe prove that if *SYSTEM* is transformed into *SYSTEM′* in the following way:

$$SYSTEM' \mathrel{\widehat{=}} (|||_{A \in Honest} f(P_A)) \parallel INTRUDER(IIK')$$

where *IIK'* is the new initial knowledge that was given to the intruder in the transformed system, then f is fault-preserving if it satisfies the following two conditions:

1. $\forall B \in \mathbb{P}(Message); \mathrm{M} \in Message \bullet B \cup IIK \vdash \mathrm{M} \Rightarrow f(B) \cup IIK' \vdash f(\mathrm{M})$
2. $f(IIK) \subseteq IIK'$

Informally, if the intruder can deduce a message in the original *SYSTEM* he must be able to deduce the equivalent message $f(\mathrm{M})$ in the corresponding $SYSTEM'$, in this case we say that function f satisfies the *distribution property*. Secondly, the corresponding messages of all of the intruder's initial knowledge in the original *SYSTEM* must be possessed by the intruder in $SYSTEM'$.

The proof of this result does not rely on the structure of the *Message* datatype or the deduction set of the intruder. In other words, the result is valid regardless of the type of messages the agents exchange and of the deductions the intruder has. Thanks to this fact we can apply this result to the WSS model. Therefore, in order to prove property (2) of ϕ, it is enough to prove that:

1. $\forall B \in \mathbb{P}(Envelope); \mathrm{E} \in Envelope \bullet B \cup IIK_{wss} \vdash \mathrm{E} \Rightarrow \phi(B) \cup IIK_{wss} \vdash \phi(\mathrm{E})$
2. $\phi(IIK_{wss}) \subseteq \phi(IIK_{wss})$

The second condition is obviously satisfied. What remains to be proven is that ϕ satisfies the *distribution property* which we prove by structural induction over the deduction set of the intruder.

4.2 Proving ϕ satisfies the *distribution property*

For proving the *distribution property* we need the following Lemma taken from [17]

Lemma 1 $B \vdash \mathrm{E} \wedge B \subseteq B' \Rightarrow B' \vdash \mathrm{E}$
$B \vdash \mathrm{E} \wedge B \cup \{\mathrm{E}\} \vdash \mathrm{E}' \Rightarrow B \vdash \mathrm{E}'$.

We start by proving that ϕ satisfies the *distribution property* for all the deductions taken from the original system.

Lemma 2 *For every intruder's deduction taken from the traditional system, ϕ satisfies the distribution property.*

Proof: If $B \vdash \mathrm{M} \Rightarrow B \vdash \mathrm{M}'$ for every $\mathrm{M}, \mathrm{M}' \in Message$ and for every deduction rule from the traditional system, then by the inductive hypothesis, $B \cup IIK_{wss} \vdash \phi(\mathrm{M}) = \mathrm{M}$ and by the original deductive rule $\Rightarrow B \cup IIK_{wss} \vdash \mathrm{M}' = \phi(\mathrm{M}') \Rightarrow B \cup IIK_{wss} \vdash \phi(\mathrm{M}')$

□

We complete the proof by proving that ϕ satisfies the *distribution property* for the deductions added to WSS model. An example is given below, the rest of the clauses may be found in the complete paper.

WSEncryptedKey1 If $SE =$ <EncryptedKey>.
⟨<KeyInfo>.R(K_1).</KeyInfo>,
<CipherData>.$\{\mathrm{K}_2\}_{\mathrm{K}_1}$.</CipherData>,
$\textsc{Sq}.Element^*$
⟩.
</EncryptedKey>

and $B \vdash SE$, then by the inductive hypothesis, $\phi(B) \cup IIK_{wss} \vdash \phi(SE) =$ $\{\mathrm{K}_2\}_{\phi(\text{<KeyInfo>.R(K}_1\text{).</KeyInfo>,ENC})}$, $\phi($<KeyInfo>.R(K_1).</KeyInfo>$) = \{\mathrm{K}_2\}_{\mathrm{K}_1}$, R(K)= $\phi(\{\mathrm{K}_2\}_{\mathrm{K}_1})$,$\phi(\mathrm{R(K)}) \Rightarrow \phi(B) \cup IIK_{wss} \vdash \phi(\{\mathrm{K}_2\}_{\mathrm{K}_1}) \wedge \phi(B) \cup IIK_{wss} \vdash \phi(\mathrm{R(K)})$

4.3 Agreement authentication property

We now prove that property (2) of ϕ is valid with respect to the agreement property as well. In [17], it is proven that if the following condition is satisfied:

$$\forall Es \in AgreementSet; Es' \in Envelope \bullet Es \neq Es' \Rightarrow f(Es) \neq f(Es') \quad \text{then}$$

$$\neg Agreement_{AgreementSet}(tr) \Rightarrow \neg Agreement_{f(AgreementSet)}(f(tr))$$

Unfortunately, ϕ is not injective when applied to the `KeyInfo`, `EncryptedData` and `EncryptedKey` if we use the usual equality over the datatype *Envelope*. Moreover, `SecurityTokenReference` can often be replaced by the element it points to and by that to create two different SOAP messages in which their ϕ transformation is identical. For those reasons, it looks that ϕ does not satisfy the latter condition. It can however be shown that all equivalence classes of messages which map to the same image over ϕ are, so far as security it concerned, entirely equivalent and in particular the authentication properties carried by any one of them is carried by them all. So we can proceed as if this property did hold. See the full paper for more details.

4.4 Ramifications

We have established that ϕ is safe with respect to both secrecy and authentication under the WSS model. If one extends the WSS model by adding more deduction rules, then the proof can be extended as well (see Section 5).

Since ϕ transforms the *Envelope*s into *Message*s, we don't need the added deduction rules from section 3 for analysing a protocol after it was transformed by ϕ. Therefore a WSS protocol can be analysed in the traditional model after it was transformed by ϕ without the worry of generating false proof of correctness.

We would like to emphasise that although this proof is based on the theory of CSP, it is valid for any tool regardless of its underlying theory. Consequently, any established tool for analysing security protocols can use ϕ for analysing WSS protocols.

ϕ's input is the SOAP messages of the protocol to be analysed. This fact allows the non-specialist user to analyse complex WSS protocols in a few minutes. We have tested our automated version of ϕ on various WSS protocols had received satisfying results. We will introduce a new tool in the future for analysing WSS protocols either using Casper and FDR or (subject to adaptation) by other available tools.

As a result of Lemma 2 any extension of the intruder abilities made in the traditional protocol model can be used when analysing WSS protocols transformed by ϕ.

We saw in 4.3 that if there are two semantically equivalent but syntactically different WSS protocols then ϕ will transform them to the same traditional protocol. Due to this character of ϕ it was suggested that ϕ can be used for making the semantics of WSS clearer.

5 Proving the correctness of a WS-SecureConversation based protocol

In the previous sections we presented our method of analysing WS-Security Protocols. We have also implemented ϕ so that together with Casper and FDR it makes a tool for analysing WS-Security protocols. However, the framework that we created can be used for extending ϕ to encapsulate elements defined and by new Web Services specifications by local protocol designers.

In this section we give an example of extending ϕ to analyse a WS-SecureConversation based protocol taken from WSE [23] (See the full SOAP messages' description in the full paper). We demonstrate how the extension can be proven using the model we presented in Section 3 and provide a general proof of correctness of this protocol using Casper and FDR.

5.1 Background

The Web Services Security specification defines security tokens to provide secrecy, authentication, integrity and other security properties to the claims that these token encapsulate. Yet, the process of verifying these tokens against the security policy has to be repeated for each SOAP message. This is obviously problematic performance-wise, for example an `EncryptedKey` element might be used in each message instead of agreeing upon a session key once. WS-SecureConversation[2] addresses this issue. It builds upon WS-Security and WS-Trust [3] to allow a requester and a Web Service to set up a mutually authenticated *security context* that can be used by two parties as their session key or for deriving new keys. WS-Trust defines three different ways for establishing *security context*. In the end of each following procedure a `SecurityContextToken` element (SCT) is created and the new *security context* is associated with it.

1. **Security Context Token is created by a security token service (STS)** - The context initiator sends a *security token service* (STS) a `RequestSecurityToken` (RST) request, if the policy permits and the requester's requirements are met, a `RequestSecurityTokenResponse` (RSTR) is returned. The RSTR contains `RequestedSecurityToken` specifying a new SCT and a `RequestedProofToken` pointing to a "secret" to be assigned to the SCT. The newly created *security token* is distributed to other parties by the STS in a similar way (via RSTR).

2. **Security Context is created by one of the agents and propagated with a message** - If an entity is trusted by other parties then it can create an SCT and send a signed unsolicited RSTR to the other parties. Once again, the RSTR contains `RequestedSecurityToken` specifying a new SCT and a `RequestedProofToken` pointing to a "secret".

3. **Security Context created by negotiation** - WS-Trust defines ways for negotiating or exchanging sequence of messages on the contents of the SCT (e.g. the shared secret) when it is needed. The negotiation is done by the parties sending each other challenges before establishing the *Security context*. WS-Trust defines a variety of challenges for example one party might specify a fact and expect the other party to returned it signed.

After the *security context* establishment, the parties may use this shared secret with WS-Security for signing and encrypting messages. However, it is recommended by the WS-SecureConversation specification that derived keys will be used for these purposes.

In this case the DerivedKeyToken can be used for indicating which derivation is used within a given message.

5.2 Modelling WS-SecureConversation

In this section we show how to create the necessary extension to ϕ for analysing the WSE's WS-SecureConversation based protocol. For brevity we don't present here the complete extension for dealing with collections of tokens and negotiation although such an extension can made and proven. Below just two of the extensions are given, rest of them, and the proof that the extension keeps ϕ safe, can be found in the full paper.

5.2.1 RequestSecurityToken

The RequestSecurityToken element (RST) is defined in the WS-Trust specification and is used by the WS-SecureConversation specification for requesting *security context token* (SCT)[6]. The context attribute is an optional URI specifies a context for the request. All subsequent RSTR elements relating to this request must carry this attribute.

$$\phi(\langle \mathit{RequestSecurityToken\ Context} = \text{``}\mathit{Con}\text{''}\rangle ... \langle / \mathit{RequestSecurityToken}\rangle) = \phi(\langle \mathit{Base}\rangle ... \langle / \mathit{Base}\rangle), \phi(\langle \mathit{Supporting}\rangle ... \langle / \mathit{Supporting}\rangle), \phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle, \text{``}\mathit{Con}\text{''})$$

When LifeTime is a child of the RequestSecurityToken element, it is abstracted away since the responder's decision and she may ignore the LifeTime in the request.

5.2.2 Entropy

The Entropy element allows a requester and a responder to specify entropy that is to be used in creating a key.

- If BinarySecret is the child of Entropy then
  $\phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle) = K \qquad \phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle, \text{``}\mathit{Key}\text{''}) = K$
  Where $K = \psi(\langle \mathit{BinarySecret}\rangle ... \langle / \mathit{BinarySecret}\rangle)$ is an abstracted secret in BinarySecret.

- If EncryptedKey is the child of Entropy then
  $\phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle) = \phi(\langle \mathit{EncryptedKey}\rangle ... \langle / \mathit{EncryptedKey}\rangle)$
  $\phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle, \text{``}\mathit{Key}\text{''}) = \phi(\langle \mathit{EncryptedKey}\rangle ... \langle / \mathit{EncryptedKey}\rangle, \mathit{ENC})$

Finally if ϕ get the Entropy element and a context name as a parameter then ϕ creates a context that can be extracted later by related RSTRs using ψ:

$$\phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle, \text{``}\mathit{Con}\text{''}) = \phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle), \mathit{Context}(\text{``}\mathit{Con}\text{''}, \phi(\langle \mathit{Entropy}\rangle ... \langle / \mathit{Entropy}\rangle, \text{``}\mathit{Key}\text{''}))$$

[6]Since RequestSecurityToken can be used for requesting any *Security Token*, when used in the WS-SecureConversation context it should contain a TokenType element indicating that this is an SCT request.

5.3 Analysing the protocol

We applied ϕ to the WSS protocol (see Appendix III of the full paper) and obtained:

MSG 1. A$\rightarrow$B:	RST, UMI1, anonymous, B, ts1,
	{sha1(ts1), sha1(SecurityToken-b8...,$\{K_1\}_{PK(B)}$),
	sha1(RST), sha1(UMI1), sha1(anonymous), sha1(B)$\}_{p\text{-}sha1(pass(A),N_A+ts1)}$,
	sha1(password(A),N_A,ts1), N_A, ts1,$\{K_1\}_{PK(B)}$
MSG 2. B$\rightarrow$A:	RSTR, UMI2, UMI1, anonymous, ts2,
	{sha1(RSTR), sha1(UMI2), sha1(UMI1), sha1(anonymous),
	(sha1(ts2),sha1(uuid1, ts2', $\{K_2\}_{K_1}$)$\}_{SK(B)}$,uuid1, ts2', $\{K_2\}_{K_1}$
MSG 3. A$\rightarrow$B:	UMI3, anonymous, B, ts3, (uuid1, ts2'), {sha1(UMI3), sha1(anonymous),
	sha1(B), sha1(ts3), sha1(body1)$\}_{p\text{-}sha1(K_1,K_2)}$,$\{\text{body1}\}_{p\text{-}sha1(K_1,K_2)}$
MSG 4. B$\rightarrow$A:	UMI4, UMI3, A, ts4, (uuid1, ts2'),{sha1(UMI4), sha1(UMI3), sha1(anonymous),
	sha1(ts4),sha1(body2)$\}_{p\text{-}sha1(K_1,K_2)}$,$\{\text{body2}\}_{p\text{-}sha1(K_1,K_2)}$

After applying some *Simplifying Transformations*[17] for making it clearer we got the following protocol:

MSG 1. A$\rightarrow$B:	UMI1, {UMI1, B, $\{K_1\}_{PK(B)}$)$\}_{p\text{-}sha1(pass(A),N_A)}$,
	sha1(password(A),N_A)), N_A, $\{K_1\}_{PK(B)}$
MSG 2. B$\rightarrow$A:	{UMI1, UMI2, $\{K_2\}_{K_1}\}_{SK(B)}$,uuid1,$\{K_2\}_{K_1}$
MSG 3. A$\rightarrow$B:	{UMI3, B, body1$\}_{p\text{-}sha1(K_1,K_2)}$,$\{\text{body1}\}_{p\text{-}sha1(K_1,K_2)}$
MSG 4. B$\rightarrow$A:	{UMI3, UMI4, body2$\}_{p\text{-}sha1(K_1,K_2)}$,$\{\text{body2}\}_{p\text{-}sha1(K_1,K_2)}$

We analysed this protocol using the *Data Independence* technique for modelling infinite runs of the protocol with the following Casper specifications:

$$Secret(A, body1, [B]) \qquad Secret(A, pass(A), [B])$$
$$Secret(B, body2, [A]) \qquad Agreement(A, B, [body1, K_1])$$

These specify that agent A's password and the body elements sent by both parties are secret at the end of the protocol run. In addition, they specify that *body*1 and the client (A) entropy correctly authenticated to the server (B). We checked the data-independent model of this protocol on FDR and it failed to find any attacks. Since ϕ is safe, by the properties of the data-independent models we have therefore proved that this protocol is satisfies these specifications for arbitrarily many runs in an arbitrarily large implementation.

However, when analysing the protocol with the following Casper authentication specification an attack was found.

$$Agreement(B, A, [body2, K_2])$$

This specifies that if A thinks he has successfully completed a run of the protocol with B, then B has been previously running the protocol apparently with A and B was the one who sent K_2 and *body*2. The following attack is found by FDR:

MSG 1α.	$\mathrm{A} \rightarrow \mathrm{I_B}$	: UMI1, {UMI1, B, $\{\mathrm{K}_1\}_{PK(B)}$)}$_{p\text{-}sha1(pass(Alice),N_A)}$, sha1(password(Alice),$\mathrm{N}_A$)), N_A, $\{\mathrm{K}_1\}_{PK(B)}$
MSG 1β.	$\mathrm{I} \rightarrow \mathrm{B}$	: UMI1, {UMI1, B, $\{\mathrm{K}_1\}_{PK(B)}$)}$_{p\text{-}sha1(pass(I),N_A)}$, sha1(password(I),$\mathrm{N}_A$)), N_A, $\{\mathrm{K}_1\}_{PK(B)}$
MSG 2β.	$\mathrm{B} \rightarrow \mathrm{I}$	: {UMI1, UMI2, $\{\mathrm{K}_2\}_{K_1}\}_{SK(B)}$,uuid1,$\{\mathrm{K}_2\}_{K_1}$
MSG 2α.	$\mathrm{I_B} \rightarrow \mathrm{A}$	: {UMI1, UMI2, $\{\mathrm{K}_2\}_{K_1}\}_{SK(B)}$,uuid1,$\{\mathrm{K}_2\}_{K_1}$
MSG 3α.	$\mathrm{A} \rightarrow \mathrm{I_B}$	: {UMI3, B, body1}$_{p\text{-}sha1(K_1,K_2)}$,{body1}$_{p\text{-}sha1(K_1,K_2)}$
MSG 3β.	$\mathrm{I} \rightarrow \mathrm{B}$	: {UMI3, B, body1}$_{p\text{-}sha1(K_1,K_2)}$,{body1}$_{p\text{-}sha1(K_1,K_2)}$
MSG 4β.	$\mathrm{B} \rightarrow \mathrm{I}$	: {UMI3, UMI4, body2}$_{p\text{-}sha1(K_1,K_2)}$,{body2}$_{p\text{-}sha1(K_1,K_2)}$
MSG 4α.	$\mathrm{I_B} \rightarrow \mathrm{A}$	: {UMI3, UMI4, body2}$_{p\text{-}sha1(K_1,K_2)}$,{body2}$_{p\text{-}sha1(K_1,K_2)}$

At the end of the protocol run, A believes she has completed a run of the protocol with B using data items body2 and K_2 where clearly B wasn't aware she was trying to communicate with him. This attack is possible on the original protocol only if messages 3 and 4 are sent before the time stamps expire ($ts2', ts3, ts4$.)

The protocol in question binds messages to agents by by signing the WS-Addressing [1] elements. The problem is that the latter specification allows to identify endpoints as "anonymous" instead of giving them meaningful URIs. Since this is how the client (A) is identified by the WS-Addressing in this protocol, the second message is not bound properly to her. This flaw can be fixed, by including the client's identity in the signatures of message 2.

Still, even if instead of "anonymous" the correct identity of the client is signed, when analysing the WS-Trust of the protocol (First two messages) with the following Casper authentication specification an attack was found by FDR.

$$Agreement(A, B, [N_A, \mathit{UMI1}])$$

This specifies that if B thinks he has successfully completed a run of the WS-Trust part of the protocol with A, then A has been previously running the protocol apparently with B and A was the one who sent N_A and *UMI1*.

An intruder can intercept MSG 1 of a valid run of the protocol between the service and a honest agent. The intruder can then re-send the message large number of times to the service before the time stamp expires, making the service allocate enormous number of *security contexts*. Using this method the intruder accomplishes a denial-of-service attack forcing the service to allocate all his resources and not being able to serve other agents.

It should be pointed out that the WSE platform has a "detect replay" mechanism which prevents two RST messages from the same client. We were able to exercise this attack since this mechanism was set off as a default. We argue as follows that the solution to this flaw should be a correct authentication of N_A and $UMI1$ rather than mechanisms in the style of the detect replay mechanism.

1. Careless use of this protocol makes it prone to DOS attacks. An unexperienced user might not be aware of the implications of the detect replay mechanism, leave it off and make his implementation vulnerable.

2. In case of an implementation with two servers (B), if a client (A) sends MSG 1 to one of the servers the other server should be informed and reject any attempt of connection by the same client.

3. The use of the detect replay mechanism suggests that every valid agent can have only one session with the service. It may be the case that an implementor would want his service to open several sessions with the same client. If authentication of the RST was correct it would not be an issue.

4. The detect replay mechanism in the WSE framework only works for `UsernameToken` and will be extended to Kerberos ticket in the next version. In addition the WSE can be configured with a custom replay detection mechanisms. A correct protocol would work for any type of authentication without the need of tailoring the detect replay mechanism.

A close inspection reveals that the WS-SecureConversation part of the protocol (last two messages) suffers from the same flaw. The WS-SecureConversation was designed such that infinite messages can be exchanged after the *security context* establishment. In this case the same type of attack can be exercised and the detect replay mechanism cannot prevent it.

One possible solution for fixing the flaw without the detect replay mechanism is to bind the RST message to the client using *negotiation* in the following way:

MSG 1.	A→B:	Session Request
MSG 2.	B→A:	UMI0
MSG 3.	A→B:	$\{$UMI0, UMI1, B, $\{K_1\}_{PK(B)})\}_{p\text{-}sha1(pass(A),N_A)}$, sha1(password(A),$N_A$), N_A, $\{K_1\}_{PK(B)}$
MSG 4.	B→A:	$\{$UMI1, UMI2, A, $\{K_2\}_{K_1}\}_{SK(B)}$,uuid1,$\{K_2\}_{K_1}$
MSG 5.	A→B:	$\{$UMI2, UMI3, B, body1$\}_{p\text{-}sha1(K_1,K_2)}$,$\{$body1$\}_{p\text{-}sha1(K_1,K_2)}$
MSG 6.	B→A:	$\{$UMI3, UMI4, body2$\}_{p\text{-}sha1(K_1,K_2)}$,$\{$body2$\}_{p\text{-}sha1(K_1,K_2)}$

Alternatively a third message can be added in the original protocol, in which the client (A) signs UMI2.

5.4 Reflections

It looks like it is impossible to authenticate correctly the RST when trying to stick to the two step protocol run as is suggested in the WS-Trust specification. This may be the reason that the WSE team developed the detect replay mechanism, believing that it is more important to stick to the specification than having the protocol fixed in a conventional way.

The latter example suggests that WS-Trust should be mended or at least a cautionary note ought to be added indicating the possible vulnerability of the two step *Security Context* establishment.

A close look at the former attack indicates that perhaps it is due to incorrect implementation rather than mistaken design. The designers understood the importance of binding and therefore the WS-Addressing fields were signed. Ever since the flawed implementation of APM [26] of a protocol proven correct by Paulson [24], one of the goals of the formal verification community was to make it possible to analyse the implementation rather than the design. This work was the first step of achieving it in the Web Services world.

6 Related Work

Gordon and Pucella [16] proposed a security abstraction to RPC services in which requests and responses are encoded as SOAP messages. This abstraction is modelled using an object calculus which its semantics is defined by pi-calculus. This approach is currently limited to checking authentication properties.

Bhargavan, Fournet, Gordon and Pucella developed a tool (TulaFale [18]) based on the Blanchet's `ProVerif` [5]. The tool compiles a description of SOAP-based security protocol and its properties into the pi-calculus and then runs `ProVerif` to analyse it. At present it relies on a hand-coded description of a protocol: there is no front end comparable to our function ϕ and its automation. They choose to verify web service protocols at what it effectively a lower level, with their models being implemented at a level of abstraction something like our *Envelope* type.

In [6] TulaFale specification language was extended for modelling WS-Trust and WS-SecureConversation based protocols. The authors point out measures to be taken for correctly secure such protocols, some are similar to the flaws found in this paper (i.e. binding properly the messages to the sender and detect replay messages.) Using our technique however, it was possible to find the subtle flaw of binding the message using the WS-Addressing elements and automatically detect the "replay" attack.

Damiani *et al* [13] propose an access control model for flexible access to SOAP based services, relying on a secure channel such as TLS/SSL.

7 Conclusion

That part of SOAP Message Security which lies outside of any cryptographic operators may be constructed at will by any user, trustworthy or malicious. There is nothing secret about it. So its purpose must be 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.

Since *Kerckhoff's Known Design Principle* is adopted by most if not all crypto-analysts, the extra information about the structure of the messages given by the XML-tagging is assumed anyway in abstract models used for the analysis of security protocols. In other words the tagging provides an implementation of what this style of work assumes, usually implicitly. What we have provided is an automated way of moving from the specific XML tagging structure to familiar abstract protocols in a way such that attacks can be translated to and forth.

We have already shown in [14] that there are some interesting ways in which SOAP can assist security by providing degree of protection against type flaw attacks. In future work we hope to provide a more precise characterisation of this.

References

[1] "Web Services Addressing (WS-Addressing)". W3C, 2004. http://www.w3.org/Submission/ws-addressing/.

[2] "Web Services Secure Conversation (WS-SecureConversaion)", 2005. ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf.

[3] "Web Services Trust Language (WS-Trust)", 2005. ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf.

[4] 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.

[5] B. Blanchet. An Efficient Cryptographic protocol verifier based prolog. *14th IEEE Computer Security Foundations Workshop (CSFW 14)*, 2001.

[6] K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon. Secure sessions for web services. In *ACM Workshop on Secure Web Services (SWS)*, page to appear, Fairfax, Virginia, Oct 2004. ACM Press, New York.

[7] 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.

[8] 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.

[9] 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.

[10] 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/.

[11] 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/.

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

[13] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi and P. Samarati. Securing SOAP E-Services. *International Journal of Information Security*, 2002.

[14] E. Kleiner and A.W. Roscoe. Web Services Security: a preliminary study using Casper and FDR. *Proceedings of Automated Reasoning for Security Protocol Analysis (ARSPA 04), Cork, Ireland*, 2004.

[15] E. Kleiner and A.W. Roscoe. On the relationship of traditional and Web Services Security protocols (Full version). 2005. http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/pubs.html/.

[16] Andrew D. Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *XMLSEC '02: Proceedings of the 2002 ACM workshop on XML security*, pages 18–29, New York, NY, USA, 2002. ACM Press.

[17] Mei Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.

[18] K. Bhargavan, C. Fournet, A. Gordon and R. Pucella. TulaFale: A security tool for web services. *Formal Methods for Components and Objects*, 2003.

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

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

[21] Gavin Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, page 31. IEEE Computer Society, 1997.

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

[23] Microsoft Corporation. "Web Services Enhancement (WSE) 2.0 SPI", 2004. http://msdn.microsoft.com/webservices/building/wse/default.aspx.

[24] Lawrence C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.

[25] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. Modelling and analysis of security protocols, 2001.

[26] Peter Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.