

April-June 2004, Vol 1, No. 2

Table of Contents

- v **Editorial Preface: Behind the Scenes of Web Services Negotiation and Agreement**
Patrick C.K. Hung, Hong Kong University of Science and Technology (HKUST), Hong Kong
Liang-Jie Zhang, IBM T.J. Watson Research Center, USA
The authors of the editorial preface introduce this issue by highlighting each paper and propose evidence to support the idea that a Web services negotiation and agreement model is required in Web services.
- Reviewed Papers**
- 1 **An Extensible Workflow Architecture through Web Services**
Jinyoung Jang, Hyunse Systems, Inc., Korea
Yongsun Choi, Inje University, Korea,
J. Leon Zhao, University of Arizona, USA
In this study, the authors propose an extensible workflow architecture based on Web services and object-oriented techniques. The proposed architecture contains three components, the Open Kernel Framework, the Universal Resource Manager, and the Process Definition Publisher.
- 16 **Web Service Business Context – The Normative Perspective**
Olivera Marjanovic
University of New South Wales, Australia
This paper investigates the normative perspective of the Web service business context. The main objective of this research is to facilitate better understanding of this phenomenon and ultimately initiate development of the new tools for management of the normative aspect both by the service providers and their customers.
- 37 **Merkle Tree Authentication in UDDI Registries**
Elisa Bertino, Purdue University, USA
Barbara Carminati, Università dell’Insubria, Italy
Elena Ferrari, Università dell’Insubria, Italy
UDDI registries are today the standard way of publishing information on Web services. They can be thought of as a structured repository of information that can be queried by clients to find the Web services that better fit their needs. In this paper, the authors focus on authenticity issues by proposing a method based on Merkle Hash Trees, which does not require the party managing the UDDI to be trusted with authenticity.
- 58 **QoS-Aware and Federated Enhancement for UDDI**
Chen Zhou, Liang-Tien Chia, Bu-Sung Lee
Nanyang Technological University, Singapore
With more and more Web services appearing on the Web, a Web service’s discovery mechanism becomes essential. UDDI is an online registry standard to facilitate the discovery of business partners and services. However, the ability to predict the quality of service is missing in UDDI. To address this problem, the authors propose UX (UDDI eXtension), a system that facilitates requesters to discover services with good performance.

Merkle Tree Authentication in UDDI Registries

Elisa Bertino, Purdue University, USA
Barbara Carminati, Università dell'Insubria, Italy
Elena Ferrari, Università dell'Insubria, Italy

ABSTRACT

UDDI registries are today the standard way of publishing information on Web services. They can be thought of as a structured repository of information that can be queried by clients to find the Web services that better fit their needs. Even if, at the beginning, UDDI has been mainly conceived as a public registry without specific facilities for security, today security issues are becoming more and more crucial, due to the fact that data published in UDDI registries may be highly strategic and sensitive. In this paper, we focus on authenticity issues by proposing a method based on Merkle Hash Trees, which does not require the party managing the UDDI to be trusted with authenticity. In the paper, besides giving all the details of the proposed solution, we show its benefit with standard digital signature techniques.

Keywords: UDDI authenticity; digital signature; XML; Merkle hash trees

INTRODUCTION

XML Web services are today becoming the platform for application integration and management on the Internet. Basically, an XML Web service is a software service with three main characteristics: 1) the use of a standard Web protocol (in most cases SOAP [Soap]) to expose the service functionalities; 2) an XML-based description (through WSDL

[WSDL]) of the interface; and 3) the use of UDDI [UDDIv3] to publish information regarding the Web service and to make this information available to potential clients. UDDI is an XML-based registry with the primary goal of making widely available information on Web services. It thus provides a structured and standard description of the Web service functionalities, as well as searching facilities to help in finding the provider(s) that better fit the

client requirements. Even if, at the beginning, UDDI has been mainly conceived as a public registry without specific facilities for security, today security issues are becoming more and more crucial, due to the fact that data published in UDDI registries may be highly strategic and sensitive. In this respect, a key issue regards authenticity: it should be possible for a client querying a UDDI registry to first verify that the received answer actually originated at the claimed source, and, then, that the party managing the UDDI registry did not maliciously modify some of its portions before returning them to a client. To deal with this issue, UDDI specifications allow one to optionally sign some of the elements in a registry, according to the W3C XML Signature syntax.

Authenticity issues are particular crucial when UDDI registries are managed according to a third-party architecture. The basic principle of a third-party architecture is the distinction between the owner, who produces the information, and one or more *publishers*, who are responsible for managing (a portion of) the owner information and for answering client queries. Such architectures are today becoming more and more popular, because of their scalability and the ability of efficiently managing a large number of clients and a great amount of data. UDDI can be implemented according to either a third-party or a two-party architecture. A third-party architecture consists of a *service provider*, that is, the owner of the services, the *service requestors*, that is, the parties who request the services, and a *discovery agency*, that is, the UDDI registry. In a two-party architecture, there is no dis-

tinction between the service provider and the discovery agency. In this paper we focus on authenticity issues for third-party implementations of UDDI. The main problem is how the owner of the services can ensure the authenticity of its data, even if the data are managed by a third-party (i.e., the discovery agency). The most intuitive solution is that of requiring the discovery agency to be trusted with respect to authenticity. However, the main drawback of this solution is that large Web-based systems cannot be easily verified to be trusted and can be easily penetrated. For this reason, in this paper, we propose an alternative approach, which we have previously developed for generic XML data (BCFTG) distributed according to a third-party architecture. The main benefit of the proposed solution is that it does not require the discovery agency to be trusted with authenticity. It is important to remark that in the scenario we consider, it is not possible to directly apply standard digital signature techniques to ensure authenticity, since a client may require only selected portions of a document, depending on its needs, and thus it is not enough that the owner of the data signs each document it sends to the publisher. For this reason, we apply an alternative solution, which requires that the owner sends the publisher, in addition to the information it is entitled to manage, a summary signature, generated using a technique based on Merkle hash trees (Merkle, 1989). The idea is that when a client submits a query to a publisher requiring any portion of the managed data, the publisher sends him/her, besides the query result, also the signatures of the documents on which the query

is performed. In this way, the client can locally recompute the same bottom-up hash value signed by the owner, and by comparing the two values he/she can verify whether the publisher has altered the content of the query answer and can thus verify its authenticity. The problem with this approach is that since the client may be returned only selected portions of a document, he/she may not be able to recompute the summary signature, which is based on the whole document. For this reason, the publisher sends the client a set of additional hash values referring to the missing portions that make the client able to locally perform the computation of the summary signature.

In the current paper, we show how this approach can be applied to UDDI and we discuss its benefits. Additionally, we describe the prototype implementation we have developed for supporting the proposed approach.

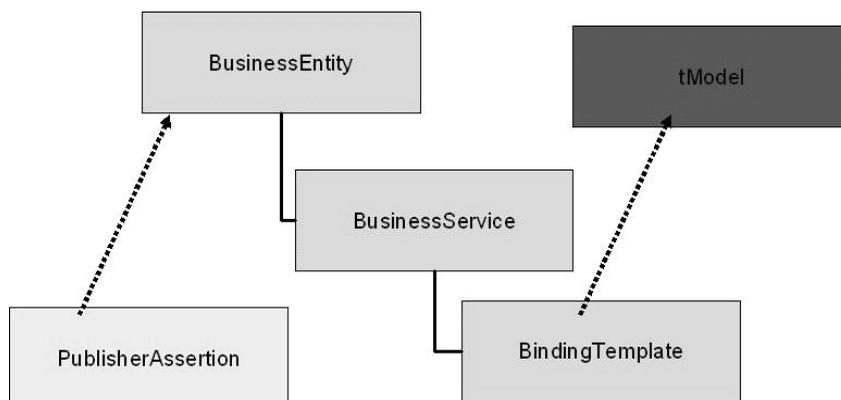
The remainder of this paper is organized as follows. After a brief overview of the UDDI registries, in Section 3 we

present the authentication method based on Merkle hash tree, whereas in Section 4 we show how it can be exploited in the UDDI environment. In Section 5, we compare our approach with the traditional digital signature techniques. Then, in Section 6 we give some details about the prototype implementation. Finally, Section 7 concludes the paper, whereas in Appendix A we report a brief explanation of the XML signature syntax.

UDDI REGISTRIES

The main goal of a UDDI registry (UDDIv3) is to supply potential clients with the description of businesses and the services they publish, together with technical information about the services, thus making the requestor able to directly require the service that better fits its needs. The UDDI registry organizes all these descriptions into a single entry of the UDDI register. More precisely, each entry is composed of five main data structures (see Figure 1), namely, the `businessEntity`, the

Figure 1. UDDI main data structures



businessService, the bindingTemplate, the publisherAssertion and the tModel, which are briefly described in what follows.

The businessEntity provides general information about the business or the organization providing the Web services (e.g., the name of the organization, or the contact person). Additionally, a UDDI entry contains one businessService data structure for each service provided by the business or organization and described by the businessEntity. This data structure contains a technical description (i.e., the bindingTemplate data structure) of the service, and information about the type of the service (i.e., the tModels data structure). By contrast, the PublisherAssertion data structure models the relationships existing among different businessEntity elements.

For example, by this data structure it is possible to represent the relationships among the UDDI entries corresponding to subsidiaries of the same corporations.

Figure 2 reports an example of the XML representation of a UDDI entry. In particular, this entry represents the DICO organization (i.e., the name element contained into the businessEntity element), which has specified only one contact person, that is, Barbara Carminati (i.e., the personName element contained into the businessEntity element). According to Figure 2, the DICO organization provides only one service, called Service1 (i.e., the name element contained into the businessService element), whose binding template is accessible at URL www.example.it/service.asmx (i.e., the

Figure 2. The businessEntity element

```
<?xml version="1.0" encoding="UTF-8"?>
<businessEntity businessKey="9ECDC890-23EC-11D8-B78C-89A8511765B5" operator="jUDDI.org"
authorizedName="Carminati">
  <discoveryURLs>
    <discoveryURL useType="BusinessEntity">http://dico.unimi.it</discoveryURL>
    <discoveryURL
useType="businessEntity">http://localhost:8080/juddi/discovery?businessKey=9ECDC890-23EC-11D8-
B78C-89A8511765B5</discoveryURL>
  </discoveryURLs>
  <name xml:lang="it">DICO</name>
  <description xml:lang="it">Dipartimento Informatica e Comunicazione</description>
  <contacts>
    <contact>
      <personName>Barbara Carminati</personName>
      <email>carminati@dico.unimi.it</email>
      <address>
        <addressLine>Via Comelico, 39</addressLine>
        <addressLine>20135 Milano</addressLine>
      </address>
    </contact>
  </contacts>
  <businessServices>
    <businessService serviceKey="9F063DB0-23EC-11D8-B78C-ECBB5F8B0CFC" businessKey="9ECDC890-23EC-
11D8-B78C-89A8511765B5">
      <name>Service 1</name>
      <description>Example service</description>
      <bindingTemplates>
        <bindingTemplate bindingKey="9F063DB0-23EC-11D8-B78C-F7A09CE94F7B" serviceKey="9F063DB0-23EC-
11D8-B78C-ECBB5F8B0CFC">
          <description>Binding Example 1</description>
          <accessPoint URLType="www.example.it/service.asmx"></accessPoint>
          <tModelInstanceDetails />
        </bindingTemplate>
      </bindingTemplates>
    </businessService>
  </businessServices>
  <identifierBag />
  <categoryBag />
</businessEntity>
```

accessPoint element contained into the bindingTemplate element).

UDDI registries give clients searching facilities for finding the provider(s) that better fit the client requirements. More precisely, according to the UDDI specification, UDDI registries support two different types of inquiry: the drill-down pattern inquiries (i.e., get_XXX API functions), which return a whole core data structure (e.g., businessTemplate, businessEntity, operationalInfo, businessService, and tModel), and the browse pattern inquiries (i.e., find_XXX API functions), which return overview information about the registered data.

XML MERKLE TREE AUTHENTICATION

The tree authentication mechanism proposed by Merkle (1989) is a well-known mechanism for certifying query processing. One of the most important uses of this mechanism has been proposed by Naor and Nissim (1998), which exploits the Merkle trees for solving the problem of creating and maintaining efficient authenticated data structures holding information about the validity of certificates. The Merkle trees are also used in the contest of micropayments (Charanjit & Yung, 1996) to minimize the number of public key signatures needed in issuing or authenticating a sequence of certificates for payments. Moreover, an approach exploiting the Merkle tree authentication mechanism for proving the completeness and authenticity of queries on relational data has been proposed by Devanbu et

al. (Devanbu, Gertz, Martel & Stubblebine, 2000). In Bertino, Carminati, Ferrari, Thuraingham and Gupta (to appear) we have proposed the use of such trees for XML documents. In this section, we summarize the basic principles of our approach.

Merkle Signature

The approach we propose for applying the Merkle tree authentication mechanism to XML documents is based on the use of the so-called *Merkle signatures*. This signature allows one to apply a unique digital signature on an XML document by ensuring at the same time the authenticity and integrity of both the whole document, as well as of any portion of it (i.e., one or more of its elements/attributes). The peculiarity of the Merkle signature is the algorithm used to compute the digest value of the XML document being signed. This algorithm, which exploits the Merkle tree authentication mechanism, associates a different hash value, called Merkle hash value, with each node (i.e., elements/attributes) in the graph representation of an XML document. Before presenting the function computing these Merkle hash values, we need to introduce the notation we adopt throughout the paper. Given an element e , we use the dot notation $e.content$ and $e.tagname$ to denote the data content and the tagname of e , respectively. Moreover, given an attribute a , the notation $a.val$ and $a.name$ is used to denote the value and the name of attribute a , respectively.

Definition 1. Merkle hash func-

tion()). Let d be an XML document, and v a node of d (i.e., an element, or a node). The Merkle hash value associated with a node v of d , denoted as $MhX_d(v)$, is computed by the following function:

$$MhX_d(v) = \begin{cases} h(h(v.val)||h(v.name)) & \text{if } v \text{ is an attribute} \\ h(h(v.content)||h(v.tagname)||MhX_d(child(1,v))||\dots \\ ||MhX_d(child(Nc_v,v))) & \text{if } v \text{ is an element} \end{cases}$$

where ‘||’ denotes the concatenation operator, and function $child(i,v)$ returns the i -th child of node v , with Nc_v denoting the number of children of node v .

According to Definition 1, the Merkle hash value associated with an attribute is the result of a hash function applied to the concatenation of the hashed attribute value and the hashed attribute name. By contrast, the Merkle hash value of an element is obtained by applying the same hash function over the concatenation of the hashed element content, the hashed element tagname, and the Merkle hash values associated with its children nodes, both attributes and elements.

As an example, consider the XML document d in Figure 2, containing the businessEntity element defined according to the UDDI specification. The Merkle hash value of the contacts element ($MhX_d(contacts)$) is the result of the hash function computed over the concatenation of the element content, if any, its tagname, and the Merkle hash values associated with its children nodes (i.e., contact elements).

The important point of the proposed approach is that, if the correct Merkle hash value of a node v is known by a client, an

untrusted third party or an intruder cannot forge the value of the children of node v , as well as its content and tagname. Thus, for instance by knowing only the Merkle hash value of the root element of an XML document, the client is able to verify the authenticity and integrity of the whole XML document. To ensure the integrity of the Merkle hash value of the document root element, we impose that the owner of the data signs this value, and we refer to this signature as the Merkle signature of the document.

The main benefit of the proposed technique with traditional digital technique is when a third-party architecture is adopted like the UDDI, that is, when there exists a third party that may prune some nodes from a document, as a result of the query evaluation. In this case, the traditional approach of digital signatures is no longer applicable, since its correctness is based on the assumption that the signing and verification processes are performed on exactly the same bits. By contrast, if the Merkle signature is applied, the client is still able to validate the signature, provided that he/she receives from the third party a set of additional hash values, referring to the missing document portions. This makes the client able to locally perform the computation of the summary signature and compare it with the received one. We refer to this additional information as the *Merkle hash path*, defined in the next section.

Merkle Hash Paths

Intuitively, the Merkle hash paths can be defined as the hash values of those

nodes pruned during query evaluation, and needed by the client for computing the Merkle signature. In general, given two nodes v, w in an XML document d , such that $v \in Path(w)$ ¹, the Merkle hash path between w and v , denoted as $MhPath(w, v)$, is the set of hash values necessary to compute the Merkle hash value of v having the Merkle hash value of w . The formal definition is given in what follows.

Definition 2. Merkle Hash Path – $MhPath()$. Let d be an XML document, and let v, w be two nodes in d such that $v \in Path(w)$. $MhPath(w, v)$ is a list consisting of the following hash values:

- $\{h(f.content), h(f.tagname) \mid f \in Path(w, v) \setminus \{w\}\}$
- $\{MhX_d(e) \mid e \in sib(f), \text{ where } f \in Path(w, v) \setminus \{v\}\}$

where $sib()$ is a function that, given a node f , returns f 's siblings

Thus, the Merkle hash path between w and v consists of the hash values of the tagname and content of all the nodes belonging to the path connecting w to v (apart from w), plus the Merkle hash values of all the siblings of the nodes belonging to the path connecting w to v (apart from v).

To better clarify how the proposed approach works, consider Figure 3, which depicts three different examples of Merkle hash paths.² In the figure, the triangle denotes the view returned to the client, whereas black circles represent the nodes whose Merkle hash values are returned together with the view, that is, the Merkle hash paths. Consider the first example reported in Figure 3. The Merkle hash path between nodes 4 and 1 consists of the Merkle hash values of nodes 5 and 3, plus the hash values of the tagname and content of nodes 2 and 1. Indeed, by using node w , the Merkle hash value of node 5, and the hash value of the tagname and content of node 2, it is possible to compute the Merkle hash value of node 2. Then, by using the Merkle hash values of

Figure 3. Examples of Merkle hash paths

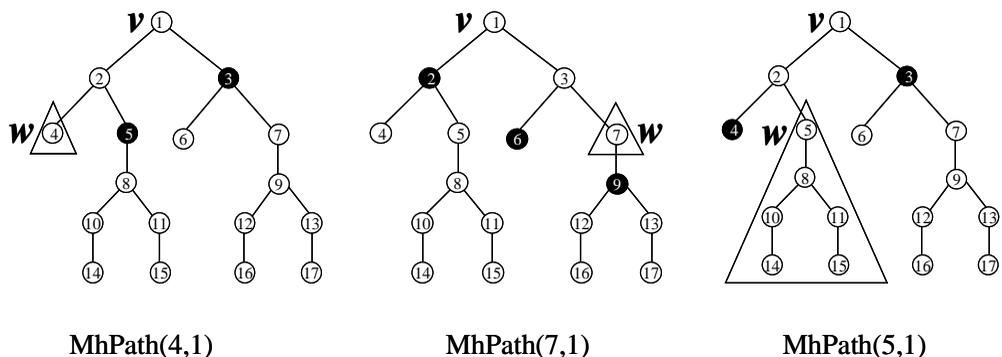
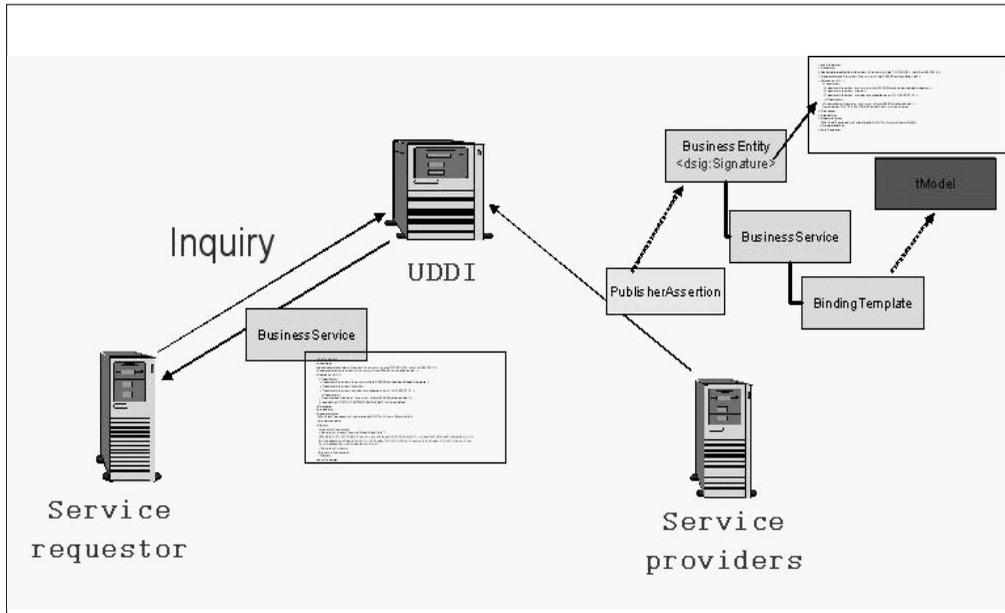


Figure 4. The Merkle signature in UDDI environment



nodes 2 and 3, and the hash values of the tagname and content of node 1, it is possible to compute the Merkle hash value of node 1. In the second example in Figure 3, the view consists of a non-leaf node. In such a case $MhPath(7,1)$ contains also the Merkle hash values of the child of node 7, that is, node 9. Thus, by using the Merkle hash value of nodes 9 and 7, it is possible to compute the Merkle hash value of node 7. Then, by using this value, the Merkle hash value of node 6 and the hash values of the tagname and content of node 3, it is possible to generate the Merkle hash value of node 3. Finally, by using the Merkle hash values of nodes 3 and 2, and the hash values of the tagname and content of node 1, it is possible to generate the Merkle hash value of node 1. By contrast, in the third example the view consists of the whole subtree rooted at node 5. In such a case, $MhPath(5,1)$ does not

contain the hash values of the children of node 5. Indeed, since the whole subtree rooted at 5 is available, it is possible to compute the Merkle hash value of node 5 without the need for further information. Then, similarly to the previous examples, by using the Merkle hash values of nodes 5 and 4, and the hash values of the tagname and content of node 2 (these last values supplied by $MhPath(5,1)$), it is possible to compute the Merkle hash value of node 2. Finally, by having the Merkle hash values of nodes 2 and 3, and the hash values of the tagname and content of node 1, it is possible to compute the Merkle hash value of node 1. We can note that if the query result consists of an entire subtree, the only necessary Merkle hash values necessary are those associated with the siblings of the node belonging to the path connecting the subtree root to the document root.

APPLYING THE MERKLE SIGNATURE TO UDDI REGISTRIES

In this section, we show how we can apply the authentication mechanism, illustrated in the previous section, to UDDI registries. As depicted in Figure 4, the proposed solution implies that the service provider first generates the Merkle signature of the businessEntity element, and then publishes it, together with the related data structures, in the UDDI registry. Then, when a client inquires the UDDI, the Merkle signature as well as the set of necessary hash values (i.e., the Merkle hash paths, computed by the UDDI) are returned by the UDDI to the requesting client together with the inquiry result.

Adopting this solution requires determining how the Merkle signature and the Merkle hash paths have to be enclosed in the businessEntity element, and inquiry result, respectively. To deal with this issue, we make use of the dsig:Signature element introduced in the latest UDDI specification (UDDI, 2002). Indeed, to make the service provider able to sign the UDDI entry, the latest UDDI specifica-

tion supports an optional dsig:Signature element that can be inserted into the following elements: businessEntity, businessService, bindingTemplate, publisherAssertion, and tModel. Thus, according to the XML Signature syntax (World Wide Web Consortium, 2001), a service provider can sign the whole element to which the signature element refers to, and also it can exclude selected portions from the signature, by applying a transformation (see Appendix A for an overview on the XML signature syntax).

Therefore, in order to apply the Merkle signature to the UDDI environment, and at the same time to be compliant with the UDDI specification, we represent both the Merkle signature and the Merkle hash paths according to the XML Signature syntax, that is, by using the dsig:Signature element. In the following sections, we give more details on the proposed representation.

Merkle Signature Representation

In Figure 5, we show how the dsig:Signature element can be used to wrap the Merkle signature. Note that the URI

Figure 5. An example of signature element storing the Merkle signature

```

<dsig:Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xnl-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        <Transform Algorithm="urn:uddi-org:schemaCentricC14N:2002-07-10"/>
        <Transform Algorithm="Merkle"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>1fR07/Z/XFW375JG22bNGmFblMY</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue> W0uO9b47TqmipunAwmF4ubn1mdsb4HYR17c+3ULmLL2BxslwSsl6kQ
</SignatureValue>
</ dsig:Signature>

```

attribute of the Reference element is empty and thus it identifies the XML document where the Signature element is contained, that is, the businessEntity element. In addition to the required enveloped signature and scheme-centric canonicalization transformations, the dsig:Signature element specifies also a Merkle transformation, through a Transform element whose Algorithm attribute is equal to "Merkle". This last transformation indicates to the client and UDDI registries that the service provider has computed the Merkle signature on the businessEntity element.

It is important to note that the syntax of the Transforms element implies an order according to which the various transformations should be applied. In particular, this order is given by the order in which the corresponding Transform elements appear in their parent element. Thus, to generate the digital signature contained into the dsig:Signature element presented in Figure 5, it is first necessary to apply the enveloped signature transformation and the scheme-centric canonicalization. Then, the Merkle hash function is computed on the obtained result. Finally, the obtained digest value is digitally signed according to the XML Signature Recommendation.

Merkle Hash Path Representation

According to the strategies depicted in Figure 4, once a client inquires a UDDI registry, the UDDI registry computes the corresponding Merkle hash path and returns it to the client together with the inquiry result. As we will see in the next section, the latest UDDI specification states that for some kinds of inquiries (i.e., the

get_xxx inquiries), the UDDI registry has to include in the inquiry answer also the dsig:Signature element corresponding to the data structure returned as inquiry results. For this reason, we also represent the Merkle hash paths into the dsig:Signature element, supplying thus the client with the additional information needed for verifying the authenticity and integrity of the inquiry results.

To enclose this information into the dsig:Signature element, we exploit the dsig:SignatureProperties element, in which additional information useful for the validation process can be stored.

In Figure 6 we present an example of dsig:Signature element containing the dsig:SignatureProperties element, which is inserted as direct child of an Object element. It is important to note that, according to the XML Signature generation process, the only portion of the dsig:Signature element which is digitally signed is the SignedInfo element.

Thus, by inserting the Object element outside the SignedInfo element, the UDDI registry does not invalidate the signature. This allows the UDDI to complement the dsig:Signature element representing the Merkle signature of the businessEntity element with the dsig:SignatureProperties element containing the appropriate Merkle hash paths, and then to insert it into the inquiry answer. More precisely, during the Merkle signature validation, the client must be able to recompute the Merkle hash value of the businessEntity element to compare it with the Merkle signature. In order to do that, the client must know the Merkle hash value of each subelement of the businessEntity element not included into

Figure 6. An XML Signature element complemented with Merkle hash paths

```

<dsig:Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xnl-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        <Transform Algorithm="urn:uddi-org:schemaCentricC14N:2002-07-10"/>
      </Transforms>
      <Transform Algorithm="Merkle"/>
    </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>1fR07/Z/XFW375JG22bNGmFblMY=</DigestValue>
  </Reference>
</SignedInfo>
<SignatureValue>
W0uO9b47TqmpunAwmF4ubn1mdsb4HYR17c+3ULmLL2BxslwSsl6K
</SignatureValue>
<Object>
  <SignatureProperties>
    <SignatureProperty Target="MerkleHashPath">
      <businessEntity authorizedName="valore" operator="juddi.org" hash="sldghoghor....">
        <discoveryURLs hash="fdsgbdsl...." />
        <identifierBag hash="57438tgvk...." />
          <categoryBag hash="57438tgvk...." />
            <businessServices>
              <businessService>
                <description hash="gherogh..." />
                <bindingTemplates hash="hgkvdlsvf...." />
                <categoryBag hash="hdsbghfdlb..." />
              </businessService>
              <businessService>
                <description hash="gherogh..." />
                <bindingTemplates hash="hgkvdlsvf...." />
                <categoryBag hash="hdsbghfdlb..." />
              </businessService>
            </businessServices>
          </businessEntity>
        </SignatureProperty>
      </SignatureProperties>
    </Object>
  </dsig:Signature>

```

the inquiry answer (i.e., the Merkle hash path). To make the validation simpler, the Merkle hash paths are organized into an empty businessEntity element (see Figure 6), whose children contain a particular attribute, called hash, storing the Merkle hash value of the corresponding element. This businessEntity element is inserted into the dsig:SignatureProperties element.

MERKLE SIGNATURES VS. XML SIGNATURES IN UDDI REGISTRIES

In this section, we explain the differences and the benefits that could be at-

tained by adopting in UDDI registries the Merkle signature approach instead of the traditional digital signature techniques. Before doing that, it is interesting to note that similarly to Merkle signature, also the XML Signature syntax allows one to generate a different hash value for each different node of the XML document, and then to generate a unique signature of all these values. This feature is obtained by means of the Manifest element, which creates a list of Reference elements, one for each hashed node. However, this solution does not consider the structure of the XML document, thus ensuring only the authenticity of the data content and not of the

relationships among nodes.

In the following, we separately consider the possible inquiries that a client can submit to a UDDI registry, that is, the `find_xxx` and `get_xxx` inquiries.

get_xxx inquiries

According to the UDDI latest specification, the service provider can complement all the data structures that could be returned by a `get_xxx` API call with a `dsig:Signature` element. However, to ensure the authenticity and integrity of all the data structures the service provider must compute five different XML signatures (one for each different element). Whereas, by using the Merkle signature approach the service provider generates only one signature, that is, the Merkle signature of the `businessEntity` element. Thus, a first benefit of our approach is that by generating only a unique signature it is possible to ensure the integrity of all the data structures. When a client submits a `get_xxx` inquiry, the UDDI returns him/her the whole requested data structure, where the inserted `dsig:Signature` element contains the Merkle signature generated by the service provider, together with the Merkle hash path between the root of the returned data structure and the `businessEntity` element.

As an example, consider the `get_bindingDetail` inquiry. The UDDI specification states that the answer to the `get_bindingDetail` inquiry must be the `bindingTemplates` element, containing a list of `bindingTemplate` elements together with the corresponding `dsig:Signature` elements. In such a case, a UDDI registry exploiting the Merkle signature approach should sub-

stitute each `dsig:Signature` element contained into the `bindingTemplate` elements with the signature generated by the service provider, that is, the `dsig:Signature` element published together with the `businessEntity`. Moreover, according to the representation proposed in the previous sections, the UDDI registry should insert into the `dsig:Signature` element the `dsig:SignatureProperties` subelement, which stores the Merkle hash path between the `bindingTemplate` element and the `businessEntity` element.

find_xxx inquiries

We now analyze the other types of inquiry, that is, the `find_xxx` inquiries. We recall that these inquiries return overview information about the registered data. Consider, for instance, the inquiry API `find_business` that returns a structure containing information about each matching business, including summaries of its business services. This information is a subset of those contained in the `businessEntity` element and the `businessService` elements. For this kind of inquiry, the UDDI specification states that if a client wants to verify the authenticity and integrity of the information contained in the data structures returned by a `find_xxx` API call, he/she must retrieve the corresponding `dsig:Signature` elements by using the `get_xxx` API calls. This means that if a client wishes to verify the answer of a `find_business` inquiry, he/she must retrieve the whole `businessEntity` element, together with the corresponding `dsig:Signature` element, as well as each `businessService` element, together with its `dsig:Signature`

element.

By contrast, if we consider the same API call performed by using the Merkle signature approach, to make the client able to verify the authenticity of the inquiry result it is not necessary to return the client the whole `businessEntity` element and the `businessService` elements, together with their signatures. By contrast, only the Merkle hash values of the missing portions are required, that is, those not returned by the inquiry. These Merkle hash values can be easily stored by the UDDI into the `dsig:Signature` element (i.e., `dsig:SignatureProperties` subelement) of the `businessEntity`.

As discussed above, the main problem in applying the Merkle signature to the `find_XXX` inquiries is that the expected answers, defined by the UDDI specification, do not include the `dsig:Signature` element. For this reason, we need to modify the data structure returned by the UDDI by inserting one or more `dsig:Signature` elements. In particular, to state where the `dsig:Signature` element should be inserted, we need to recall that the `find_XXX` API

calls return overview information taken from different nodes of the `businessEntity` element, and wrapped into a fixed element. For instance, the `find_business` API returns a `businessList` structure, which supplies information about each matching businesses, together with summary information about their services. All this information is wrapped into the `businessInfo` element, which contains the name and the description of the service provider, and a different `serviceInfo` element for each published service.

We can say thus that the `find_XXX` API returns a list of results, each of them wrapped by a precise element (i.e., `businessInfo` for `find_business` API), which will be called, hereafter, *container* element. The proposed solution is thus to insert the `dsig:Signature` element, complemented with the appropriate Merkle hash paths, into each *container* element.

Figure 7 reports an algorithm for generating the answer for a `find_XXX` inquiry. The algorithm receives as input the answer returned according to the UDDI specification, that is, the `xxxList`. Then, in

Figure 7. Computation of `find_XXX` inquiry answers exploiting the Merkle signatures

<p>Algorithm 1</p> <p><i>Input</i></p> <p style="padding-left: 20px;"><code>xxxList</code> the answer of a <code>find_XXX</code> API call</p> <p><i>Output</i></p> <p style="padding-left: 20px;">The <code>xxxList</code> complemented by the <code>dsig:Signature</code> element</p> <ol style="list-style-type: none"> 1. For each container <i>n</i> into the <code>xxxList</code>: <ol style="list-style-type: none"> a. Let <i>MhX</i> be the set of Merkle Hash values associated with the <code>businessEntity</code> to which <i>n</i> belongs to b. Create the <code>dsig:SignatureProperties</code> element using the Merkle hash values in <i>MhX</i> c. Let <i>Sign</i> be the <code>dsig:Signature</code> element of the <code>businessEntity</code> to which <i>n</i> belongs to d. Insert the <code>dsig:SignatureProperties</code> element into <i>Sign</i> e. Insert the obtained <i>Sign</i> element as direct child of the <i>n</i> EndFor 2. Return <code>xxxList</code>

step 1, the algorithm interactively considers each *container* element contained into the *xxxList*, and for each of them it creates the appropriate *dsig:Signature* element. This implies, as a first step, the generation of the Merkle hash values associated with the *businessEntity* element to which the information contained into the container element belongs to. Note that according definition 2, it is not necessary to create all the Merkle hash values; by contrast, the only hash values needed are those corresponding to the nodes pruned during the inquiry evaluation. Then, the obtained hash values are inserted into the *dsig:SignatureProperty* element, according to the strategies illustrated previously. Then, in step 1.d, the resulting *dsig:SignatureProperty* element is inserted into the *dsig:Signature* element generated by the service provider and published together with the *businessEntity* element. Finally, the resulting *dsig:Signature* element is inserted into the *xxxList* as direct child of the corresponding container element.

As an example, let us suppose that a client submits a *find_business* inquiry on the *businessEntity* presented in Figure 2. The answer generated by UDDI according to Algorithm 1 is shown in Figure 8. Given this answer the client is able to verify the Merkle signature generated by the service provider. In order to do that the client exploits the Merkle hash values stored into the *dsig:SignatureProperty* element, which correspond to those nodes of the *businessEntity* not included in the *find_business* answer. In order to compute the Merkle hash value of the *businessEntity* element, and thus to verify the Merkle signature, the client needs to

have all the Merkle hash values of all children of the *businessEntity* element. The *find_business* inquiry returns to the client only the name and description element (see Figure 8). For this reason the *dsig:SignatureProperty* element contains the Merkle hash values of all the remaining children nodes, that is the *discoveryURLs*, the *contacts*, *identifierBag* and *categoryBag* element. Another Merkle hash value needed for the validation of the Merkle signature is the one corresponding to the *businessService* element. The *find_business* inquiry returns only the name of the services (i.e., the name element contained in the *businessService* element), whereas the description, the *bindingTemplate* and the *categoryBag* element are omitted. According to definition 1, to compute the Merkle hash value of the *businessService* element, the client must have the Merkle hash values of all its children. These values are contained into the *dsig:SignatureProperty* element.

PROTOTYPE OF AN ENHANCED UDDI REGISTRY

In this section, we describe the prototype we have developed for implementing a UDDI registry exploiting the Merkle signature technique. The prototype consists of two different components: the UDDI registry, called *enhanced-UDDI registry* and a UDDI client, playing the role of both service provider publishing data to a UDDI, and service requestor inquiring the enhanced UDDI registry.

As reported in Figure 9, the enhanced-UDDI registry is built on top of

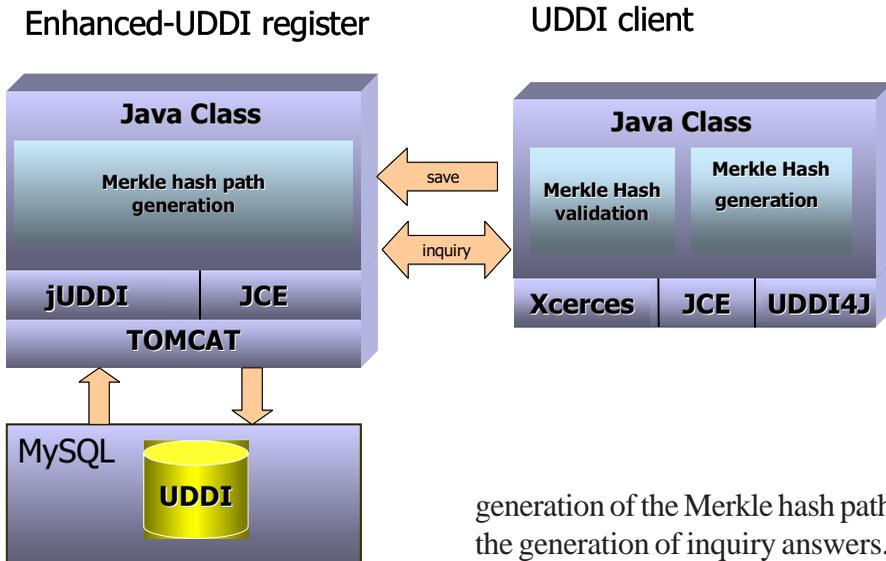
Figure 8. An example of find_xxx answer generated according to the algorithm in Figure 7

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <businessList generic="2.0" operator="jUDDI.org" xmlns="urn:uddiorg:api_v2">
      <businessInfos>
        <businessInfo businessKey="9ECDC890-23EC-11D8-B78C-89A8511765B5">
          <name xml:lang="it">DICO</name>
          <description xml:lang="it">Dipartimento Informatica e Comunicazione</description>
          <serviceInfos>
            <serviceInfo serviceKey="E27F6560-2579-11D8-A560-A95B48063A06">
              <name>Service 1</name>
            </serviceInfo>
          </serviceInfos>
          <Signature>
            <SignedInfo>
              <Reference URI="">
                <Transforms>
                  <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                  <Transform Algorithm="urn:uddi-org:schemaCentricC14N:2002-07-10"/>
                  <Transform Algorithm="Merkle"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/01/xmldsig/sha1"/>
                <DigestValue>CysG5cZQelvxENwHwxBXLMBYGgo</DigestValue>
              </Reference>
              <CanonicalizationMethod Algorithm="http://www.w3.org/1999/07/WD-xml-c14n-19990729"/>
              <SignatureMethod Algorithm="http://www.w3.org/2000/01/xmldsig/rsa"/>
            </SignedInfo>
            <Signature Value=n2XH0Jk6g7jVgGnZxp+7PyBEJhCrVXNx2bdjgzN4zOu1Q52jOfFh3VHMMi6nZsRHHZb5TgqFl
QFgG/Z3JGZJ9P1AWLUVn+kuX1CIZPxKdZ12oe4w/pa/qqXex/K8szgmrBUDIzXNfGEGQIUF+Nbh2WpHK/vUumLNf+Hlg+
jD+StWLTalqIv4jfbdaeEO7EQyiS3AJ+FBYvd7qtArJvzAwAQ8WLI06uprG+
/soHewJLNNgHywPjpSh9FMKraFSyhyjVcrXXgX4Aauv5M3YM6k7ZOEFD0VWVTMk8ukU31rQ9dIPoJvp/aRQPtBb4D
CqD4tM0701s1a6Pxm+8p7IvfvKWWHy3nWNXTLZtGIYssN/BN3clLuiXijW3slaBU=
            </Signature Value>
            <KeyInfo>
              <KeyValue>
                <RSAKeyValue>
                  <Modulus>ALkV0Yv6NSWMQ/GxX7VElnUCmBiBB2kA92iRuXzjr+TesJ6mJWsu
NrQTdaLXNUeLaCfTyibXCHEo8GKhGr3+6UlxknfPbApqRMG2Z6f
                  </Modulus>
                  <Exponent>AQAB</Exponent>
                </RSAKeyValue>
              </KeyValue>
            </KeyInfo>
          </Object>
          <SignatureProperty Target="MerkleHashPath">
            <businessEntity authorizedName="Barbara" operator="jUDDI.org">
              <discoveryURLs hash="sB/kzmjVacE9iBuLdyxC5S2Ha9E="/>
              <contacts hash="bMwPAQ5nAZZhhKcAMswsxDAfPeY="/>
              <identifierBag hash="PF1c19Gspd46sXkdP4f2+i8yajk="/>
              <categoryBag hash="ako/7rv5NZdpx5qjDGQ/W0++acY="/>
              <BusinessServices>
                <BusinessService>
                  <description hash="az8oQfVMxw1C7Dtf5logCtIzNtQ="/>
                  <bindingTemplates hash="cQw/q+Z4IL50QOA/7hj0jnXhkmg="/>
                  <categoryBag hash="ako/7rv5NZdpx5qjDGQ/W0++acY="/>
                </BusinessService>
              </BusinessServices>
            </businessEntity>
          </SignatureProperty>
        </Object>
      </businessInfo>
    </businessInfos>
  </businessList>
</soapenv:Body>
</soapenv:Envelope>

```

Figure 9. The enhanced UDDI registry



jUDDI, which is a Java open source implementation of a UDDI registry. In particular, in the prototype jUDDI exploits a MySQL database (MySQL) as UDDI entries' repository. Moreover, since the latest jUDDI implementation has been developed according to UDDI Version 2 that, unlike the latest specification, does not provide support for the `dsig:Signature` element, we have integrated the prototype also with the IAIK JCE (Java Cryptography Extension) toolkit. This last component makes the prototype able to exploit hash functions, symmetric and asymmetric encryption, and thus to validate the Merkle signature. Thus, in the current version of our enhanced-UDDI registry the standard API functions are implemented by means of jUDDI, whereas the functionalities devoted to the Merkle signature management are implemented by two distinct java classes, directly invoked by jUDDI. These functionalities are the

generation of the Merkle hash paths, and the generation of inquiry answers. More precisely, the last task implies the insertion of the computed Merkle hash path into the `dsig:Signature` element and the insertion of the obtained element into the inquiry answer.

The UDDI client plays the role of both service provider and requestor. To support both these tasks, the UDDI client exploits UDDI4j [UDDI4j], a Java class library providing APIs for interacting with a UDDI registry. UDDI4j supports the UDDI Version 2. For this reason, the UDDI client also makes use of additional Java classes, implementing the functionalities devoted to Merkle signatures management, that is, the Merkle signature generation and the Merkle signature validation. Such classes are directly invoked by the UDDI4j implementation (see Figure 9), and exploit IAIK JCE for signature generation and validation. An example of businessEntity generated by the UDDI client and published to the enhanced-UDDI registry is reported in Figure 10.

Figure 10. The businessEntity element generated by UDDI client

```

<?xml version="1.0" encoding="UTF-8"?>
<save_business xmlns="urn:uddi-org:api_v2" generic="2.0">
  <authInfo xmlns="">authToken:9EF0E0F0-23EC-11D8-B78C-8DDAF5C9614A</authInfo>
  <businessEntity xmlns="" businessKey="9ECDC890-23EC-11D8-B78C-89A8511765B5"
operator="jUDDI.org" authorizedName="Barbara">
    <discoveryURLs>
      <discoveryURL useType="BusinessEntity">http://dico.unimi.it</discoveryURL>
    </discoveryURLs>
    <discoveryURL
useType="businessEntity">http://localhost:8080/juddi/discovery?businessKey=9ECDC890-23EC-11D8-
B78C-89A8511765B5</discoveryURL>
    </discoveryURLs>
    <name xml:lang="it">DICO</name>
    <description xml:lang="it">Dipartimento Informatica e Comunicazione</description>
    <contacts>
      <contact>
        <personName>Barbara Carminati</personName>
        <email>carminati@dico.it</email>
        <address>
          <addressLine>Via Comelico, 39</addressLine>
          <addressLine>20135 Milano</addressLine>
        </address>
      </contact>
    </contacts>
    <businessServices>
      <businessService serviceKey="9ECF4F30-23EC-11D8-B78C-D4B4D63A03DD" businessKey="9ECDC890-
23EC-11D8-B78C-89A8511765B5">
        <name>Service 1</name>
        <description>Example service</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="9ED25C70-23EC-11D8-B78C-E6B2648DFC70" serviceKey="9ECF4F30-
23EC-11D8-B78C-D4B4D63A03DD">
            <description>Binding Example 1</description>
            <accessPoint URLType="www.example.it/service.asmx"></accessPoint>
            <tModelInstanceDetails />
          </bindingTemplate>
        </bindingTemplates>
      </businessService>
    </businessServices>
    <identifierBag />
    <categoryBag />
    <Signature>
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/1999/07/WD-xml-c14n-19990729"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/01/xmldsig/rsa"/>
        <Reference URI="">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
            <Transform Algorithm="urn:uddi-org:schemaCentricC14N:2002-07-10" />
            <Transform Algorithm="Merkle" />
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/01/xmldsig/sha1" />
          <DigestValue>PyAeAtNeYcRQq2gI6Fq7NXOgEnI=</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>pJQn61Vo7ZjzBQNh944I1aMMJPO/ofR16CdHmTNpEYEoI8f3UoDI2OIjR9u+JiBA2MaN7TlwxnKR
ks/mdnWCL85SABOADHwqD1+zoF/VLnaFeGfCJfbWfOTiTNOxjxZFKYISpbfRm6hLFG/qhMb1RRmMp9v+jJKNh00ktpx9Vn
g=</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>ALkV0Yv6NSWMQ/GxX7VElnUCmBiBB2kA92iRuXzjr+TessJ6mJWsuEjWgU2CkezriMRsulMbrGeXb
EORSXluH4VpcE4IYECEb5pheQCeA1eFhDS+BHAXmF1x0sNrQTdaLXNueLaCfTyibXCHEo8GKhGr3
+6UlxkNfPbApqRMG2Z6f </Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</businessEntity>
</save_business>

```

CONCLUDING REMARKS

In this paper we have presented an approach based on Merkle hash trees, which provides a flexible authentication mechanism for UDDI registries.

The proposed approach has two relevant benefits. The first is the possibility for the service provider to ensure the authenticity and integrity of the whole data structures by signing a unique small amount of data, with the obvious improvement of the performance. The second benefit regards browse pattern inquiries (i.e., `find_xxx` API), which return overview information taken from one or more data structures. According to the UDDI specification, in such a case if a client wishes to verify the authenticity and integrity of the answer, it must request the whole data structures from which the information is taken. Besides being inefficient, this solution is not always applicable. Indeed, the information contained in the data structures may be highly strategic and sensitive, and thus may not be made available to all the clients. In such a case, if the client does not have the proper authorization it is not able to verify the authenticity and integrity of the received answer. By contrast, the proposed solution supports the browse pattern inquiries by ensuring at the same time the confidentiality of the data, in that by using Merkle hash paths it is not necessary to send clients the whole data structures.

We plan to extend this work along several directions. One extension regards the support for additional security properties, such as confidentiality and completeness, using strategies similar to those

presented in Bertino et al. (to appear) and an extensive testing and performance evaluation of our prototype.

APPENDIX A

The XML Signature

An XML Signature is an articulated XML object that encodes into an XML format both a digital signature and all the information necessary for its correct verification. The idea is to take advantage of the semantically rich and structured nature of XML to provide a flexible framework for the representation of digital signatures of arbitrarily digital contents. In this context, the W3C XML Signature Working Group (W3C-Sig), in conjunction with IETF, is working on a standard, called XML Signature, with the twofold goal of defining an XML representation for the digital signature of arbitrarily data contents (called *data objects* in what follows), not necessarily XML-encoded data, being at the same time particularly well suited for digitally signing XML documents. In particular, the standard proposal supports three different kinds of signatures, which differ for the localization of the data objects being signed.

Enveloping Signature. The data object is embedded into the XML Signature, in the Object element; thus the Signature is the parent of the signed data.

Enveloped Signature. The data object embeds its signature; thus it is the parent of its Signature.

Detached Signature. The data object is either external data, or a local data object included as a sibling element

Figure 11. Basic structure of an XML Signature

```

<Signature>
  <SignedInfo>
    (CanonicalizationMethod)

    (SignatureMethod)
    (<Reference (URI=)? >
      (Transforms)?
      (DigestMethod)
      (DigestValue)
    </Reference>)+
  </SignedInfo>
  (SignatureValue)
  (KeyInfo)?

  (<Object>
    (SignatureProperties) ?
    (Manifest)?
  </Object>)*
</Signature>

```

of its Signature.

In what follows we give a first brief overview of the essential steps necessary to create an XML Signature. In describing these steps we refer to the basic structure of an XML Signature, reported in Figure 11 (where symbol “?” denotes zero or one occurrences; “+” denotes one or more occurrences; and “*” denotes zero or more occurrences). The first step specifies which are the data objects to be signed. To this purpose, an XML Signature contains a Reference element for each signed data object. The address of the signed data object is stored into attribute URI of the Reference element, through a Uniform Resource Identifier (URI). In the case of enveloping signature, attribute URI is omitted since the data object is contained in the signature element itself, whereas for enveloped signature the URI attribute denotes the element being signed via a fragment identifier. Then, the digest of the data object referenced by attribute URI is calculated and placed into the DigestValue subelement. Information on

the algorithm used to generate the digest is stored into the DigestMethod element. The Reference element may contain an optional Transforms subelement, which specifies an ordered list of transformations (such as for instance canonicalization, compression, XSLT, XPath expressions) that have been applied to the data object before it was digested. For instance, if the URI attribute contains the address of a whole XML document, then the Transforms element can contain an XPath expression identifying selected portions within the document. Such portions are the only ones that are signed.

The next step is to collect all the Reference elements into the SignedInfo element, which contains the information that is actually signed. Before applying the digital signature, the SignedInfo element is transformed into a standard form, called *canonical form*. The aim of such transformation is eliminating from the element additional symbols eventually introduced during the processing of the element (for instance, spaces introduced by an XML

parser and so on) that may cause mistakes during the signature validation process. The algorithms used for the canonicalization are specified in the CanonicalizationMethod subelement. After the canonical form has been generated, the digest of the whole SignedInfo element is computed and signed. The resulting value is stored into the SignatureValue element. Information about the algorithm used for generating the digital signature is contained in the SignatureMethod element. The Signature element can also give the recipient additional information for obtaining the keys to validate the signature. Such information is stored into the optional KeyInfo subelement. The last step is to wrap the SignedInfo, SignatureValue, and KeyInfo elements into a Signature element.

In the case of enveloping signature the Signature element also contains the data being signed, wrapped in the Object subelement.

ENDNOTES

- 1 Given a node w , $Path(w)$ denotes the set of nodes connecting w to the root of the corresponding document.
- 2 In the graph representation adopted in this paper we do not distinguish elements from attributes by treating them as generic nodes.

REFERENCES

Bertino, E., Carminati, B., Ferrari, E., Thuraisingham, B., & Gupta, A. (To appear.). Selective and authentic third-

party distribution of XML documents. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

Charanjit, S., & Yung, M. (1996). *Paytree: Amortized signature for flexible micropayments. Proceedings of the 2nd USENIX Workshop on Electronic Commerce*.

Devanbu, P., Gertz, M., Martel, C., & Stubblebine, S.G. (2000, August). Authentic third-party data publication. *Proceedings of the 14th Annual IFIP WG 11.3 Working Conference on Database Security*, Schoorl, the Netherlands.

The Internet Engineering Task Force. Available: <http://www.ietf.org/>

Merkle, R.C. (1989). A certified digital signature. *Advances in Cryptology-Crypto '89*.

MySQL. Available: <http://www.mysql.com/>

Naor, M., & Nissim, K. (1998). Certificate revocation and certificate update. *Proceedings of the 7th USENIX Security Symposium*.

Universal Description, Discovery and Integration (UDDI). (2002). UDDI Version 3.0. UDDI Spec Technical Committee Specification. Available: <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>

World Wide Web Consortium. Simple Object Access Protocol (SOAP) 1.1. Available: <http://www.w3.org/TR/SOAP/>

World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.2. Available: <http://www.w3.org/TR/wsdl12/>

World Wide Web Consortium. XML Signature Syntax and Processing 2001.

W3C Candidate Recommendation. Available: <http://www.w3.org/TR/2001/CR-xmldsig-core-20010419/>

World Wide Web Consortium. XML Signature Working Group. Available: <http://www.w3.org/Signature/>

Elisa Bertino is professor of Computer Sciences at the CS Department of Purdue University and serves as Research Director of CERIAS. She is also a faculty member at Department of Computer Science and Communication of the University of Milan where she is the director of the DB&SEC laboratory. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Telcordia Technologies. Her main research interests include security, privacy, database systems, object-oriented technology, multimedia systems. In those areas, Prof. Bertino has published more than 250 papers in all major refereed journals, and in proceedings of international conferences and symposia. She is a co-author of the books "Object-Oriented Database Systems - Concepts and Architectures" 1993 (Addison-Wesley International Publ.), "Indexing Techniques for Advanced Database Systems" 1997 (Kluwer Academic Publishers), and "Intelligent Database Systems" 2001 (Addison-Wesley International Publ.). She is a co-editor in chief of the Very Large Database Systems (VLDB) Journal and a member of the advisory board of the IEEE Transactions on Knowledge and Data Engineering. She serves also on the editorial boards of several scientific journals, including IEEE Internet Computing, ACM Transactions on Information and System Security, Acta Informatica, the Parallel and Distributed Database Journal, the Journal of Computer Security, Data & Knowledge Engineering, the International Journal of Cooperative Information Systems, Science of Computer Programming. She has been consultant to several Italian companies on data management systems and applications and has given several courses to industries. She is involved in several projects sponsored by the EU.

Barbara Carminati is an Assistant Professor of Computer Science at the University of Insubria at Como, Italy. Barbara Carminati received a MS degree in Computer Sciences in 2000, and a PhD. in computer science from the University of Milano, in 2004. Her main research interests include: database and web security, XML, secure information dissemination and publishing. She is also a lecturer at the Computer Science School of the University of Milano and University of Insubria at Como, and she has given industrial courses on topics such as database systems and security. Barbara has served as program committee member for the 9th ACM Symposium on Access Control Models and Technologies (SACMAT'04).

Elena Ferrari is professor of database systems at the University of Insubria at Como, Italy. She has also been on the faculty in the Department of Computer Science of the University of Milano, Italy, from 1998 to March 2001. She received a PhD. in Computer Science from the University of Milano, in 1997. Elena Ferrari has been a visiting researcher at George Mason University, Fairfax (VA) and at Rutgers University, Newark (NJ). Her main research interests include database and web security, temporal and multimedia databases. In those areas, Prof. Ferrari has published several papers in all major refereed journals, and in proceedings of international conferences and symposia. She is in the editorial board of the VLDB Journal and the International Journal of Information Technology (IJIT).