Modern Technologies in Web Services Research

Liang-Jie Zhang
IMB T.J. Watson Research Center, USA

IGI PUBLISHING
Hershey • New York

Acquisition Editor:	Kristin Klinger
Senior Managing Editor:	Jennifer Neidig
Managing Editor:	Sara Reed
Assistant Managing Editor:	Sharon Berger
Development Editor:	Kristin Roth
Copy Editor:	Katie Smalley
Typesetter:	Jamie Snavely
Cover Design:	Lisa Tosheff
Printed at:	Yurchak Printing Inc.

Published in the United States of America by
IGI Publishing (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
IGI Publishing (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanonline.com

Library of Congress Cataloging-in-Publication Data

Modern technologies in Web services research / L.J. Zhang.
p. cm.
Summary: "Web service technologies are constantly being recreated, continuously challenging Web service professionals and examiners. This book facilitates communication and networking among Web services and e-business researchers and engineers in a period where considerable changes are taking place in Web services technologies innovation"—Provided by publisher.
Includes bibliographical references and index.
ISBN 978-1-59904-280-0 (hardcover)—ISBN 978-1-5904-282-7 (ebook)
1. Web services. I. Zhang, Liang-Jie.
TK5105.88813.M64 2006
006.7'6—dc22

2006033768

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter II

Authentication Techniques for UDDI Registries

Elisa Bertino, Purdue University, USA

Barbara Carminati, University of Insubria at Varese, Italy

Elena Ferrari, University of Insubria at Varese, Italy

Abstract

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Web services are based on a set of XML standards, such as Web services description language (WSDL), simple object access protocol (SOAP) and universal description, discovery and integration (UDDI). A key role in the Web service architecture is played by UDDI registries, i.e., 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 chapter, 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 wrt authenticity. In the chapter, besides giving all the details of the proposed solution, we show its benefit wrt standard digital signature techniques.

Introduction

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Web services are based on a set of XML standards, such as Web services description language (WSDL) (Christensen, Curbera, Meredith, & Weerawarana, 2001), simple object access protocol (SOAP) (Mitra, 2003), and universal description, discovery and integration (UDDI) (Clement, Hately, von Riegen, & Rogers, 2002). A key role in the Web service architecture is played by UDDI registries. 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 client requirements. In the beginning, UDDI was 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: For a client querying a UDDI registry it should be possible to first verify that the received answer is actually originated at the claimed source, and, then, that the party managing the UDDI registry has not maliciously modified some of answer portions before returning them to the client. To deal with this issue the current version of UDDI specifications allows one to optionally sign some of the elements in a registry, according to the W3C XML signature syntax (Eastlake, Reagle, & Solo, 2001).

UDDI can be implemented according to either a third party or 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, which is responsible for managing (a portion of) the service provider information and for answering service requestors queries. By contrast, in a two party architecture, there is no distinction between the service provider and the discovery agency. Authenticity issues are particularly crucial when UDDI registries are managed according to a third party architecture. For this reason, in the chapter we focus on authenticity issues for third party implementations of UDDI. In this architecture 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 chapter, we propose an alternative approach, previously developed by us for generic XML data distributed, according to a third party architecture (Bertino, Carminati, Ferrari, Thuraisingham, & Gupta, 2004). The main benefit of the proposed solution is that it does not require the discovery agency to be trusted wrt 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. Indeed, since a client may retrieve only selected portions of a document, depending on its needs, it will not be able to validate the signature generated on the whole document. For this reason, we apply an alternative solution that requires the owner to send the discovery agency, 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 discovery agency requiring any portion of the managed data, the discovery agency sends it, alongside the query result, the signatures of the documents on which the query is performed. In this way, the client can locally re-compute the same bottom-up hash value signed by the owner, and by comparing the two values it can verify whether the discovery agency 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 only returned selected portions of a document, they may not be able to re-compute the summary signature, which is based on the whole document. For this reason, the discovery agency 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 chapter, 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 chapter is organized as follows. The "Background" section briefly overviews the authentication mechanisms devised so far for third party architectures and summarizes the basic concepts of UDDI registries. The section "XML Merkle Tree Authentication" presents in detail the Merkle tree-based authentication method conceived for XML documents. In the "Applying the Merkle Signature to UDDI Registries" section, we show how this authentication mechanism can be exploited in the UDDI environment. In "Merkle Signatures vs. XML Signatures in UDDI Registries," we compare our approach with the traditional digital signature techniques. Then, in "Prototype of an Enhanced UDDI Registry," we give some details about the prototype implementation. Finally, "Conclusion" ends the chapter.

Background

In this section we provide the reader with some background that can be useful while reading the chapter. In particular, we start by reviewing the mechanisms introduced so far to face up the authenticity issues in third party architectures. We then summarize the basic concepts of UDDI registries.

Authentication Mechanisms for Third Party Architectures

Mechanisms exploiting traditional digital signatures are not suitable for third party architectures. This is mainly due to the fact that users must be able to validate the owner's signature even if they received a selected portion of the signed data. A naive solution to overcome this problem, by still exploiting traditional signature schemes, is to impose the owner to separately sign each possible portion of data. This set of digital signatures can be outsourced together with the data, and properly returned to users by the third party. However, this solution implies an enormous overhead, both in owner computation and in query answer size. For this reason, in the recent years several alternative strategies have been presented. We can group these strategies on the basis of the underlying adopted techniques. In particular, there are two main exploited techniques, that is, Merkle tree authentication (Merkle, 1989) and signature aggregation (Boneh, Gentry, Lynn, & Shacham, 2003). In what follows we present them by introducing some of the related proposals.

Merkle Tree Authentication

Merkle (1989) proposed a method to sign multiple messages by producing a unique digital signature. The method exploits a binary hash tree generated by means of the following bottom-up recursive construction: At the beginning, for each different message *m*, a different leaf containing the hash value of *m* is inserted in the tree; then, for each internal node, the value associated with it is equal to $H(h\_l||h\_r)$, where $h\_l||h\_r$ denotes the concatenation of the hash values corresponding to the left and right children nodes, and *H()* is an hash function. The root node of the resulting binary hash tree can be considered as the digest of all messages, and thus it can be digitally signed by using a standard signature technique and distributed. The main benefit of this method is that a user is able to validate the signature by having a subset of messages, providing him/her with a set of additional hash values. Indeed, a user, by having hash values of the missing messages, is able to locally build up the binary hash tree and thus to validate the signature.

Merkle hash trees have been used in several computer areas for certified query processing. For instance, they have been exploited by Naor and Nissim (Naor & Nissim, 1998) to create and maintain efficient authenticated data structures holding information about certificate validity. More precisely (Naor & Nissim, 1998), proposed as data structure a sorted hash tree, which is built in such a way that tree leaves correspond to revoked certificates. Thus, verifying that a certificate is revoked or not is equivalent to verify the existence of certain leaves in the tree. Similar schemes have also been used for micro-payments (Charanjit & Yung, 1996), where Merkle hash trees are used to minimize the number of public key signatures that are required in issuing or authenticating a sequence of certificates.

Merkle hash trees have also been exploited for data outsourcing. For instance, (Devanbu, Gertz, Martel, & Stubblebine, 2000) Devanbu et al. adapt Merkle hash trees to the relational data model to prove the completeness and authenticity of query answers. In particular, in Devanbu et al. approach for each relation R a different Merkle hash tree is generated, in such a way that leaf nodes represent hash values of tuples.

Merkle hash trees have been investigated also for third party distribution of XML data (Bertino et al., 2004;Devanbu et al., 2001). Here the challenge is how the XML hierarchical data model can be exploited in construction of Merkle trees. A brief introduction to Merkle tree application to XML documents is given in the "XML Merkle Tree Authentication" section, where we refer interested readers to (Bertino et al., 2004) for a deeper presentation of it.

Signature Aggregation Schemes

Another technique recently exploited to ensure authenticity in third party architectures is based on signature aggregation. In general, signature aggregation schemes allow one to aggregate into a unique digital signature *n* distinct signatures generated by *n* distinct data owners (Boneh et al., 2003). The validation of this unique digital signature implies the validation of each component signature. Aggregate signature schemes have also been investigated to aggregate into a unique signature *n* generated by the same owner. This last kind of aggregation scheme can be adopted to ensure authenticity in third party scenarios. Indeed, according to this solution, data owners could generate a distinct signature for each distinct message and then aggregate them into a unique digital signature. By having only the aggregate signature and by simply validating it, a user is able to authenticate selected messages received by the third party. Such kind of solution has been proposed by Mykletun, Narasimha and Tsudik (2004) for relational data, where two different aggregate signature schemes, namely condensed-RSA and Boneh et al. (2003), have been compared.

UDDI Registries

The main goal of a UDDI registry (Clement et al., 2002) is to supply potential clients with the description of businesses and the services they publish, together with technical information about the services, making thus the requestor able to directly require the service that better fits its needs. The UDDI registry organizes all these descriptions into a single entry.

Figure 1. UDDI main data structutres

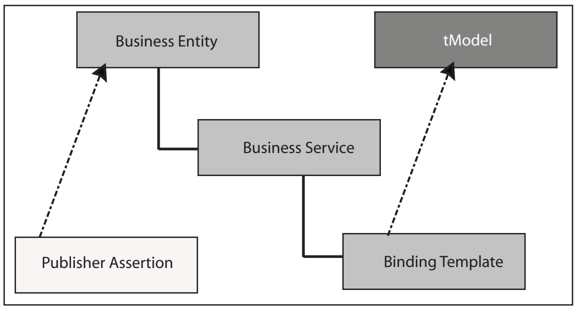


Figure 2. The BusinessEntity element

```
<?xml version="1.0" encoding="UTF-8"?>
 <businessEntity businessKey="9ECDC890-23EC-11D8-B78C-89A8511765B5" operator="jUDDI.org"
authorizedName="Carminati">
  <discoveryURLs>
   <discoveryURL useType="BusinessEntity"> http://www.dicom.uninsubria.it/ </discoveryURL>
   <discoveryURL
useType="businessEntity">http://localhost:8080/juddi/discovery?businessKey=9ECDC890-23EC-11D8-
B78C-89A8511765B5</discoveryURL>
  </discoveryURLs>
  <name xml:lang="it">DICOM</name>
  <description xml:lang="it">Dipartimento d'Informatica e Comunicazione</description>
  <contacts>
   <contact>
    <personName>Barbara Carminati</personName>
    <email>barbara.carminati@uninsubria.it</email>
    <address>
     <addressLine>Via Mazzini, 5</addressLine>
     <addressLine>21100 Varese </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>
```

More precisely, each entry is composed of five main data structures (see Figure 1), namely, the BusinessEntity, the 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, 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 BusinesEntity. 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 tModel 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 DICOM organization (i.e., the name element contained in the BusinessEntity element), which has specified only one contact person, that is, Barbara Carminati (i.e., the personName element contained in the BusinessEntity element). According to Figure 2, the DICOM organization provides only one service, called Service1 (i.e., the name element contained in the BusinessService element), whose binding template is accessible at URL www.example.it/service.asmx (i.e., the accessPoint element contained in the bindingTemplate element).

UDDI registries provide clients searching facilities for provider(s) that better fits 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 approach we propose in Bertino et al. (2004) 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 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 to being signed. This algorithm, which exploits the Merkle tree authentication mechanism (see "Authentication Mechanisms for Third Party Architectures" section), associates a different hash value, called Merkle hash value, with each node (i.e., elements/attributes) in the graph representing an XML document. Before presenting the function computing these Merkle hash values, we need to introduce the notation we adopt throughout the chapter. 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 function*(). Let *d* be an XML document, and *v* a node of *d* (i.e., an element, or an attribute). The Merkle hash value associated with a node *v* of *d*, denoted as $\mathrm{MhX}_d(v)$, is computed by the following function:

$$\mathrm{MhX}_d(v) = \begin{cases} h(h(v.val)\|h(v.name)) & \text{if } v \text{ is an attribute} \\ h(h(v.content)\|h(v.tagname)\|\mathrm{MhX}_d(child(1,v))\|\ldots & \end{cases}$$

$\|\mathrm{MhX}_d(child(Nc_v,v)))$ if *v* is an element 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 an 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, the element 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 untrustworthy third party or an intruder cannot forge the value of the children of node *v*, as well as its content and tagname. Thus, 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 wrt traditional digital signature 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 it 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 comparing it with the received one. We refer to this additional information as the *Merkle hash path*, defined in what follows.

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* belongs to the path connecting *w* to the root, 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*∈ *Path*(*w*)*,* where *Path*(*w*) denotes the set of nodes connecting *w* to the root of the corresponding document. *MhPath*(*w*,*v*) is a list consisting of the following hash values:

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

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. In the graph representation adopted in this chapter we do not distinguish elements from attributes by treating them as generic nodes. In the figure, the triangle denotes the view returned to the client, whereas black circles represent the nodes whose Merkle hash values is returned together with the view, that is, the Merkle hash paths. Consider the first example reported in

Figure 3. Examples of Merkle hash paths

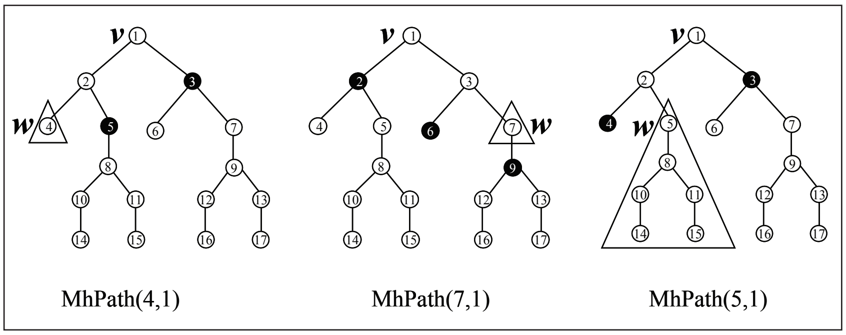


Figure 3. The Merkle hash path between nodes 4 and 1 consists of the Merkle hash values of nodes 5 and 3, plus the hash values of the tagname and content of nodes 2 and 1. Indeed, by using node *w*, the Merkle hash value of node 5*,* and the hash value of the tagname and content of node *2,* it is possible to compute the Merkle hash value of node 2. Then, by using the Merkle hash values of 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 node 9 and the hashed content of node 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 sub-tree 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 sub-tree rooted at 5 is available, it is possible to compute the Merkle hash value of node 5 without the need of 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 sub-tree, 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.

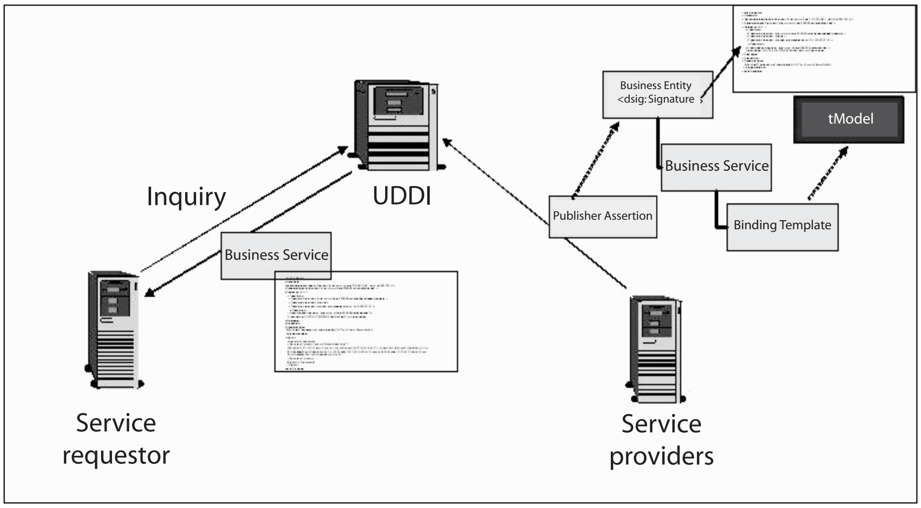
Applying the Merkle Signature to UDDI Registries

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 inquiries 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 to determine 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 (Clement, Hately, von Riegen, & Rogers, 2002). Indeed, to make the service provider able to sign the UDDI entry the latest UDDI specification 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 (Eastlake, Reagle, & Solo, 2001), a service provider can sign the whole element to which the signature element refers to, as well as it can exclude selected portions from the signature, by applying a transformation.

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

Figure 4. The Merkle signature in UDDI environment



the Merkle signature and the Merkle hash paths according to the XML signature syntax (i.e., 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 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.

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> W0uO9b47TqmlpunAwmF4ubn1mdsb4HYR17c+3ULmLL2BxslwSsl6kQ
          </SignatureValue>
</ dsig:Signature>
```

Merkle Hash Path Representation

According to the strategies depicted in Figure 4, once client inquiries 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 kind 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 result. For this reason, we represent also 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 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 attained by adopting in UDDI registries the Merkle signature approach instead of the tra-

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"/>
                        <Transform Algorithm="Merkle"/>
                        </Transforms>
                        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <DigestValue>1fR07/Z/XFW375JG22bNGmFblMY=</DigestValue>
                </Reference>
        </SignedInfo>
        <SignatureValue>
        W0uO9b47TqmlpunAwmF4ubn1mdsb4HYR17c+3ULmLL2BxslwSsl6kQ
        </SignatureValue>
        <Object>
                <SignatureProperties>
                        <SignatureProperty Target="MerkleHashPath">
                                <businessEntity autorizhedName="valore"  operator="juddi.org" hash="sldghoghor....">
                                <discoveryURLs hash="fdsgbdsl...." />
                                <identifierBag hash="57438tgfkv...." />
                                        <categoryBag hash="57438tgfkv...." />
                                        <businessServices>
                                                <businessService>
                                                        <description hash="gherogh..." />
                                                        <bindingTemplates hash="hgkvdlsfv...." />
                                                        <categoryBag  hash="hdsbghfdlb..." />
                                                </businessService>
                                                <businessService>
                                                        <description hash="gherogh..." />
                                                        <bindingTemplates  hash="hgkvdlsfv...." />
                                                        <categoryBag  hash="hdsbghfdlb..." />
                                                </businessService>
                                        </businessServices>
                                </businessEntity>
                        </SignatureProperty>
                </SignatureProperties>
        </Object>
</dsig:Signature>
```

ditional digital signature techniques. Before we do 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 signatures 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 care about the structure of the XML document, ensuring thus 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 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 substitute 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 inquiries, 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, it 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 previously, 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 ore 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 its 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 (i.e., the xxxList). Then, in 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 belong to. Note, that according to 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 1d, 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 Algortihm 1 is shown in Figure 8. Given this answer the client is able to verify the

Figure 7. Computation of find_xxx *inquiry answers exploiting the Merkele signatures*

```
Algorithm 1
Input
      xxxList the answer of a find_xxx API call
Output
   T  he xxxList complemented by the disg:Signature element

1. For each container n into the xxxList:
      a. Let MhX be the set of Merkle Hash values associated with the businessEntity to
         which n belongs to
      b. Create the dsig:SignatureProperties element using the Merkle hash values in
         MhX
      c. Let Sign be the dsig:Signature element of the businessEntity to which n
         belongs to
      d. Insert the dsig:SignatureProperties element into Sign
      e. Insert the obtained Sign element as direct child of the n
   EndFor
2. Return xxxList
```

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 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 into 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

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

```xml
<?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">DICOM</name>
           <description xml:lang="it">Dipartimento d’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>
                   <SignatureValue>n2XH0Jk6g7jVgGnZxp+7PyBEJhCrVXNx2bdjgzN4zOu1Q52jOfFh3VHMMi6nZsRHHZb5TgqFl
           QFgG/Z3JGZJ9P1AWLUVn+kuX1ClZPxKdZ12oe4w/pa/qqXex/K8szgmrBUDIzXNfGEgQIUF+Nbh2WpHK/tVumLNfF+hIg+
           jD+StWLTalqlV4jfJbdaeEO7EQyiS3AJ+FByvd7qtArlJvzAwAQ8WLIO6uprG+
           /soHewJLNNgHywPjpSh9FMKraFSyhyjVcrXXgX4Aauv5M3YM6k7ZOEDfD0WVQTMk8ukbU31rQ9dlPOgJvp/aRQPtBb4D
           CqD4tM0701s1a6Pxmf+8p7IvvfKWWHy3nWNXTLZtGIYssN/BN3clLuiXijW3sIaBU=
                   </SignatureValue>
                   <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="PFIc19Gspd46sXkdP4f2+i8yajk="/>
                                           <categoryBag hash="ako/7rv5NZdxp5qjDGQ/W0++acY="/>
                                           <BusinessServices>
                                                   <BusinessService>
                                                      <description hash="az8oQfVMxw1C7Dtf5logCtlZNtQ="/>
                                                      <bindingTemplates hash="cQw/q+Z4iL50QOA/7hj0jnXhkmg="/>
                                                      <categoryBag hash="ako/7rv5NZdxp5qjDGQ/W0++acY="/>
                                                   </BusinessService>
                                           </BusinessServices>
                                   </businessEntity>
                           </SignatureProperty>
                   </Object>
           </Signature>
       </businessInfo>
      </businessInfos>
    </businessList>
  </soapenv:Body>
</soapenv:Envelope>
```

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 jUDDI, which is a Java open source implementation of a UDDI registry. In particular, in the prototype jUDDI exploits a MySQL database 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 Java cryptography extension (JCE) 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, a Java class library providing APIs for interacting with a UDDI registry. UDDI4j supports the UDDI version 2. For this reason, the UDDI client makes also 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

Figure 9. The enhanced IDDI registry

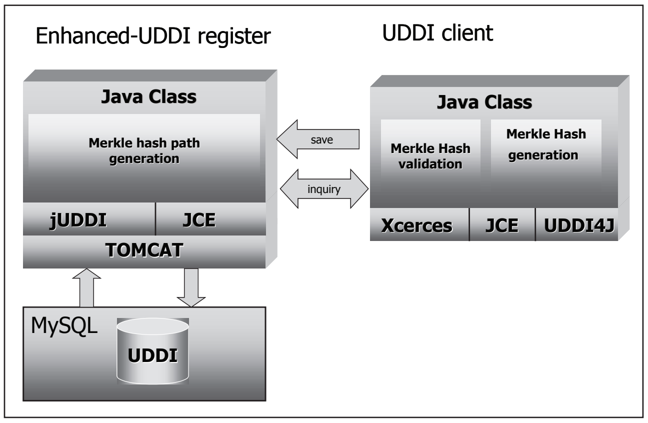


Figure 10. The BusinessEntity *element generated by UDDI client*

```xml
<?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://www.dicom.uninsubria.it/ </discoveryURL>
   <discoveryURL
useType="businessEntity">http://localhost:8080/juddi/discovery?businessKey=9ECDC890-23EC-11D8-
B78C-89A8511765B5</discoveryURL>
  </discoveryURLs>
  <name xml:lang="it">DICOM</name>
  <description xml:lang="it">Dipartimento d'Informatica e Comunicazione</description>
  <contacts>
   <contact>
    <personName>Barbara Carminati</personName>
    <email>barbara.carminati@uninsubria.it</email>
    <address>
     <addressLine>Via Mazzini, 5</addressLine>
     <addressLine>21100 Varese</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/ofR16CdHmTNpEYEoI8f3U0dI2OIjR9u+JiBA2MaN7TlwxnKR
ks/mdnWCL85SABOADHwqD1+zoF/VLnaFeGfCJfbWfOTiTN0xjxZFkYISPbfrM6hLFG/qhMb1RRmMp9v+jJKNh00ktpx9Vn
g=</SignatureValue>
   <KeyInfo>
    <KeyValue>
     <RSAKeyValue>
      <Modulus>ALkV0Yv6NSWMQ/GxX7VElnUCmBiBB2kA92iRuXzjr+TesJ6mJWsuEjWgU2CkezriMRsu1MbRGeXb
E0RSXluH4VPcE4IYECEb5pheQCeA1eFHdS+BHAXmFIx0sNrQTdaLXNUeLaCfTyibXCHEo8GKhGr3
+6UlxkNfPbApqRMG2Z6f </Modulus>
      <Exponent>AQAB</Exponent>
     </RSAKeyValue>
    </KeyValue>
   </KeyInfo>
  </Signature>
 </businessEntity>
</save_business>
```

by the UDDI client, and published to the enhanced-UDDI registry is reported in Figure 10.

Conclusion

In this chapter 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 are taken. Besides being not efficient, 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 for instance confidentiality and completeness, using strategies similar to those presented in Bertino, et al. (2004) and an extensive testing and performance evaluation of our prototype.

References

Bertino, E., Carminati, B., Ferrari, F., Thuraisingham, B., & Gupta A. (2004). Selective and authentic third-party distribution of XML documents. *IEEE Transactions on Knowledge and Data Engineering(TKDE)*, *16*(10), 1263-1278.

Boneh, D., Gentry, C., Lynn, B., & Shacham, H. (2003). Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Advances in Cryptology, International Association for Cryptologic Research*. Berlin, Germany: Springer-Verlag.

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

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web services description language (WSDL), Version 1.2* (World Wide Web Consortium recommendation). Retrieved from http://www.w3.org/TR/wsdl12/

Clement, L., Hately, A., Von Riegen, C., & Rogers, T. (2002). *Universal description, discovery and integration (UDDI), Version 3.0* (UDDI Spec Technical Committee specification). Retrieved from http://uddi.org/pubs/uddi-v3.00-published-20020719.htm

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

Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., & Stubblebine, S.G. (2001). Flexible authentication of XML documents. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*. ACM Press.

Eastlake, D., Reagle, J., & Solo, D. (2001). *XML signature syntax and processing 2001* (World Wide Web Consortium Recommendation). Retrieved from http://www.w3.org/TR/2001/CR-xmldsig-core-20010419/

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

Mitra, N. (2003). *Simple object access protocol (SOAP), Version 1.1* (World Wide Web Consortium recommendation). Retrieved from http://www.w3.org/TR/SOAP/

Mykletun, E., Narasimha, M., & Tsudik, G. (2004), Authentication and integrity in outsourced databases. In *Proceedings of Network and Distributed System Security (NDSS 2004).*

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