Verification of Web Service Protocols by Logic of Knowledge

Qingliang Chen a,b,c,* , Kaile Su a,d , Ying Cao b , Chanjuan Liu a

a School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
b Department of Computer Science, Jinan University, Guangzhou 510632, China
c Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai, China
d Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia

Abstract

Web services is a popular distributed systems technology and its effectiveness and efficiency rely badly on the underlying protocols. And web service protocols are designed in XML formats so the message structures within are quite different from the conventional ones. Therefore, the well-established formal verification techniques for conventional protocols, which have gained substantial achievements in practice, cannot be applied directly to them because of the inherently different syntax. In this paper, we propose a justification-oriented and automatic formal approach to verify, in the standard Dolev-Yao model, concerned security properties expressed as epistemic notions, i.e., logic of knowledge, for web service protocols, based on a fault-preserving mapping tool called SuD (SOAP under Dolev-Yao). Our approach is significant because, instead of finding flaws in finite number of protocol sessions, the specifications we are to verify can hold in arbitrary number of sessions.

Keywords: Web Services; Formal Verification; Logic of Knowledge

1 Introduction

Web services is a popular distributed systems technology and has become a successful crystallization of SOA (Service-oriented Architecture) with wide industrial supports. Over the past few years, in order to standardize this technology, a growing number of specifications has been proposed to define and regulate formally every aspect of Web services by standard body like...
W3C and OASIS [1]. Among these, security is a serious concern and is addressed, in particular, by the WS-Security specification [2]. This specification provides an XML vocabulary for design cryptographic protocols so as to protect the exchanged data between different parties in service interaction from being leaked, exploited or forged. This specification, however, presents no real and practical theory and methodology to stating, validating, and reasoning about security in a rigorous way. Therefore, the users do not have an effective way to be convinced that the underlying web service protocols are buggy or not.

Meanwhile, in another research community of formal verification, there has been a sustained and successful effort to develop formalisms for expressing and verifying conventional cryptographic protocols, for instance, [3, 4]. They provide models, theories, frameworks and tools to verify various security properties against a standard threat model called Dolev-Yao model [5], in which an opponent is able to monitor and manipulate messages sent over the network. Actually, the verification technology can be divided into two categories: falsification-oriented and justification-oriented ones. The former one tries to find flaws within a finite number of protocol sessions by enumerating all possible states implicitly or explicitly; the latter one aims to ‘prove’ mathematically and, sometimes, automatically that a protocol will satisfy a certain security specification within arbitrary number of sessions.

However, the above theory and technology cannot directly be applied to verify web service protocols because they all assume that protocols are expressed in Alice&Bob-style descriptions using fixed, high-level, ad-hoc message formats for dual transmission, instead of the ordered-tree-with-pointers XML files adopted by web service protocols. What’s more, the WS-security specification only focuses on the message formats rather than security goals and their enforcement, because it only details the syntactical issues such as how to sign and encrypt SOAP (Simple Object Access Protocol) messages by the abstraction of security token, which covers data making security claims such as user identifiers, cryptographic keys or certificates. As a result, for any given WS-security compliant protocol, security properties still have to be specified and validated formally.

In order to bridge the gap between the conventional security protocol verification and web service protocol verification induced by their inherent different syntax in exchanged messages, a lot of research effort has been done. Bhargavan, Fournet, Gordon and Pucella made the first attempt to develop a new tool TulaFale [6] based on the spi-calculus and successfully found the XML-rewriting attacks. Kleiner and Roscoe made a significant progress to introduce a fault-preserving mapping function $\phi$ from WS-Security protocols to traditional ones such that web service security protocols can be analyzed formally in Dolev-Yao mode [7]. Furthermore, they developed a corresponding tool called SuD (SOAP under Dolev-Yao) to implement $\phi$ [8]. With the formal tools of CAPSER and FDR [3], they revealed the unexpected attacks upon a web service protocol proposed by the standard body OASIS. In addition to these work, a lot of researchers also uncovered the attacks on different web service protocols based on the theories and tools from traditional formal verification practices such as [9, 10, 11]. Nonetheless, all of them are falsification-oriented and up to this point, no work that is justification-oriented per se has been proposed yet, i.e., to prove web service protocols satisfy some security properties in arbitrary runs of sessions.

This paper aims to address this deficit. Our goal is to justify automatically with tool support, that web service security protocols have some intrinsic features which can be captured by some
logical formalisms. In particular, we state the security properties in epistemic logics [12], i.e., logics of knowledge. The motivation of this paper is to use the logic of knowledge for both the specification and verification of web service security protocols. Based on the fault-preserving mapping function $\phi$ by Kleiner and Roscoe [7] and the corresponding tool SuD (SOAP under Dolev-Yao) [8], we first translate web service security protocols into the normal format, and then feed them to our developed verifier called SPV (Security Protocol Verifier) [13, 14] to check if the expected epistemic specifications can hold or not.

The paper is structured as follows. We will introduce the fault-preserving mapping function $\phi$ by Kleiner and Roscoe [7] in the next section. And then we will discuss how to formalize security properties of web service protocols by logic of knowledge, along with tool to check that. The corresponding experimental results will be illustrated also. Finally, we will conclude the paper in the last section.

### 2 Web Service Protocols under Dolev-Yao Model

Web services is an XML-based architecture that has been developed in order to integrate distributed components better. In network environment, security is always an essential issue. One obvious way to secure Web services is to rely on a secure transport layer such as SSL. However, this technique provides security only in a secure channel and it does not conform to the Web services architecture in which the intermediaries can manipulate the message on its way because once using a secure transport layer intermediaries are not able to control the messages any more.

A better option is using the WS-Security specification [2] that can deal with and define standards and ways of securing SOAP messages by using the XML-signature and XML-encryption specifications, without relying on a secure transport layer. Actually it has become a new sphere for cryptographic protocols in terms of design and implementation.

Yet, due to the inherent difference between the syntax of web service security protocols and that of the traditional ones, the current formal verification techniques can not be applied directly. Kleiner and Roscoe made a promising first step to show that Dolev-Yao [5] model is an accurate model for security analysis for web service protocols, taken that SOAP messages have encapsulated all the security elements [7]. Then they introduce a fault-preserving mapping function $\phi$ from SOAP messages to Casper input which is a concretization of Dolev-Yao model [8], such that if a WS-Security protocol contains the messages $m_1, m_2, ..., m_n$ then,

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

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

Then by property (1) of $\phi$, they use Casper and the FDR [15] to successfully find two attacks on WSS protocols proposed by OASIS in [16]. The contribution of their effort is very significant because they build up the translation while ensuring that flaws and attacks are preserved so that they can be recovered after analysis.
2.1 $\phi$’s Definition

Now we briefly introduce the mapping function $\phi$ here. For clarity, we will use an example of web service protocol taken from [16] to demonstrate the way $\phi$ works. For limited space, the following message $M$ is the first of the protocol. For convenience, we remove the NameSpace (creating no ambiguity) and the TimeStamp element (we here exclude the freshness attacks). In the following the values $BV1,...,BV6$ denote boolean strings holding data (signatures, encryptions, etc).

The functionality of $\phi$ is in fact to extract all the security elements, by parsing the XML tagging system recursively, into the conventional notation of Alice&Bob-style descriptions of security protocols such as $\{A,N_a\}_{K_a}$.

$\phi$ has defined the following operations on XML files:

**Sequence**

$\phi([C_1,...,C_n]) = \langle \phi(C_1),...,\phi(C_n) \rangle$

$\phi([C_1,...,C_n], T) = \langle \phi(C_1,T),...,\phi(C_n,T) \rangle$

**Encryption**

$\phi([C]_K) = \{ \phi(C) \}^{\phi(K)}$

$\phi([C]_K, T) = \{ \phi(C,T) \}^{\phi(K)}$

**Hash**

$\phi(h(C)) = h(\phi(C))$

$\phi(h(C), T) = h(\phi(C,T))$

Then $\phi$ will be applied to a SOAP document and its context. $\phi$ has give all the corresponding rules to deal with all relevant elements within. Take the Envelope for an instance, the result of applying $\phi$ to the Envelope element is the Envelope element’s children list where $\phi$ is applied to every element in the list.

$\phi(\langle Envelope \rangle C_1,...,C_n \langle /Envelope \rangle) = \phi(<C_1,...C_n>)$.

$\phi(\langle Envelope \rangle C_1,...,C_n \langle /Envelope \rangle, T) = \phi(<C_1,...C_n>,T)$.

In a similar way, $\phi$ creates a sequence of the children of the following elements: Header, KeyInfo, SecurityTokenReference, RetrievalMethod, Embedded, SignatureValue, ReferenceList, and DigestValue, etc. For the full details, please refer to [17].

$\phi(\langle TagName \rangle C_1,...,C_n \langle /TagName \rangle) = \phi(<C_1,...C_n>)$.

$\phi(\langle TagName \rangle C_1,...,C_n \langle /TagName \rangle, T) = \phi(<C_1,...C_n>,T)$.

If we take the following message $M$ as the input of $\phi$, we will see the following derivation.

```
<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>
</EncryptedKey>
<EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#content">
<EncryptedMethod Algorithm=...}
\[ \phi(M) \]
\[ = \phi(< \text{Header} > ... < /\text{Header} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \phi(< \text{Security} > ... < /\text{Security} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \phi(< \text{BinarySecurityToken} > ... < /\text{BinarySecurityToken} >), \]
\[ \phi(< \text{EncryptedKey} > ... < /\text{EncryptedKey} >), \phi(< \text{Signature} > ... < /\text{Signature} >), \]
\[ \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \phi(< \text{EncryptedKey} > ... < /\text{EncryptedKey} >), \phi(< \text{Signature} > ... < /\text{Signature} >), \]
\[ \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \phi(< \text{ReferenceList} > ... < /\text{ReferenceList} >, \{K\}, \{K\}_{\phi(<\text{KeyInfo}...<\text{KeyInfo}>,\text{ENC})}, \]
\[ \phi(< \text{Signature} > ... < /\text{Signature} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{KeyInfo}...<\text{KeyInfo}>,\text{ENC})}, \]
\[ \phi(< \text{Signature} > ... < /\text{Signature} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{SecurityTokenReference}...<\text{SecurityTokenReference}>,\text{ENC})}, \]
\[ \phi(< \text{Signature} > ... < /\text{Signature} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{KeyIdentifier}...<\text{KeyIdentifier}>,\text{ENC})}, \]
\[ \phi(< \text{Signature} > ... < /\text{Signature} >), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{ReferenceURI} = \#body > ... < /\text{Reference} >)}, \]
\[ \{\phi(< \text{KeyInfo} > ... < /\text{KeyInfo} >, \text{SIG}), \phi(< \text{Body} > ... < /\text{Body} >) \}
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{DigestMethod}... >)(\phi(<\text{body}))), \phi(<\text{KeyInfo}...<\text{KeyInfo}>, \text{SIG}), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{ReferenceURI} = \#body > ... < /\text{Reference} >)}, \]
\[ \{\phi(< \text{KeyInfo} > ... < /\text{KeyInfo} >, \text{SIG}), \phi(< \text{Body} > ... < /\text{Body} >) \}
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{DigestMethod}... >)(\phi(<\text{body}))), \phi(<\text{KeyInfo}...<\text{KeyInfo}>, \text{SIG}), \phi(< \text{Body} > ... < /\text{Body} >) \]
\[ = \text{Context}(\text{enc}, \{K\}), \{K\}_{\phi(<\text{EncodedDataId}="enc" >...<\text{EncryptedData}>)}, \phi(<\text{KeyInfo}...<\text{KeyInfo}>, \text{SIG}), \]
\[ \phi(< \text{Body} > ... < /\text{Body} >) \]

Message: M
Fortunately, the above tedious process can be implemented by a tool SuD (SOAP under Dolev-Yao) [8] instead of by hand. After applying $\phi$ by SuD to both of the messages we get the following protocol translation (Fig. 1).

![Fig. 1: Applying $\phi$ by SuD](image)

Therefore, $\phi$ can transform SOAP messages into their equivalent notions for protocols in conventional format. And what’s more, Kleiner and Roscoe also formally prove that $\phi$ is fault-preserving. The notion of fault-preserving transformations was introduced by Lowe and Hui in [18]: transformations that preserve attacks. The effect of these transformations is that if the correctness of a transformed protocol is established, then the correctness of the original protocol is proved as well.

**Theorem 1 ([17])** $\phi$ is a fault-preserving function.
3 Verification of Web Service Protocols by Logic of Knowledge

3.1 Epistemic Specifications

The specifications of a security protocol can be very complex, concerning a principal’s knowledge or belief about another principal’s knowledge or belief during or at the end of a protocol run. The information exchange itself can be seen as an epistemic-theoretic problem. Therefore, in order to analyze those protocols with complex specifications or goals, it is natural to express them in epistemic logics, i.e., logics of knowledge.

Epistemic logics [19, 12] has been successfully applied to reasoning about communication protocols (cf., for example, the derivation of the alternating bit protocol [20] or, more recently, the analysis of TCP [21]). However, the underlying assumption for these communication protocols is that the network is not hostile.

BAN logic [22], by Burrows, Abadi and Needham, is the first and one of the most influential logical tools to reason about security protocols using epistemic notations and has inspired some variants like SVO [23]. However, either they do not have a model-theoretic semantics, or they are too abstract to have algorithmic supports.

In order to address this deficiency, we have proposed a knowledge-theoretic model for verification of cryptographic protocols [13]. Our basic model can be for both compact representations and efficient computations. And what’s more, a corresponding tool called SPV (Security Protocol Verifier) has been developed and successfully verified a lot of protocols [14].

Our framework can naturally represent and reason about the following protocol property $\varphi$:

if principal $\tau$ has normally completed a run of a protocol with a nonce number $c$, and apparently with herself taking role $B$ and principal $\rho$ taking role $A$, then principal $\rho$ has previous been running the protocol with the nonce number $c$, and apparently with principal $\tau$ taking role $B$ and principal $\rho$ taking role $A$.

In our framework, we are able to express and reason about complex and interesting properties such as “principal $\tau$ knows principal $\rho$ knows property $\varphi$”, denoted by $K_\tau K_\rho \varphi$.

3.2 Experimental Results and Analysis

In this paper, we will take a web service protocol: WS-SecureConversation based protocol taken from [24] as an example for epistemic verifications.

The protocol comprises four SOAP messages. After applying $\phi$ by SuD to them, we get the following equivalent and conventional protocol.

1. $A \rightarrow B : \text{UMI}_1, \{\text{UMI}_1, B, \{K_1\}_{PK(B)}\}_{\text{sha1(password}(A), N_A), \text{sha1(password}(A), N_A), \{K_1\}_{PK(B)}}$
2. $B \rightarrow A : \{\text{UMI}_1, \text{UMI}_2, \{K_2\}_{K_1}\}_{SK(B), \text{uuid1}, \{K_2\}_{K_1}}$
3. $A \rightarrow B : \{\text{UMI}_3, B, \text{body1}\}_{\text{sha1}(K_1, K_2), \{\text{body1}\}_{\text{sha1}(K_1, K_2)}}$
4. $B \rightarrow A : \{UMI_3, UMI_4, body2\}_{sha1(K_1,K_2)}, \{body2\}_{sha1(K_1,K_2)}$

The authentication goals can be formalized as $\text{Agreement}(A, B, [v_1, ..., v_n])$ [25], which specifies that $A$ is correctly authenticated to $B$ and the agents agree upon the roles they are taking and the values $v_1, ..., v_n$. Furthermore, there is a one-one relationship between the runs of $B$ and the runs of $A$. The same meaning can be given reversely for $\text{Agreement}(B, A, [v_1, ..., v_n])$. Actually, this can be semantically equivalent to $\text{PRECEDES } B : A[[v_1, ..., v_n]]$ in CAPSL (Common Authentication Protocol Specification Language, http://www.csl.sri.com/users/millen/capsl/).

And in another logical formalism first proposed by BAN [22] and revised by lots of efforts such as SVO [23], we know that the authentication can be further expressed in the following epistemic logic:

$$KBKA_{said}(B, [v_1, ..., v_n]), KAKB_{said}(A, [v_1, ..., v_n])$$

which means that $B$ knows (is sure) that $A$ knows (is sure) that he is talking to $B$ with message values $v_1, ..., v_n$, and the second can be explained in the same way.

### 3.3 Experimental Results and Analysis

When we input the protocol above to our tool SPV (Security Protocol Verifier) [14] and check the authentication properties concerned, we find that some of them hold and some does not. And the failed specifications imply that potential attacks may exist. Actually, the failed specifications are exactly the source of the man-in-the-middle attacks found in [17], as the authors pointed out. Our experimental results coincide with their analysis completely but in another way which is justification-oriented instead of their falsification-oriented one. The following Table 1 summarizes the results. They are carried out on a machine with 3.00GHz CPU, 1.48GB DDR2 on Ubuntu 9.10 (2.6.31-14 kernel with GCC-4.4.1).

<table>
<thead>
<tr>
<th>Specification</th>
<th>Time (in seconds)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>precedes $A : B</td>
<td>K_1$</td>
<td>1.53209</td>
</tr>
<tr>
<td>precedes $B : A</td>
<td>K_2$</td>
<td>0.15601</td>
</tr>
<tr>
<td>$KAKB_{said}(A, K_1)$</td>
<td>132.436</td>
<td>No</td>
</tr>
<tr>
<td>$KBKA_{said}(B, K_2))$</td>
<td>1.5681</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 4 Conclusion

In this paper, we propose a formal and automatic approach for web service protocols to verify, whether or not they can satisfy the concerned specifications expressed by logic of knowledge [13] within arbitrary number of sessions, based on the tool SuD (SOAP under Dolev-Yao) [8] and our SPV (Security Protocol Verifier) [13, 14]. The results of this justification-oriented approach can reveal some inherent features of web service protocols in another perspective, and thus provide some insights on how to improve the protocol in the design phase.
Our work can be deepened in the following ways. For example, to state more security goals for web service protocols by logic of knowledge such as anonymity, non-repudiation, etc; and investigate the modular approach [11] to verify complex web service protocols.

References