COLLECTIVE SIGNATURE FOR EFFICIENT AUTHENTICATION OF XML DOCUMENTS

Indrajit Ray and Eunjong Kim
Department of Computer Science, Colorado State University
Fort Collins, CO – 80523, USA.
{indrajit, kimeu}@cs.colostate.edu

Abstract: XML (eXtensible Markup Language) is the de-facto standard for document representation and exchange on the Web. Researchers have previously proposed access control models and schemes for XML documents that allow one to disseminate selectively, portions of an XML document to the user community based on different policies. Such selective dissemination of an XML document creates a new problem, namely, how to authenticate portions of the XML document independent of other portions and/or the complete document. In this paper, we present a novel scheme based on one-way accumulator functions that allows the user to have a guarantee that a portion of an XML document does indeed belong to the original document.

Key words: authentication, XML documents, message digests, digital signatures

1. INTRODUCTION

Extensible Markup Language (XML) is fast replacing Hyper Text Markup Language (HTML) as the de-facto standard for information representation and exchange on the Web. A major advantage of XML over HTML, which contributes to the former's popularity, is the provision for defining tags, nested document structures, and document types; these features together not only allows efficient representation of document content but also document structure, thus allowing increased control of the granularity of information that can be disseminated over the Web.

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

As the information size in XML format grows, we can expect these documents to be too large to be retrieved at one time. In fact, almost always will portions of an XML document be retrieved at any given instance. Consider for example an XML document that contains the most recent information about international trading practices and customs regulations for various countries. Such a document may, very well be extended over several thousand of web pages. Suppose now a client wants information about a specific import regulation at the New York harbor for imports from France. The client will obviously want only the relevant portions delivered to him or her and not the entire document. Thus, the server that hosts the information (or is responsible for delivering the information) will only send a small portion of the XML document to the client. However, the client, no doubt, will like to have the guarantee that the information delivered to him or her is from the original document containing the regulations. This will enable the client to have the most up-to-date information from the right sources.

There are other reasons also why an XML document often needs to be delivered in bits and pieces with each piece being delivered independent of other pieces. As pointed out by Bertino and Ferrari [7] it is often the case that in large organizations the same XML document is often shared among many users. Such an XML document may contain information of different sensitivity degrees and thus the document may need to be selectively disseminated among these users. Different access control policies are applied to different components within the same XML document to retrieve the various portions, which are then independently pushed to individual users.

For all these cases, the client gets only portions of the original XML document at a time. This mode of content delivery has some serious integrity issues particularly in critical areas such as government, health, finance, law, etc. The integrity of these individual portions needs, no doubt, to be ensured in transit. This is easily achieved by cryptographic techniques such as message authentication codes and digital signatures. The bigger problem is ensuring the authenticity of each individual portion. How can a recipient have the guarantee that a portion of an XML document that is received belongs to the original document?

The central goal of this paper is to allow verification of answers to queries of a small portion from a large XML document, without searching for or worrying about the remaining portions of the document. As discussed later in section 2, this problem has been previously addressed by other researchers. In this paper, we propose a new approach to signing XML documents based on the notion of one-way accumulator functions. We introduce the concept of collective signatures to sign each portion of the document. This technique allows the recipient of a portion of the XML

document to verify the integrity of the portion as well as its authenticity in an efficient manner.

The rest of the paper is organized as follows. Section 2 provides some background information about the technologies that are need. Section 3 discusses some of the more important related works. Section 4 presents our approach. We conclude in Section 5 by discussing our future work.

2. BACKGROUND

Some of the well-recognized benefits of using XML as a data container are its simplicity, richness of the data structure, and excellent handling of international characters [13].The basic concept of an XML document is that an element can be nested to any depth and can contain other sub-elements. An element contains a portion of the document delimited either by a pair of start and end tags (e.g., <tag-name> and </tag-name>). A document is represented as an ordered, node-labeled tree. The conventional terminology for XML document is found in W3C proposals such as, XML Information Set [8] or Xpath [9].

We adopt the example of a typical XML document from [7]; the example is shown in Figure 1. The document is a bulletin that provides information on trading legalities all over the world. It contains a special section, that is, the Import-Export, for international trading legislation. For each law, the document provides information on the country that issues the law, on the law topic, and on related laws.

The start tag of each element can specify a list of attributes providing additional information for the element. Attributes are of the form, *name* = *attvalue*, where *name* is a label and *attvalue* is a string delimited by quotes. Attributes can have different types allowing one to specify the element identifier, additional information about the element, or links to other elements of the document. To be referenceable, an element must have an attribute of type ID whose value provides a unique identifier. Each element can be empty and empty elements are characterized only by a start tag and do not contain data content or sub-elements.

An XML document is often represented as a labeled graph where nodes represent elements and attributes, and edges represent relationships between them. A node representing an element contains the element identifier (*id*). An element identifier can be the ID attribute value associated with the element, or can be automatically generated by the system, if no attribute of type ID is defined. A node representing an attribute contains its associated value. For simplicity, data content of an element is represented as a particular attribute whose name is content and whose value is the element

data content itself. The graph contains edges representing the element-attribute and the element-sub-element relationships, and link edges representing links between elements. Edges are labeled with the tag of the destination node and are represented by solid lines, whereas link edges are labeled with the name of the corresponding attribute and are represented by dashed lines.

```
<InternationalLaw Date="11/20/2003">
    <Law Country="USA" RelatedLaws="LK75">
        <Topic> Taxation</Topic>
        <Summary>....</Summary>
    </Law>
    <Law Id="LK75" Country="USA">
        <Topic> Import-Export </Topic>
        <Summary>.....</Summary>
    </Law>
    <BluePageReport>
        <Section GeoArea="Europe">
            <Law Country ="Germany">
                <Topic> Guns </Topic>
                <Summary>.....</Summary>
            </Law>
        .....
        </Section>
    </BluePageReport>
</InternationalLaw>
```

Figure 1. An example XML Document

Figure 2 shows the graph representation of the XML document in Figure 1. In the figure, nodes representing elements are denoted by white ovals, whereas nodes representing attributes are denoted by gray ovals. Nodes representing elements have the corresponding *id*. For meaningful exchange of XML documents and formal description of admissible structures of XML documents, a document type definition (DTD) is typically associated with a collection of XML documents. DTD specifies the rules that XML documents must follow.

A DTD consists of declarations for elements, attributes, notations, and their content. It is composed of two parts: the element declarations and the attribute list declarations. The element declarations part specifies the structure of the elements contained in the document. The attribute list

declarations part specifies, for each element, the list of its attribute names, types, optional clauses, and default values. Path queries are used to address subtree structures of an XML document using regular expressions over XML element names. DTDs can be represented as a graph using a representation similar to those of XML documents. An XML source is a set of XML documents and DTDs.

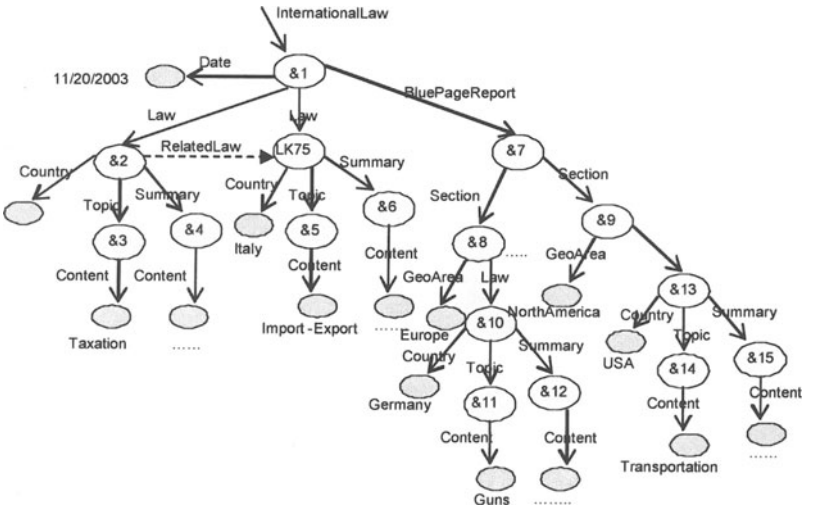

Figure 2. Graph representation of the XML document from Figure 1

3. RELATED WORK

As described in section 2, XML documents have a simple data model based on trees. DOM [6], which is a tree-like representation, is a standardized application programming interface that defines how XML documents are to be accessed by programs. DOM specification includes a hashing procedure very similar to Merkle-hashing. DOMHASH provides a concrete definition of the hash value calculation. The procedure "hashes the leaves of the document, and recursively proceeds up the document tree, hashing both the element types as well as the elements within the document" [1].

The DOMHASH process works as follows. A Merkle hash tree [10] associates hash values with nodes in a tree. The leaves get the hash of the values, and each interior node gets the hash of all the values of all its

children, combined in a suitable way. Once the root digest is authenticated, anyone can certify answers to queries, using only the hash values to provide evidence of a correctly conducted search. If the root hash of an entire document D is known to a party P, it is possible to provide evidence to P that any subtree τ of the document occurs under D without revealing all of D. For this, P can DOMHASH the subtree τ to get the root hash of τ. Then P can be given just the hash value of the siblings of τ and the siblings of all its parents and P can recompute the root hash of D. Figure 3 shows how it works. P is given the hash value of τ (for example, 23). If P has the hash values of sibling of τ (in this case, $h(34)$) and the siblings of all its parents ($h(h(312), h(1123))$ and the value *hl (hash of the left sub-tree)*, P can recompute the root hash of D and thus be sure that the value 23 did occur in the tree. P can be assured the hash values cannot be forged because the hash function h is one-way. Once a client has a secure way of obtaining the root hash of this structure, it is also possible to prove credibly that the answers to the selection queries are complete.

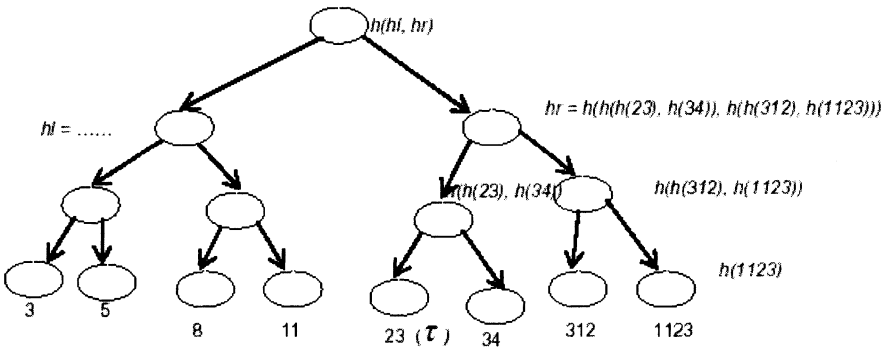

Figure 3. Merkle hash tree associates hash values with nodes in a tree

These techniques are useful in the context of XML documents. However, note that XML data is not always as structured, and requires some additional mechanism to certify answers to the queries. Another drawback of the DOMHASH technique is that it requires additional information about a node's siblings and parents.

Devanbu [1] proposed an approach to signing XML documents, which allow untrusted servers to answer certain types of path queries and selection queries over XML documents without the need for trusted on-line signing keys. This approach enhances both the security and scalability of publishing information in XML format over the Internet. In addition, it provides

flexibility in authentication parts of XML documents. However, in this approach a server and a client have to keep track of all path information to get the answer for each path query as above. Although the authors suggest a new data structure, an *xtrie*, to store the different possible answers to path queries, this is a cumbersome procedure and in some cases, it is a very inefficient way to access the data.

Steinfeld and Bull [11] introduced the Content Extraction Signature (CES) as a new type of digital signature. The use of CES with documents enable a user to handle and process these statements in a more selective manner which enable selective disclosure of verifiable content by providing dynamic loading of the transforms for signing and verifying documents. It enables the user to embrace a minimal information model whereby all of the extra details that are not required can be omitted from the transaction. This approach gives some ideas for using access control model with one-way accumulator.

4. OUR APPROACH

We begin the discussion of our approach to generating signatures for XML documents by introducing the notion of one-way accumulators. A one-way accumulator function is a special type of hash function. Hash functions [3] are generally used to reduce their arguments to a pre-determined size. Therefore, it consumes arbitrary length of input and generates a fixed length of the output. *One-way hash functions* requires that collisions of the form $h(x, y) = h(x', y')$ for given x, y, and y' are hard to find. This means, given, x, y, y', it is generally hard to find a pair $(x', y') \in X \times Y$ such that $h(x, y) = h(x', y')$.

One-way accumulator [2] is a simple one-way hash function which satisfies the *quasi-commutative* property. A function h: $X \times Y \rightarrow X$ is said to be quasi-commutative if for all $x \in X$ and for all y_1, $y_2 \in Y$, $h(h(x,y_1),y_2) = h(h(x,y_2),y_1)$.

As defined by Benaloh and Mare [2], a family of one-way accumulators is a family of one-way hash functions each of which is *quasi-commutative*. From above property, if one starts with an initial value $x \in X$, and a set of values $y_1, y_2 \ldots y_m \in Y$, then the accumulated hash $z = h(h(h(\ldots h(h(h(x,y_1),y_2),y_3),\ldots,y_{m-2}),y_{m-1}),y_m)$ would be unchanged if the order of the y_i were permuted. This forms the intuitive basis of our approach.

Suppose an XML document stored at a server has a set of values $Q = (q_1, q_2, \ldots, q_n)$. The server chooses an intial value x_0. Using an one-way accumulator function $h(x,q)$, the server can

compute $x_i = h(x_{i-1}, q_i)$, where $i \in [1, n]$. Owing to the quasi-commutative property, the result $A = x_n$ is independent of the order of the q_i's. The server calculates the signature of the XML document as $\lceil A, S_{prv} \rceil$, where S_{prv} is the private key of the server. For any value q_i, the server can compute a *witness* A_i given by: $A_i = h(h(\mathrm{K}\ h(x_0, q_1), \mathrm{K}\ , q_{i-1}, q_{i+1})\mathrm{K}\ , q_n)$.

When the server needs to send q_i to a client such that the client wants to verify the authenticity of q_i, the server basically sends A_i, q_i and the signature of the entire XML document, $\lceil A, S_{prv} \rceil$. The receiver computes $A' = h(A_i, q_i)$ and compares it with the value A that is obtained by decrypting the signature $\lceil A, S_{prv} \rceil$ using the server's public key.

We are now ready to provide complete details of our approach for collective signatures.

4.1 Collective Signature Generation

We begin by providing the details regarding generating the accumulated hash using a one-way accumulator function. The most well-known one-way accumulator function to date is based on modular exponentiation techniques similar to RSA public key cryptography. Benaloh and Mare were the first to introduce this technique [2]; it was later improved by Nyberg [13], Baric and Pfitzmann [14], Gennaro, Halevi and Rabin [15], Sander [16] and Sander, Ta-Shma and Yung [17]. We use an approach similar to the one proposed by Benaloh and Mare which uses the exponential accumulator, $\exp(y, x) = y^x \bmod N$, for suitably chosen values of y_0 and the modulus N. In particular, choosing $N = p \bullet q$ for two primes p and q, makes the exponential accumulator function as difficult to break as the RSA cryptosystem. However, as pointed out by Goodrich et al. [18], the exponential accumulator is not associative; any updates require significant recomputations. Goodrich et al. provide an optimized solution based on the notion of *universal-2 hash functions*, which we adopt. Briefly, a family $H = \{h : A \rightarrow B\}$ of functions is *universal-2* if for all $a_1, a_2 \in A$, $a_1 \neq a_2$ and for a randomly chosen function h from H

$$\Pr_{h \in H}\{h(a_1) = h(a_2)\} \leq \frac{1}{|B|}$$

The following steps are involved in the computation of an RSA based one-way accumulator. Let the source (that wants to generate the one-way accumulator) contain data set S consisting of n k-bit elements; that is $S = \{e_1, e_2, \mathrm{K}\ , e_n\}$.

1. The source chooses two strong primes p and q such that $p, q > 2^{3/2k}$.

2. The source also chooses a large base a that is relatively prime to $N = p \times q$.
3. Finally the source chooses a random function h from a universal-2 hash family and publishes a, N and h. The source keeps p and q secret.
4. For each element e_i of S, the representative x_i is computed such that $h(x_i) = e_i$ and x_i is a prime.
5. The accumulation is then computed as $A = a^{x_1 x_2 \mathrm{K}\, x_n} \bmod N$. The maximum size of A is the size of N, which is at least $3k$ bits long.
6. For each element e_i the *witness* A_i is given by $A_i = a^{x_1 x_2 \mathrm{K}\, x_{i-1} x_{i+1} \mathrm{K}\, x_n} \bmod N$.

To verify if a given e_i belongs to S the receiver needs to know the corresponding x_i, A_i and A that is signed by the source. It computes $A' = \left(A_i^{x_i} \bmod N\right)$. If $A' = A$ then e_i is verified to belong to S.

The only problem to this scheme that prevents its use in XML document verification is the assumption that each element e_i is of equal length and fixed in size. However, this is usually not the case. To solve this we replace the XML document element e_i by its corresponding digest created through a well-known cryptographically strong hash algorithm like SHA or MD5. The complete algorithms are given in the following section.

ALGORITHM 1: Accumulated signature generation

Input: An XML document D which consists of sub-documents d_i.
A cryptographically strong hash function h' to generate message digests.
A key pair $\left(K_{prv}, K_{pub}\right)$ which represents the public-private key pair for the server.

Output: A universal hash function h.
A table τ with the tuples (sub-document d_i, hashed value e_i, pointer to representative value x_i)
A binary tree τ' whose leaf nodes are the pairs $\left(e_i, x_i, A_i\right)$, where x_i is the representative for e_i, and A_i is the corresponding witness for x_i.
An accumulated hash value A for the entire document.

Steps:

1. For each leaf node d_i of the XML tree for D, compute a fixed size (say 128 bit or 160 bit) message digest $e_i = h'(d_i)$ where h' is a cryptographically strong hash function like SHA or MD5. Let the

size of each e_i be k bits. Store the result in a table τ, sorted against the value of e_i. Choose values a, p, q and N as discussed earlier. Choose also a universal-2 hash function, h. Publish a, N and h.

2. For each hashed value e_i generated in step 1, compute the representative value x_i of e_i.
3. Construct a complete binary tree τ', so that each leaf node of the tree contains a representative value x_i for e_i. Store the value (e_i, x_i) in the leaf node. Leave the internal nodes empty for the time being. Adjust the corresponding pointer in the table τ to point to point to this leaf node.
4. Perform a post-order traversal of τ' visiting each node after having visited all its children. Stop when the root node is traversed. When a node u is reached in the traversal process compute a value $\Omega(u)$ as follows:
    a. If u is a leaf node and u has a value x_i compute $\Omega(u) = x_i \bmod \phi(N)$ where $\phi(\bullet)$ represents the Euler totient function. Store $\Omega(u)$ temporarily in the location for A_i in the node.
    b. If u is not a leaf node and v and w are its children, compute $\Omega(u) = \Omega(v)\Omega(w) \bmod \phi(N)$. Store the value $\Omega(u)$ in the internal node.
5. Perform a pre-order traversal of the tree τ'. For each node u that is traversed, compute the accumulation $A(u)$ of all values that are not stored in any descendant of u as follows:
    a. If r is the root, $A(r) = a$. Store the value signed by the server's private key K_{prv}.
    b. If $u \neq r$, $A(u) = A(g)^{\Omega(w)} \bmod N$, where g is the parent node of u and w is its sibling. Replace the previous value $\Omega(u)$ by $A(u)$. It can be shown trivially that each $A(u)$ that is generated is the witness A_i for x_i.

Stop when A_i is computed for all leaf nodes.

ALGORITHM 2: Query Processing

Input: The XML document D, table τ and binary tree τ' generated in algorithm 1, and a query q for sub-document d_i from a client

Output: A message M which consists of the document d_i resulting from the query, the hash value $h'(d_i) = e_i$ corresponding to d_i, the

representative value x_i, the witness A_i corresponding to e_i and the signed accumulated hash value $\left[A, K_{prv}\right]$.

Steps:

1. Retrive the sub-document d_i from the XML document D, based on the query q.
2. Compute the message digest $h'(d_i) = e_i$ for the subdocument d_i just retrieved.
3. Perform a binary search on the table τ to locate entry e_i. Let the entry be in row r.
4. Follow pointer in row r to get to a leaf node in the binary tree τ'.
5. Retrieve the values (e_i, x_i, A_i).
6. Visit the root of the tree τ' to get the signed accumulated value $\left[A, K_{prv}\right]$.
7. Concatenate the values retrieved in steps 1-6 and create a message M.
8. Generate a message digest M_d for this whole message M.
9. Send M and M_d as answer to the query.

ALGORITHM 3: Answer Verification

Input: A message $M = \left(d_i, e_i, x_i, A_i, \left[A, K_{prv}\right]\right)$ which is the result of a query on a document D, together with a digest M_d of M.
Values N, functions h (the universal-2 hash function) and h' (to compute simple message digests) and K_{pub}, the server's public key.

Output: A yes/no answer depending on whether or not the sub-document d_i belongs to the document D.

Steps:

1. Compute the hash value $h'(M) = M'_d$ of M. If $M'_d = M_d$ proceed to the next step, else return with an answer no.
2. Retrieve and decrypt $\left[A, K_{prv}\right]$ from the message M using the public key K_{pub} of the server. If the signature verifies successfully (that is the decryption is successful) proceed to next step; else retrun with no.
3. Compute $A' = \left(A_i^{x_i} \bmod N\right)$. If $A' = A$ return with answer yes; else return with answer no.

Note that if the XML document is static, the query process incurs very little overhead compared to a standard query, which does not require such

features. The table τ and the binary tree τ' can be computed once during generation of the XML document itself. Providing the values is independent of the size of the XML document.

5. CONCLUSION AND FUTURE WORK

The main goal of this paper is to suggest an efficient way to disseminate a portion of an XML document so that the receiver is assured about the origination of the sub-document and its authenticity and at the same time has the assurance that the document is actually a portion of the original document. The information that is provided to the user to achieve this is only what is related to the user's request and nothing else. This is unlike in previous schemes. The protocol uses one-way accumulator for accumulated hash value, which is formed from a given set of values in entire document. This protocol is based on two properties of one-way accumulator; *quasi-commutative* and *one-wayness.*

At this stage of our work, we have assumed that the XML document is static. Thus, all necessary values can be precomputed. In future, we plan to investigate algorithms to efficiently compute these values when the XML document is changing. We also plan to investigate faster algorithms to compute the accumulator functions.

ACKNOWLEDGMENT

This work was partly supported by the U.S. National Science Foundation under award number IIS-0242258. Any opinion, findings and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF. The authors would like to thank the anonymous reviewers for their useful comments and suggestions.

REFERENCE:

[1] P. Devanbu, M. Gertz, A. Kwong, C.Martel, G.Nuckolls, S. G. Stubblebine. "Flexible Authentication of XML Document," *Proceedings. of the 8th ACM conference on Computer and Communications Security,* pp. 136-145, November 2001.

[2] J.C. Benaloh and M. de Mare. "One-way accumulators: A Decentralized Alternative to Digital Signatures," *Proceedings of*

EUROCRYPT'93: Advances in Cryptology, Lecture Notes in Computer Science, vol. 765, pp 274-285, Springer-Verlag, 1993.

[3] R. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *Proceedings of Crypto '87: Advances in Cryptology,* Lecture Notes in Computer Science, vol. 293, pp.369-378, Springer-Verlag, 1987.

[4] D. Eastlake, J.Reagle and D. Solo, "XML-Signature Syntax and processing", W3C Recommendation, February 2002.

[5] A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Random Sequences." *Proceedings of the 8th Colloquium on Automata, Languages and Programming ICALP '81*, Lecture Notes in Computer Science, vol. 115, pp 544-550, Springer-Verlag, 1981.

[6] Digest Values for DOM (DOMHASH). RFC 2803 Internet Society, available from http://www.faqs.org/rfcs/rfc2803.html, April 2000

[7] Elisa Bertino and Elena Ferrari, "Secure and Selective Dissemination of XML Documents," *ACM Transactions on Information and System Security*, Vol. 5, No. 3, August 2002, pp. 290-331

[8] J. Cowan, "XML Information Set". W3C Recommendation, October 2001.

[9] J. Clark and S. DeRose, "SML Path Language (XPath)". W3C Recommendation, November 1999.

[10] R. Merkle, "A Certified Digital Signature". *In Advances in Cryptology-CRYPTO'89.* Lecture Notes in Computer Science, vol. 435, pp 234-246, Springer-Verlag 1989.

[11] R. Steinfeld, L. Bull and Y. Zheng, "Content Extraction Signatures", *Proceedings of the 4th International Conference on Information Security and Cryptology (ICISC 2001)*, Lecture Notes in Computer Science, vol. 2288, pp. 285-304, Springer-Verlag 2001.

[12] H. Maruyama, K. Tamura, and N. Uramoto, "XML and Java, Developing Web Applications," Addison Wesley 1999.

[13] R. Nyberg, "Fast Accumulated Hashing", *Proceedings of the 3rd International Workshop on Fast Software Encryption.* Cambridge, U.K. 1996. Published as D. Gollman editor, Lecture Notes in Computer Science, vol. 1039, pp 83-87, Springer-Verlag 1996.

[14] N. Baric and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees", *Proceedings of EUROCRYPT '97: Advances in Cryptology.* Lecture Notes in Computer Science vol. 1233, pp 480-494. Springer-Verlag 1997.

[15] R. Gennaro, S. Halevi and T. Rabin, "Secure Hash-and-Sign Signatures Without the Random Oracle" *Proceedings of EUROCRYPT '99: Advances in Cryptology.* Lecture Notes in Computer Science, vol. 1592, pp 123-139, Springer-Verlag 1999.

[16] T. Sander, "Efficient Accumulators Without Trapdoor", *Proceedings of the 2nd International Conference on Information and Communications Security 1999 (ICICS 1999)*. Lecture Notes in Computer Science, vol. 1726, pp 252-262, Springer-Verlag 1999.

[17] T. Sander, A. T-Shma and M. Yung, "Blind, Auditable Membership Proofs", *Proceedings of Financial Cryptography '00, 2000.* Lecture Notes in Computer Science, vol. 1962, pp 53-71, Springer-Verlag 2000.

[18] M. T. Goodrich, R. Tamassia and J. Hasic "An Efficient, Dynamic and Distributed Cryptographic Accumulator", *Proceedings of the 5th International Conference on Information Security, 2002.* Lecture Notes in Computer Science, vol. 2433, pp 372-388, Springer-Verlag 2002.