Application of Model Checking to AXML System's Security: A Case Study[*]

Il-Gon Kim and Debmalya Biswas

IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
{ikim, dbiswas}@irisa.fr

Abstract. An Active XML (AXML in short) has been developed to provide efficient data management and integration by allowing Web services calls to be embedded in XML document. AXML documents have new security issues due to the possibility of malicious documents and attackers. To solve this security problem, document-level security with embedded service calls has been proposed to overcome the limitation of traditional security protocols.

The aim of this paper is to show how existing model checking technique, with CSP and FDR, used for traditional message-based security protocols, can be adapted to specify and verify AXML document-based security. To illustrate our approach, we present the framework for modelling and analyzing AXML document's security. Then, we demonstrate how this technique can be applied to analyze electronic patient record taken from [13]. Finally, we show the possible vulnerabilities due to delegated query and malicious service call.

1 Introduction

In the context of Web services and XML, data integration and management have been an important issue, due to the heterogeneity and autonomy of data sources. Active XML (AXML in short) has been developed to provide efficient data management and integration by allowing Web services calls to be embedded in XML document[1][3]. For example, the possibility of *intensional* data(embedded service calls) in AXML document leads to powerful data management by allowing dynamic collaboration with distributed systems and discovering new relevant data sources at run-time.

However, AXML has also brought the following security issues: 1) it is necessary to protect peers from malicious AXML documents, and 2) it is required to protect AXML documents from malicious peers. To solve the above security problems, document-level security with embedded service calls as well as XML Encryption and XML Signature has been studied[6][13].

Over the last decade, great attention has been paid to the question of developing formal methods for analyzing security protocols over the last decade. While some methods have been successfully applied to verify security properties

[*] This work is supported by the INRIA projects ARC-ASAX and RNRT-SWAN.

of traditional message-based security protocols, they have not yet been applied to analyze security problems specific to AXML document-based systems. For example, AXML documents support query delegation by invoking embedded service calls, which are not considered in SOAP message security. In addition, it is worth noting that the formal specification and verification issues related with AXML documents include new types of security aspects not considered in traditional message-based protocols. For example, an AXML document is basically an XML document and service calls. As such, it is necessary to develop an abstract model by analyzing XML tagging and embedded service calls. AXML document invokes embedded security-related service calls in order to obtain a key and generate encrypted or signed document. This means that an abstract model could be extracted from two viewpoints: 1) before invoking a security service call, and 2) after invoking a security service call. Besides, it also needs to reflect the fact that there would be more security threats in addition to traditional one such as overhearing and modifying transmitted messages. For example, an intruder could embed enormous amount of false data or additional service calls in the returned AXML document to the intended recipient after intercepting the original document.

In this paper, we show how existing model checking technique, generally used for analyzing message-based protocols, can be adapted to verify new vulnerabilities of AXML systems. To do this, we have chosen formal analysis techniques based on Casper/CSP and FDR because it has already been proven to be very successful for verifying traditional protocols[20] as well as SOAP message-based protocols[14][16].

The remainder of this paper is organized as follows. Section 2 gives a brief overview of AXML and its security services. In Section 3, we show how to specify and analyze AXML documents by invoking security service calls. In Section 4, we describe how AXML documents encrypted or signed with XML Encryption and XML Signature could be translated systematically to an abstract security notation. In Section 5, we demonstrate the case study of analyzing electronic patient record. Section 6 describes some related works. Finally, we conclude in Section 7.

2 Overview of AXML Document and Security Service Calls

AXML documents are basically XML documents where some parts of data are explicitly denoted and other parts are given intensionally, by embedded service calls within the documents. The <sc></sc> tags in an AXML document represents a service call and its children subtrees denote the parameters of the Web service calls. After invoking an embedded service call on the document, a corresponding Web service is executed. Then the results of invoking the embedded service calls are appended at the location of the service call in the document. We use the terminology *materialization*, which means that the associated Web service is invoked, and its result is returned to the location of the service call.

Thus, an AXML system consists of a set of AXML documents plus the services in those documents.

The corresponding security services consist of a tuple *(p, s)*, where $p \in P$ is the identity of a peer providing the service, and $s \in S$ is the name for a security service such as encryption. The algebraic expressions for an AXML document(in short, a document), tree, function node, Web service, and service evaluation on peers are simply expressed as below[2] :

- *d@p* : a document d at peer p
- *q@p* : a query q at peer p
- *s@p* : a service s provided by peer p
- *f(para$_1$,...,para$_n$)@p* : a function node f to invoke a corresponding security service s defined on peer p, with parameters, para$_1$,...,para$_n$
- *Result(Z)* : evaluation result Z of service s (defined on p) on peer p

In the rest of this paper, the above algebraic expressions will be used to describe the document exchange between peers and the document d represents all or some forest of AXML documents. For more details about AXML syntax and semantics, see [3].

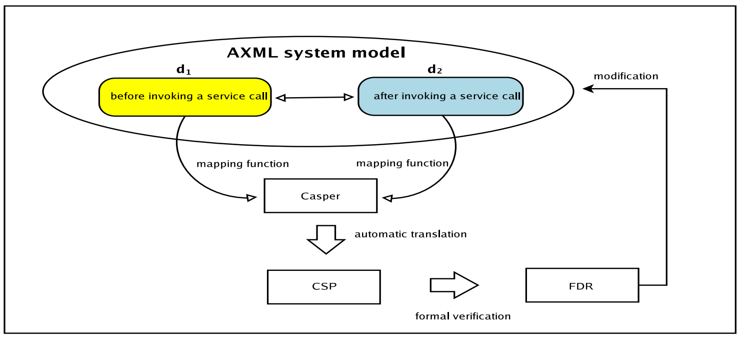
3 Modelling and Analyzing AXML Document Embedded with Security Service Call

3.1 Framework for Modelling and Analysis

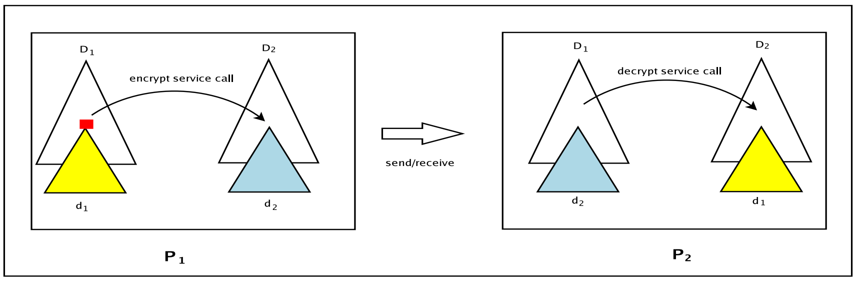
In Fig. 1(a), the framework for modelling and analyzing security services in AXML system is illustrated. In this framework, Casper[18] (Compiler for the Analysis of Security Protocols) is a compiler that converts a high level description of a protocol into CSP (Communicating Sequential Processes) code that can be run in a model checker FDR (Failure-Divergence Refinement). CSP[9] is a process algebra language to describe systems as a number of processes which operate independently and communicate with each other over well-defined channels. FDR[11] is a model checking tool for state machines, with foundations in the theory of concurrency based on CSP.

Given an AXML system, its model can be considered from two viewpoints: 1) *an AXML document before invoking a service call*, and 2) *an AXML document after invoking a service call.* We denote the former as 'd_1' and the latter as 'd_2'. These two models('d_1' and 'd_2') are transformed into a high-level security notation according to derivation rules of the δ mapping function.

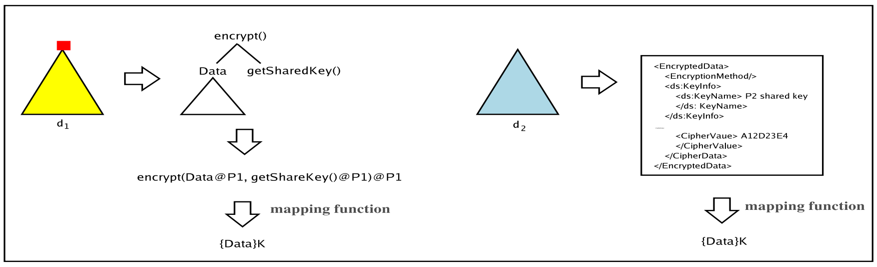
First, the common security notation of Casper input is created($\delta(d_1) = \delta(d_2)$), after applying the δ functions to d_1 and d_2. Next, CSP code is generated automatically using Casper's compilation function. Then, the FDR model checker shows the possible attacker scenarios if the CSP code doesn't hold any given security property. Thus, the verification results will be helpful for a designer to modify an AXML system to be robust against security vulnerabilities.



(a) framework for modelling and analyzing security services



(b) service call processes of encryption and decryption



```
<EncryptedData>
  <EncryptionMethod/>
  <ds:KeyInfo>
    <ds:KeyName> P2 shared key
    </ds: KeyName>
  </ds:KeyInfo>
<CipherData>
    <CipherVaue> A12D23E4
    </CipherValue>
  </CipherData>
</EncryptedData>
```

(c) mapping function to generate Casper input

Fig. 1. Model-based verification of security services in an AXML system

Fig. 1(b) shows the process of encryption and decryption in AXML documents. P_1 is an AXML peer and it invokes a local service call, *encrypt*(denoted by a square), and d_1 refers to a subtree including a service call node and data to be encrypted. d_2 is the materialized result after evaluating the *encrypt* service call and it is encoded based on XML Signature and XML Encryption standards. For example, P_1 could obtain the shared key by using the function node *getSharedKey*, then it could invoke the *encrypt* service call, *encrypt(Data, EncryptedData)*, to encrypt the data with the shared key. Then encrypted AXML document is encoded according to the standard format defined in XML Encryption(see Fig. 1(c)). Similarly, if the function node related with signature(e.g., *sign(Data, getPrivateKey()))* is called, the AXML document would be signed according to the standard format defined in XML Signature.

The main advantages of using the proposed framework for analyzing security and constructing the δ mapping function to generate Casper input can be summarized as follows:

– Using the δ mapping functions enables us to generate Casper input systematically.
– If an attack is found on $\delta(\mathrm{d}_1)$ and $\delta(\mathrm{d}_2)$, then the corresponding attack exists on a real AXML system as well.
– If an attack is found on a real AXML system, then the corresponding attack exists on $\delta(\mathrm{d}_1)$ and $\delta(\mathrm{d}_2)$ as well.
– If an attack is found on an AXML document before invoking a service call, then the corresponding attack still exists in the AXML document after materialization of the service call.

3.2 Extending CSP Model for AXML Documents

***Document* Datatype.** Analogous to the *Message* type defined in [17], AXML *document* datatype could be based on the *Atom* set where security function nodes are such as *getSharedKey()@p* $\subseteq$ *Atom*, *getPublicKey()@p* $\subseteq$ *Atom*, *getPrivateKey()@p* $\subseteq$ *Atom*, *Hash(data)@d* $\subseteq$ *Atom*, *encrypt(data, getSharedKey()@p)@p* $\subseteq$ *Atom*, *encrypt(data, getPublicKey()@p)@p* $\subseteq$ *Atom*, and *sign(data, getPrivateKey()@p)@p* $\subseteq$ *Atom*. Simple definition of the *Document* datatype by the BNF(Backus-Naur Form) expression could be similar to the *Message* type as shown below:

Definition 1. *A document d or data value v* $\in$ *Document might be an atom (Atom), concatenated data(v.v), data encrypted with a key(*$\{v\}_K$*), or a digested message with hash function h.*

$$\begin{aligned}
&a \in Atom ::= P \mid N \mid K \\
&f(para_1,\ldots,para_n)@p ::= a \\
&d \in Document ::= v \mid d.d \mid \{d\}_K \mid h(d) \\
&v \in Data ::= a \mid v.v \mid \{v\}_K \mid h(v)
\end{aligned}$$

where P ranges over the set Agent of agent names, K over the set Key of keys(e.g., PK(p) : public key of agent p, SK(p) : private key of agent p), and N over the nonce(random number) set. The concatenation notation '.' is associative.

AXML document security allows *selective encryption* which means that it is possible to encrypt all or a specific part of an AXML document(denoted by $\{d\}_K$ and $\{v\}_K$, respectively). Here, the expression *f(para*$_1$*,...,para*$_n$*)@p* could be considered as all or some parts of an AXML document before or after invoking a service call(see d_1 or d_2 in Fig. 1).

Intruder Model. In the classical CSP model for an attacker, it is generally assumed that an intruder has the following abilities to attack honest agents:

- overhear or intercept all messages flowing through the network
- construct and deliver spurious messages disguised as a trusted peer
- forward intercepted messages to another peer
- decrypt messages that are encrypted with his own public key

In addition to the above attack abilities, we assume that an intruder p_I could have the following new abilities : 1) manipulate the XML-based elements and 2) return falsified document *d'* containing other malicious service calls and fabricated data.

Because of the addition of two attack abilities in the CSP model, we also need to modify the traditional CSP model, based on the five basic deduction rules[17], that allow an intruder to construct new data or document. In the classical inference model, $B \vdash m$ represents that the intruder may derive message m from the set of messages B. For example, if the intruder can produce an encrypted message and the corresponding decrypting key, then he could decrypt the message. This sample decryption rule instance for an intruder can be adapted as follows:

decryption rule : $B \vdash \{d\}_k \wedge B \vdash$ k $\Longrightarrow B \vdash d$
AXML system's decryption :
$B \vdash$ <EncryptedData>
        <EncryptionMethod.../ >
        <KeyInfo><KeyName> shared key k < /KeyName>< /KeyInfo>
        <CipherData><CipherValue>A12D23E< /CipherValue>< /CipherData>
    < /EncryptedData>
$\wedge$ $B \vdash$ <KeyInfo><KeyName> shared key k < /KeyName>< /KeyInfo>
$\Longrightarrow$ $B \vdash$ <document>...< /document>

The intruder CSP process in an AXML system consists of three main channels: 1)*send* to intercept every document or data sent by the honest peers, 2) *receive* to forward intercepted document or data disguised as an honest peer, and 3) *leak* to decrypt secret information and create falsified document *d'*.

$$\begin{aligned} \mathit{Intruder(B)} \;\widehat{=}\; & \square_{d \in Document} \; \mathit{send?P_1?P_2!d} \rightarrow \mathit{Intruder(close(B \cup \{d\}))} \\ \square\; & \square_{d \in Document, B \vdash d} \; \mathit{receive?P_1?P_2!d} \rightarrow \mathit{Intruder(B)} \\ \square\; & \square_{d \in Document, B \vdash d} \; \mathit{leak.d} \rightarrow \mathit{Intruder(B)} \end{aligned}$$

The initial state of the intruder is *Intruder(IK)* containing initial knowledge *IK* which is a member of *facts*(such as all peer's identity, all kinds of keys that peers' possess). The function close(B) calculates all *facts*(simply *B*) that are deducible or buildable from B under the deduction rules.

The complete AXML system is constructed similarly with a classical CSP model. For more detail information about CSP model for a traditional security protocol, see [20].

$$\mathrm{SYSTEM}_{AXML} \;\widehat{=}\; (\mathrm{P}_1 \;|||\; \mathrm{P}_2 \;|||\; \ldots \;|||\; \mathrm{P}_n) \;||\; \mathrm{INTRUDER}_{PI}$$

Fault-Preserving Simplifying Transformation. Hui et al.[10] have proved that if one can verify the transformed protocol, then it will have the same effect as the verification of the original protocol. Based on this fault-preserving technique, E. Kleiner et al. have [15] proved that even if automatic translation function is used to generate Casper input from WS-Security SOAP messages, it preserves the same inference process of intruder and the corresponding attacks in the real WS-Security application. We can apply this proof to the AXML system model. Thus, the abstract CSP model($SYSTEM_{AXML}$) for AXML systems satisfies the following two conditions for fault-preserving as shown in [10],[15]:

1. $\forall B \in \mathbb{P}(Document); d \in Document \bullet B \cup IK \vdash d \Rightarrow \delta(B) \cup IK_{AXML} \vdash \delta(d)$
2. $\delta(\mathrm{IK}) \subseteq \mathrm{IK}_{AXML}$

The first condition means that if an intruder can deduce the document or data in the original *SYSTEM*, he would be able to deduce the equivalent one d in the transformed $SYSTEM_{AXML}$. The second condition represents that all the corresponding *facts* of an intruder's initial knowledge *IK* in the original *SYSTEM* is a subset of the transformed $SYSTEM_{AXML}$.

Therefore, we can say that if an attack is found on the abstract $SYSTEM_{AXML}$, then the corresponding attack can also be found on the original *SYSTEM* and vice versa.

4 Case Study: Electronic Patient Record

Step 1: Dr. Kim(p_1) sends query q_1 to Paris hospital(p_2) by invoking the service call "*diagnosis@*p_2" in order to look into the patient record of the patient *Suzzanne* before diagnosing her.

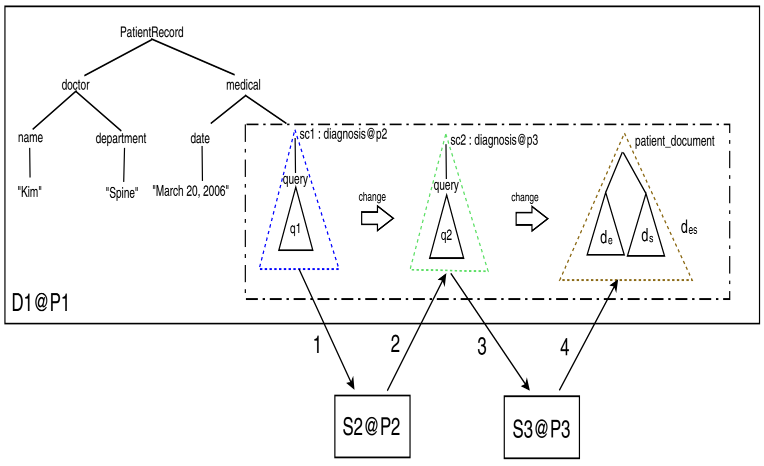


Fig. 2. Service call steps of AXML document in peer p_1

Step 2: Paris hospital(p_2) performs access control by enforcing the relevant access control rules(denoted "AC") and the query "q_1" gets rewritten into "q_2" . In this example, we assume that a corresponding access control rule for Dr. Kim is defined in the access control systems of Paris hospital :

AC: Dr. Kim, /PatientRecord/(name ∪ ssn ∪ visit/(medical_doctor ∪ diagnosis ∪ xray))

Now, suppose the query q_1 and the filtered query q_2 are :
q$_1$ **:** /PatientRecord[name="Suzzanne", ssn="123-45-6789"]
q$_2$ **:** /PatientRecord[name="Suzzanne", ssn="123-45-6789"]/(name ∪ ssn ∪ visit/(medical_doctor ∪ diagnosis ∪ xray))

For details of the access control mechanism proposed for AXML systems, the reader is referred to [5].

Paris hospital filters the query q_1 as q_2 and it finds that there is no patient related after evaluating the query q_2. Given this, let us assume that the Paris hospital(p_2) finds out that Rennes hospital(p_3) has related a patient record. Paris hospital also subscribes to regular patient record updates from other hospitals such as Rennes hospital. Then, it returns a query signed by itself for delegating Dr. Kim to invoke a service provided by Rennes hospital(*query delegation*).
Step 3: Dr. Kim invokes the service call "*diagnosis@*p_3" with the parameter of "q_2" signed by Paris hospital so that the query will be evaluated by Rennes hospital.
Step 4: Rennes hospital verifies the query q_2 using its own public key and assures itself that Dr. Kim has been delegated to invoke the service call "*diagnosis@*p_3" and use q_2. After evaluating the service call, Rennes hospital returns the diagnosis record for *Suzzanne*, encrypted with a shared key k and signed by Rennes hospital (the shared key k itself is encrypted by the public key of *Dr. Kim*). It is extremely important that a patient record should be protected from any unauthorized modification, whether accidental or not.

4.1 Model Construction from d_1

As mentioned in Section 4, we show how to construct an AXML system model from d_1 which is the document before invoking a security service call. First, we describe a model in AXML algebraic expression(see Section 2), then we write it in Casper notation(see Section 4.3).

AXML Expression

1. $p_1 \rightarrow p_2$: *q1@*p_1 [p_2 computes : q_2]
2. $p_2 \rightarrow p_1$: *encrypt(*q_2*@*p_2*, getPrivateKey()@*p_2*)@*p_2
3. $p_1 \rightarrow p_3$: *Result(Z)*

4a. $p_3 \rightarrow p_1$ *: encrypt(d@*p_3*, getSharedKey(random()@*p_3*)@*p_3*)@*p_3*,*
encrypt(getSharedKey()@p_3*, getPublicKey()@*p_1*)@*p_3
4b. $p_3 \rightarrow p_1$ *: encrypt(digest(d@*p_3*)@*p_3*, getPrivateKey()@*p_3*)@*p_3

The message sequences listed above represent exchanges of queries, data, or documents. p_1, p_2, and p_3 are *Dr. Kim*, *Paris hospital*, and *Rennes hospital* peers, respectively. In step 3, *Result(Z)* represents the result document of evaluating the service call in step 2. The messages 4a and 4b show the nested service calls to generate encrypted and signed patient records, where $d@p_3$ is the document of patient diagnosis d on peer p_3. The first *encrypt* call and the second one are used to transform the d encrypted with a random shared key and generate the shared key encrypted with the public key of p_1, respectively. The two materialized results are combined into the XML Encryption encoded document, d_e. Similarly, the *encrypt* service in message 4b is used to generate the XML Signature encoded document, d_s. After finishing all the data exchanges, p_1 invokes the *decrypt* local service calls related to d_e and d_s.

4.2 Model Construction from d_2

We demonstrate how to construct an AXML system model from d_2(see Section 3) by showing the translation process from the materialization results of the patient document $\delta(d_{es})$ to Casper input. The d_e and the d_s in d_{es} are encoded in XML based on the XML Encryption and XML Signature standards.

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod
        Algorithm="http://www.w3.org/2000/09/xmlenc#3des-cbc"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptedMethod
                Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:KeyName>
                    Dr. Kim's Public Key
                </ds:KeyName>
            </ds:KeyInfo>
            <CipherData>
                <CipherValue>A23B45C56 ... </CipherValue>
            </CipherData>
            <CarriedKeyName>
                Symmetric Key with Dr. Kim
            </CarriedKeyName>
        </EncryptedKey>
        <ds:KeyName> Symmetric Key with Dr. Kim </ds:KeyName>
    </ds:KeyInfo>
    <CipherData>
        <CipherValue>ErBGCQHKJOOaqbmiibhGk ... </CipherValue>
    </CipherData>
</EncryptedData>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <SignatureMethod
            Algorithm="http://www.w3.org/2000/07/xmldsig#rsa-sha1"/>
        <Reference URI="">
            <DigestMethod
                Algorithm="http://www.w3.org/2000/07/xmldsig#sha1"/>
            <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
        </Reference>
    <SignedInfo>
    <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>
        <KeyName>Rennes Hospital's Integrity Key</KeyName>
    </KeyInfo>
</Signature>
```

We use '$\Rightarrow$' to represent the message derivation process from $\delta(d_e)$ and $\delta(d_s)$ to Casper input. The derivation rule for generating a Casper input is similar to [14], but our approach is different from it because the former is focused towards SOAP messages based on WS-Security.

$\delta(d_{es}) = \delta(d_e),\ \delta(d_s)$

$\delta(d_e)$
$\Rightarrow \delta($<EncryptedData>. . .</EncryptedData>$)$
$\Rightarrow \delta($<CipherData>. . .</CipherData>$)$
$\Rightarrow \delta($<KeyInfo>. . .</KeyInfo>$)$, $\delta($<EncryptedMethod. . ./>$)$
$\Rightarrow \delta($<KeyName>. . .</KeyName>, R$)$, $\delta($<EncryptedKey>. . .</EncryptedKey>$)$,
$\delta($<EncryptedMethod Algorithm="http://www.w3.org.2001/04/xmlenc♯3des-cbc"/ >$)$
$\Rightarrow \{\text{d}\}_k$, $\delta($<CipherData>. . .</CipherData>$)$
$\Rightarrow \{\text{d}\}_k$, $\delta($<KeyInfo>. . .</KeyInfo>$)$, $\delta($<EncryptedMethod. . ./>$)$
$\Rightarrow \{\text{d}\}_k$, $\delta($<KeyName>. . .</KeyName>, $R)$,
$\delta($<EncryptedMethod Algorithm="http://www.w3.org.2001/04/xmlenc♯rsa-1_5"/ >$)$
$\Rightarrow \{\text{d}\}_k,\ \{k\}_{PK(p3)}$

$\delta(d_s)$
$\Rightarrow \delta($<Signature>. . .</Signature>$)$
$\Rightarrow \delta($<SignatureValue>. . .</SignatureValue>$)$
$\Rightarrow \{\delta($<SignedInfo>. . .</SignedInfo>$)\}\delta($<KeyInfo>. . .</KeyInfo>$)$
$\Rightarrow \{\delta($<SignatureMethod. . .>. . .</SignatureMethod. . .>$)$,$\delta($<Reference. . .>. . .</Reference. . .>$)\}$
$\delta($<KeyInfo>. . .</KeyInfo>$)$
$\Rightarrow$ $\{\delta($<SignatureMethod Algorithm = "http://www.w3.org/2000/07/xmldsig♯rsa-sha1"/>$)$,
$\delta($<Reference URI="">$)\}$ $\delta($<KeyInfo>. . .</KeyInfo>$)$
$\Rightarrow \{\delta($<DigestMethod>. . .<DigestMethod>$)\}$ $\delta($<KeyName>. . .</KeyName>, $Sig)$
$\Rightarrow \{\delta($<DigestValue>. . .<DigestValue>$))\}$ $SK(p3)$
$\Rightarrow \{sha(d)\}SK(p3)$

$\therefore\ d_{es} = \delta(d_e),\ \delta(d_s) = \{\text{d}\}_k,\ \{\text{k}\}_{PK(p1)},\ \{\text{sha(d)}\}_{SK(p3)}$

4.3 Analysis of Security Services

The design of security protocol using document-level security would be error-prone when considering security requirements in new emerging applications, complex communication steps with many peers, and a powerful attacker. In this subsection, we use Casper notation to model sequences of exchanging queries or documents depicted in Fig. 2. Then we analyze some security requirements (confidentiality and authentication) using FDR model checker:

security requirements:
- authentication
  - *Dr. Kim*(p_1) must be sure that it received a patient record document from *Rennes hospital*(p_3).
- confidentiality
  - A confidential document of patient record d must not be leaked by an unauthorized peer.

For a security analysis, we assume that all encryption algorithms are secure and an intruder cannot perform any cryptanalysis. We also assume that an intruder p_I has the following initial knowledge set:

intruder knowledge:
$\{p_1,\ p_2,\ p_3,\ PK(P),\ SK(p_I),\ K_I\} \in Intruder(IK)$ where K_I is an intruder's share key, and $\{p_1,\ p_2,\ p_3,\ p_I\} \in P$.

Here, we translate a protocol description into Casper syntax based on sequences for exchanging documents in Fig. 2. In addition, we generate Casper input systematically from AXML documents denoted in XML Encryption and XML Signature in a similar way to [14].

Sequences for exchanging queries or documents:

1. $\mathrm{p}_1 \longrightarrow \mathrm{p}_2 : \mathrm{q}_1$
2. $\mathrm{p}_2 \longrightarrow \mathrm{p}_1 : \{\mathrm{q}_2\}_{SK(p_2)}$
3. $\mathrm{p}_1 \longrightarrow \mathrm{p}_3 : \{\mathrm{q}_2\}_{SK(p_2)}$
4. $\mathrm{p}_3 \longrightarrow \mathrm{p}_1 : \{\mathrm{d}\}_k, \{\mathrm{k}\}_{PK(p_1)}, \{\mathrm{sha(d)}\}_{SK(p_3)}$

In casper notation, we use the expression $\{d\}_k$ to represent the data or document d encrypted with key k. The public key function is represented as *PK* and the private key function is expressed as *SK*. For example, a pair of public key and private key of peer 'p_1' is written in *PK(*p_1*)* and *SK(*p_1*)*, respectively. The hash function *SHA-1* in XML Signature is denoted as *sha* in protocol description. For example, the message 4 means that p_3 sends the messages of the patient record d encrypted with shared key k, encrypted shared key with the public key of p_1, and signed message digest with a hash function.

```
Secret(p3, d, [p1])
Secret(p1, d, [p3])
Secret(p3, k, [p1])
Secret(p1, k, [p3])
Agreement(p3, p1, [d, k])
```

We verified the confidentiality and authentication properties, which are defined in the above. The lines beginning with *Secret* represent the *confidentiality property.* For example, the statement '*Secret(*p_3*, d, [*p_1*])*' is interpreted as "p_3 believes that the confidential information d is a secret that should be known only to p_1".

The line starting with *Agreement* defines the *authentication property.* The *authentication property* represents the establishment guarantees when it has completed, concerning the party it has apparently been running with. For example, the fourth one means that "p_3 is authenticated to p_1 with d and k".

In particular, we assume that the intruder could generate the falsified document *d'* containing other embedded service calls in the materialization result. Then, this may lead to DoS(Denial-of-Service) attack in a peer if the following two properties of secrecy and authentication are not satisfied in CSP trace event sets *tr* of $SYSTEM_{AXML}$.

1. $\mathit{signal.Claim\_Secret.p_a.p_b.d}$ **in** $tr \wedge \mathrm{p}_a \in \mathrm{Honest} \wedge \mathrm{p}_b \in \mathrm{Honest} \Rightarrow \neg(\mathit{leak.d}$ **in** $tr)$
2. $p_b \in \mathit{Honest} \Rightarrow \mathit{signal.Running.RESPONDER.p_b.p_a}$
   precedes $\mathit{signal.Commit.INITIATOR.p_a.p_b}$

where:

- *signal.Claim_Secret.*p_a*.*p_b*.d* means that p_a thinks that the patient document d is a secret which should be known only to p_b.
- *signal.Running.RESPONDER.*p_b*.*p_a represents that the responder p_b thinks he started a protocol run apparently with the initiator p_a.
- *signal.Commit.INITIATOR.*p_a*.*p_b represents that the initiator p_a thinks that he has completed a protocol run apparently with the responder p_b.

After analyzing the property statements of '*Secret(*p_3*, d, [*p_1*])*' and '*Secret(*p_3*, k, [*p_1*])*', the FDR tool shows no counterexample about them. However, when the FDR is applied to other property statements, we found that the following attack scenario could be derived from its counterexample:

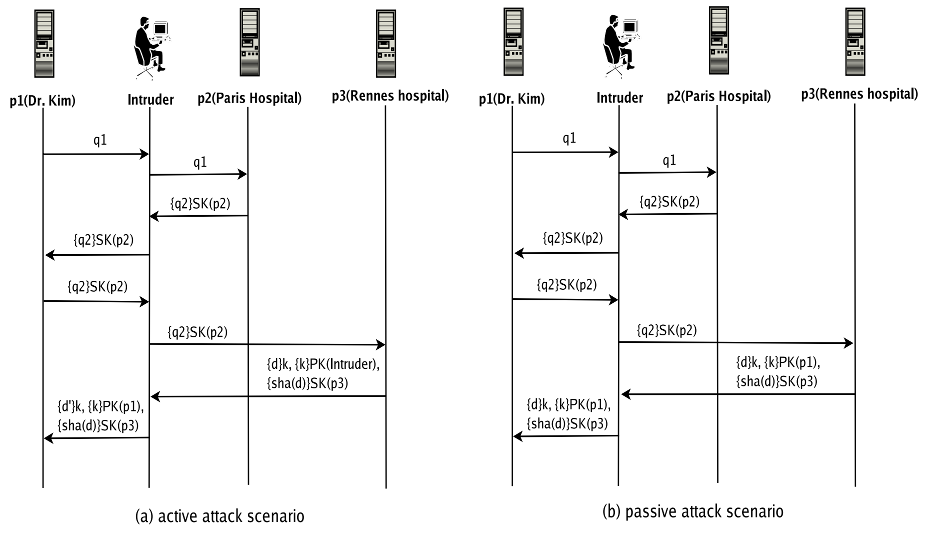


Fig. 3. Attack scenarios on the electronic patient record

Security Vulnerabilities. In Fig. 3, an intruder could monitor query q_1 and intercept the filtered query q_2 delegated by p_2. Then, the intruder could send the intercepted delegation query to p_3 disguised as an honest peer p_1. Here we can consider two different kinds of attack scenarios:

In the first case, p_3 might regard an attacker p_I as an honest peer because it has already been authorized and possesses q_2 delegated by p_2. This disguise is possible in a real Web service world because p_3 may not know who the original initiator of the service transaction is. Then, the intruder can successfully decrypt the encrypted patient record d and create the falsified patient record *d'*. A more intelligent intruder may return an original patient document without modification and instead he can embed other recursive service calls inside the document itself. Although p_1 uses *lazy query evaluation*[1] to filter irrelevant service calls, it must spend important resources wastefully such as CPU processor and memory space. Even in the worst case, this result might be linked to denial-of-service attack.

In the second case, (assumption) p_3 exactly knows p_1 as the initiator of Web Service transaction . Through the man-in-the middle attack, the intruder can intercept the encrypted signed documents($\{\mathrm{d}\}_k$, $\{\mathrm{k}\}_{PK(p_1)}$, $\{\mathrm{sha(d)}\}_{SK(p_3)}$) from p_3. Even if the intruder can not decrypt the patient's record due to non possession of the private key of p_1, there might be potential attack that the received documents could be reused for a different patient next time. Eventually, mismatched patient records may has a dangerous effect on the patient's health.

The main vulnerability in this example is based on the insecure usage of *query delegation*, not the vulnerability of the document-level security itself. As such, the intruder could intercept the delegation query '$\{\mathrm{q}_2\}_{SK(p_2)}$' that bypasses access control in p_2 and disguises as an honest p_1, even if the intruder has no proper right to request a patient record document of p_2. A simple solution to this attack scenario is to add the identity p_1 and the filtered query q_2 signed by itself in message 3, as p_1, $\{\{q_2\}_{SK(p_2)}\}_{SK(p_1)}$. This countermeasure prevents the intruder from generating a modified message and sending it to p_3, because p_3 checks who would be the intended responder for d through p_1's identity in the signed message. Another solution is to use a time-stamp with a short validity period against replay attack. More detailed information related to using a time-stamp against replay attack can be found in [20].

5 Related Works

Model checking with FDR has proved to be very successful for modelling and analyzing security aspects in traditional protocols[17],[18]. Relatively, few studies have been devoted to analyze Web services security with model checking technology.

Eldar Kleiner et al.[14] showed how the WS-Security specification[12] could be mapped to Casper and analyzed with FDR. Llanos Tobarra et al.[16] also used Casper/FDR tools and illustrated how to analyze some security properties of a Web service application as a licence server, developed by Microsoft Web Services Enhancement (WSE)[19]. Karthikeyan Bhargavan et al.[4] developed a tool called *TulaFale* to specify SOAP-based security protocols in pi-calculus and analyze its vulnerabilities.

The above approaches analyzed some vulnerabilities in existing XML-based Web service messages focusing only on the SOAP communication channel constructed by WS-Security. They do consider document-level security. To the best of our knowledge, there is no research to describe how to model AXML documents combined with embedded security service calls and analyze the vulnera-

[1] *Lazy query evaluation* was proposed to detect which calls may bring relevant data for query execution and to avoid the materialization of irrelevant information[1].

bilities due to delegated query and malicious service calls in the document. In this regard, we believe our approach to be different from the above related works.

6 Conclusion

Active XML (AXML) has been evolving as one of the new challenging researches in distributed, autonomous Web Services paradigm, by combining XML data and embedded Web services calls to allow simple and dynamic data management. Furthermore, security is one of the most vital topics in Web services development today and will in the foreseeable future.

In this paper, we have shown how existing model checking techniques with Casper/CSP and FDR, used for the verification of classical security protocols, could be applied to analyze vulnerabilities of AXML documents as well. To the best of our knowledge, this is the first approach to analyze the security of AXML documents using model checking.

We have explained the framework for adapting a classical CSP model to AXML systems and have shown how to build a Casper input from two models: 1) the document before invoking a service call, and 2) the document after invoking a service call.

Finally, we have demonstrated the usefulness of our approach by modelling and analyzing an electronic patient record. We found that a careless usage of delegated query could lead to security weakness and this vulnerability may induce falsification of data or DoS attacks by malicious document from an untrusted peer.

The analysis results also provide a hint that security services based on XML Encryption and XML Signature between AXML peers do not provide a complete security solution by themselves. The combination of other complementary security standards such as SAML and XACML would make an AXML system more robust against powerful, intellectual attacks in distributed networks.

Acknowledgements. We would like to thank Stefan Haar, Loic Helouet, Serge Abiteboul, Bogdan Cautis, and anonymous referees for their valuable feedback which helped to improve the work in the previous draft considerably.

References

1. S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. "*Lazy Query Evaluation for Active XML*", *Proceedings of ACM SIGMOD Conference*, pp.227-238, 2004.
2. S. Abiteboul, I. Manolescu, and F. Taropa, "*A Framework for Distributed XML Data Management*", *Proceedings of EDBT 2006*, pp.1049-1058, 2006.
3. Active XML Home Page (AXML), *http://activexml.net*, 2004.
4. K. Bhargavan, C. Fournet, A. Gordon, and R. Pucellla. "*TulaFale: A security tool for web services*", In *Formal Methods for Components and Objects: International Symposium, FMCO 2003*, volume 3188 of *Lecture Notes in Computer Science*, pp.197-22. 2003.

5. S. Abiteboul, B. Alexe, O. Benjelloun, B. Cautis, I. Fundulaki, T. Milo, and A. Sahuguet. "*An Electronic Patient Record on Steroids : Distributed, Peer-to-Peer, Secure and Privacy-conscious*", *Proceedings of the 30th VLDB Conference*, pp.1273-1276, 2004.
6. S. Abiteboul, O. Benjelloun, B. Cautis, and T. Milo. "*Active XML, Security and Access Control*", *Proceedings of the SBBD Workshop*, pp.13-22, 2004.
7. D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and E. Simon. "*XML-Encryption synatx and Proceeding*", W3C Recommendation, 2001.
8. 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.
9. C.A.R. Hoare,*Communicating Sequential Processes*, 1985.
10. M.L. Hui and G. Lowe. Fault-preserving Simplifying Transformations for Security Protocols. *Journal of Computer Security*, 9(1/2):3-46, 2001.
11. Formal Systems(Europe) Ltd. FDR2 User Manual, Aug. 1999.
12. IBM, Microsoft, and VeriSign, Web Services Security(WS-Security), Version 1.0, April 2002.
13. I.G. Kim and D. Biswa. Secure Data Management based on AXML Document : Electronic Patient Record, 2006(submitted).
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)*, 2004.
15. E. Kleiner and A.W. Roscoe. "*On the Relationship between Web Services Security and Traditional Protocols*", DIMACS Workshop on Security of Web Services and E-Commerce, 2005.
16. L. Tobarra, D. Cazorla, F. Cuartero, and Gregorio Diaz. "Applicatoin of Formal Methods to the Analysis of Web Services Security", *2nd International Workshop on Web Services and Formal Methods*, pp.215-229, 2005.
17. G. Lowe. "*Breaking and fixing the Needham-Schroeder public-key protocol using FDR*", *Proceedings of TACAS, number 1055 in LNCS. Springer*, pp.147-166, 1996.
18. G. Lowe, "A Compiler for the Analysis of Security Protocols", *Proceedings of the 10th Computer Security Foundations Workshop*, 1997.
19. Microsoft, Microsoft Web Services Enhancements (WSE) 2.0, *http://msdn. microsoft.com/webservices/building/wse. Proceedings of ACM SIGMOD*, pp.289-300, 2003.
20. P.Y.A. Ryan and S. A, Schneider. *Modelling and Analysis of Security Protocols: the CSP Approach*, Addison-Wesley, 2001.