Integrity verification for XML data

Jules R. Nya Baweu and Huiping Guo *

Abstract—**The success of the Internet has made the communication very easy between parties and XML is one of the most used standard for information representation and exchange. A huge amount of data is crossing the Internet on the daily basis. Since the Internet is an unreliable network, XML data can be easily captured and tampered by unauthorized users. This study is proposing a scheme that uses fragile watermarking to detect and localize any modification made to an XML data. This scheme is distortion free since it doesn't introduce any distortion to the XML data. This scheme uses an algorithm that distributes elements of the XML data into groups in a secure manner according to certain parameters. The watermark is verified without the need to have the initial XML data, and also, the embedding and the verification process in each group are independent. Experimental results and security analysis show that the chance to detect and locate change in the XML data is very high.**

Keywords: XML, security, integrity, watermarking

1 Introduction

XML (eXtensible Markup Language) was created by the W3C [1] to overcome the limitation of the HTML. It is being used for large web site maintenance, exchange of information between organizations, database population, scientific applications, electronic books, handheld devices, electronic commerce application and more [13]. As of today, the most standard used for information representation and exchange over the Internet is XML. A considerable amount of sensitive and secret information is being exchanged between several parties on the daily basis across the Internet using XML. The Internet is an unreliable network where malicious users can easily modify and duplicate data. The huge popularity of the XML standard over the Internet comes along with a major concern for the unauthorized modification, duplication and distribution of XML data. Digital watermarking, which is a technology to embed information into digital data in a obscure, unknown, and unremarkable way, provides an artery of protecting XML data from illegitimate modification, duplication and distribution [11, 3]. Digital Watermarking is divided into two categories according to the way it is applied: fragile watermark and robust watermark. Fragile watermark is used for tamper detection while robust watermark is applied for ownership verification.

Several works were done on digital watermarking and several proven technologies were verified and appeared to be efficient. However most of the works were focused on digital watermarking of multimedia data; for example in [4]. As of today, few works have been done on non-multimedia content.

Recently, the idea to apply digital watermarking to non-multimedia data has been brought to attention. Mostly oriented in the robust watermarking direction with the goal to protect copyright of goods, we will now see software digital watermarking [2], database digital watermarking [5, 6] and XML watermarking [6, 13, 8].

Even though the need to apply digital watermarking to non-multimedia became persistent, fragile watermarking of those contents was almost left aside. we will note database fragile watermarking [9] and streaming data watermarking [10].

As we say above, some researches were actually done on watermarking XML data, but most of them were focusing on robust watermarking of XML [13, 8].

The following work presents a Tamper Detection And Localization For XML Data using Fragile Watermarking.

2 Related work

In [13], the scheme proposed is called WmXM. WmXM is described as a system for watermarking XML, which generates queries from essential semantics to identify the available watermarking bandwidth in XML documents. As further described, it uses query templates to represent data usability, generate queries from semantic, which allows elements identification and structure units for watermarking. It also considers the internal semantics to walk through vulnerabilities that come with redundancy found in XML data. This watermarking scheme starts with the initialization, where a schema is specified and XML data are validated . A set of query templates is specified to represent data usability. Using a secret key, a number of data elements or structure units is selected for the watermark embedding. Queries are then created. The process ends up with the Watermark Insertion. Just as above in [8], we note that this technique falls into ro-

*Department of Computer Science, California State University at Los Angeles, Los Angeles, CA 90032 Email: hpguo@calstatela.edu

bust watermarking.

In [15], a scheme for watermarking semi-structured data that uses a graph labeling is proposed; As other schemes mentioned before this one also falls into robust watermarking.

In [9], a scheme for fragile watermarking is proposed. The technique used here is a proven method for detecting and localizing malicious alterations made to a database relation with categorical attributes. Furthermore, the scheme is distortion free, which means there is no distortion made to the actual data. It prevents illegal embedding and verification, since the watermarking process is governed by a key. The watermark embedding process and verification can only be done by a person who possesses the necessary key. The original database relation with no embedded watermark is not required for watermark verification. Finally, in case the watermark verification fails, the embedded watermarks indicate where the modifications are. The algorithm used in this scheme can be resumed as follow: In the watermark embedding phase, first a primary key hash value is calculated for each tuple according to a watermark key and the primary key of the tuple. According to their primary key hash value, all tuples are partitioned into relative groups. After grouping, all tuples in each group are sorted according to their primary: this is done for the purpose of synchronization. The watermark embedding process ends up by watermarking each group independently. One main thing to keep in mind which is very important is that the grouping and sorting operations are only virtual operations: there is no change of physical position of the tuples. In the watermark detection phase, the same operations as in the watermark embedding phase are repeated except that when the extracted bits from a group hash are retrieved; They are checked against the position of the tuples to verify the watermark. As opposed to all the techniques that we mention thus far that fall into robust watermarking, the one described over here falls into fragile watermarking. It cannot only detect if the database relation was tampered, but it can also localize the regions where the unauthorized modifications, insertions or deletions were made.

In [10], following quite the same idea as [9] with some adjustments, the proposed scheme is a fragile watermarking algorithm which verifies the integrity of streaming data at the application layer. The algorithm divides data into groups on the fly according to a secret key. A watermark is is embedded directly to the data in each group. As opposed to [9], here there is a slight distortion on the data, but with no harm since data manipulated are stream. The embedded watermark can also detect and locate modifications made to the data stream as in [9]. For the completeness of the data stream, watermarks are chained across sequential groups. This allows data deletion to be correctly detected.

In [14], several techniques for hiding information in an XML document has been proposed.

The first techniques is using the empty element representation in XML. The W3C recommends the representation of an empty element as follow "<element-Tag-Name> < /element-Tag-Name> or <element-Tag-Name/ >." With those two representations of an empty element, bit of data could be hidden in an XML document. For instance, "<element-Tag-Name> < /element-Tag-Name>" could be used to hide the bit "0" while " <element-Tag-Name/ > " is used to hide the bit "1."

The second technique uses the XML element tag name to hide information. It is proposing to insert or delete white space in the element tag name. For instance, the representation of an element tag name as follow:

"<element-Tag-Name>, < /element-Tag-Name> or <element-Tag-Name/ > " could be used to hide the bit "0" while " <element-Tag-Name >, < /element-Tag-Name > or <element-Tag-Name / > " is used to hide the bit "1."

Several other techniques for hiding information in an XML document are further discussed in [14].

The scheme used in [9] is distortion free. It embeds the watermark in the database by modifying the order of tuples. Unfortunately, the technique used here cannot be applied to XML data. Because of the structure of XML data, its content cannot be ordered as needed without affecting the data as with tuples in a database.

The scheme used in [10] is processing streaming data on the fly. The problem with this technique is that it introduces slight distortions to the actual data, which cannot be tolerated by XML data.

The techniques discussed in [14] are ways to hide information in an XML document that can be retrieved later on as needed. Those techniques alone cannot allow the watermark embedding and verification process on an XML file.

The scheme proposed in this work fulfills the lack of work and research on XML data toward fragile watermarking.

3 Algorithms

In this section, we present the algorithm relative to our fragile watermark scheme for XML data. The watermark, which can be a white space or any combination of string characters, is embedded in the tag name of an element in the XML file as the technique detailed in [14]. The verification process detects and locates any modification that is made to the text within an element tag name or the alteration of the value of its attributes and also the insertion or deletion of one or several elements.

3.1 Assumptions and Design Purposes

In this study, we only focus on the modification of the data that are present within the tag name of an element or the value of its attributes and the insertion or deletion of elements.; Any changes that can be made on the tag name itself or some structures of the XML file is not taken into consideration. Since some proven methods and techniques that are part of the XML standard such as Document Type Definition (DTD) or schema can validate an XML document, it seems legitimate in our work to focus only on the data that are contained in the XML document. The design purposes of watermarking schemes are somewhat different according to specific applications. The scheme used in this work is a fragile watermark for tamper detection on XML data. An unauthorized user will modify the XML document with the precaution of making the modification undetectable by the watermark verification process. This scheme can detect and locate any modification that is made to an XML document, since it is designed to be fragile. Also, this scheme is distortion free, which means that it doesn't introduce any distortion to the actual data that are contained in the XML file. The followings are the properties of our fragile watermark for XML data:

1. *Invisibility:* In our fragile watermarking scheme for XML data, the embedded watermark doesn't introduce distortion to the data contained in the document.
2. *Prevent illegal embedding and verification:* The watermark embedding and verification process used in the present scheme is regulated by a watermark key and the number of groups. Only authorized people with the key and the group number would be able to embed and verify the watermark process.
3. *Blind verification:* The initial XML document that is used during the watermark embedding process is not required for the watermark verification.
4. *Localization:* If the XML document is tampered, the embedded watermark should able to detect and narrow down the modification to a set of elements.

3.2 General Description

The algorithm used in our scheme takes a XML file and embeds watermark into it. It then later takes the XML file with embedded watermark to verify if the file was tampered or not. To embed the watermark, the algorithm first decomposes the XML file to elements. Each element e_i is identified with its $path_i$ (the concatenation of its tag name with all the tag names of its parents up to the root of the XML document), its $text_i$ (the concatenation of the text within the element tag name - leave of the element e_i), and its $attr_i$ (the concatenation of all the values of its attributes. After all the elements are identified and retrieved, the algorithm computes the hash value of each element according the watermark key, $path_i$, $text_i$, and $attr_i$. It then assigns each element to its relative group according to its hash value of the element. After the grouping, the elements are sorted in each group for synchronization purpose; the hash value of each group is processed according to the watermark key and the hash values of all the elements in that particular group. The extracted bits are retrieved from the group hash. The watermark is embedded in the element tag name according to the relative position and value of the extracted bits.

The verification process goes into the same steps, except the fact that when extracted bits are retrieved, instead of embedding the watermark, the relative position and value of the extracted bits are checked against the embedded watermark at the end of the element tag name (if any was embedded). The watermark verification will succeed if all the embedded watermarks are verified. The localization of the tampered data will be narrowed down to the groups that have a least one element that the embedded watermark failed to be verified.

Table 1: Notations and Parameters

e_i	the i^{th} element in the XMLFile or group
ω	number of elements in the XMLFile
g	number of groups for the XMLFile
v	average number of elements in a group
h_i	element hash value (key, path, text and attributes)
$\mathcal{H}$	group hash
$\mathcal{G}_j$	the j^{th} group
s_j	size of group j
K	watermark embedding key
$\mathcal{V}$	watermark verification result for W
att_i	attributes list of element e_i
$text_i$	all text within the element e_i
pth_i	path from element ei to the root of the XML document
W	watermark embedded in a group
W'	watermark extracted in a group

3.3 Decompose XML File to Elements

Algorithm 1 - Decompose XMLFile to elements - will browse the XML file and retrieve the element - e_i, the path of the element - pth_i, which is the concatenation of the element tag name and all the element tag names of its parents, the attributes - att_i, which is the concatenation of all the attribute values of the relative element, the concatenation of text within the element tag - $text_i$, and the total number of element in the XML file - ω

Algorithm 1 Decompose XMLFile to elements

Input: XMLFile
Output: e_i, pth_i, att_i, $text_i$, ω
while not (End Of File) **do**
  retrieve e_i, pth_i, att_i, $text_i$
end while
return e_i, pth_i, att_i, $text_i$, ω

Algorithm 2 Watermark embedding

1: For all $k \in [1, g]$
2: **for** $i = 1$ to ω **do**
3: $h_i = \mathcal{HASH}(K, e_i.pth_i, e_i.att_i, e_i.text_i)$
4: **end for**
5: $k = h_i \bmod g$
6: $\mathcal{G}_k \leftarrow e_i$
7: **for** $j = 1$ to g **do**
8: Embed Watermark $(K, h_1, h_2, \cdots, h_{s_\mathcal{G}})$ in $\mathcal{G}_j$ //See algorithm 3
9: **end for**

3.4 Watermark Embedding

Algorithm 2 - Watermark embedding - completes the process of the watermark embedding. It uses Algorithm 3. Algorithm 2 computes the $\mathcal{HASH}$ of every extracted elements. The $\mathcal{HASH}$ takes as parameters in following order: the watermark key K, pth_i, att_i, and $text_i$. It then assigns the element e_i to its corresponding group $\mathcal{G}_k$ according the its $\mathcal{HASH}$. Algorithm 2 then calls Algorithm 3 to actually embed the watermark in each group $\mathcal{G}_k$.

3.5 Embed Watermark

Algorithm 3 - Embed Watermark - embeds the watermark in each group. In our scheme the watermark embedding process and the watermark verification processes are synchronized. There is an urgent need to keep synchronization for the two processes, otherwise the watermark verification will fail even if the XML file wasn't tampered. For this reason, Algorithm 3 sorts elements e_i in the group according to their relative hash value. Algorithm 3 computes the group $\mathcal{HASH}$ that takes as parameters the watermark key K and the ordered element $\mathcal{HASH}$ h'_i. The watermark bits of the group are then extracted. According to the value of the watermark bit (in this case $W[i] == 1$) the watermark is embedded in the tag name (in this case a white space).

3.6 Watermark Detection

The watermark key K and the number of group g used in our scheme are known in advance on both side - the watermark embedding and watermark verification processes. Algorithm 4 starts by extracting the XML element and other necessary information from the XML file (e_i, pth_i, att_i, $text_i$, and ω). It then computes the $\mathcal{HASH}$ of every extracted elements. The $\mathcal{HASH}$ takes as parameters in following order: the watermark key K, pth_i, att_i, and $text_i$. It then assigns the element e_i to its corresponding group $\mathcal{G}_k$ according the its $\mathcal{HASH}$. After that, Algorithm 4 then calls Algorithm 5 for the watermark verification.

Algorithm 3 Embed Watermark

1: sort elements in the group according to their relative hash value
2: $\mathcal{H} = \mathcal{HASH}(K, h'_1, h'_2, ...h'_{\mathcal{G}_j})$ //h'_i is the element hash of the i^{th} after ordering
3: W = extracBit($\mathcal{H}$, $s_\mathcal{G}$)
4: **for** $i = 1$ to $s_\mathcal{G}$ **do**
5: **if** $W[i] == 1$ **then**
6: e_i-elementTagName = e_i-elementTagName + WhiteSpace
7: **end if**
8: **end for**
9: extractBits($\mathcal{H}$, l){
10: **if** length($\mathcal{H} \geq l$) **then**
11: W = concatenation of first l selected bits from $\mathcal{H}$
12: **else**
13: $m = l$ - length($\mathcal{H}$)
14: W = contatenation of $\mathcal{H}$ and extractBits($\mathcal{H}$, m)
15: **end if**
16: **return** W }

Algorithm 4 Watermark detection

1: decompose XMLFile to elements //See algorithm 1
2: For all $k \in [1, g]$
3: **for** $i = 1$ to ω **do**
4: $h_i = \mathcal{HASH}(K, e_i.pth_i, e_i.att_i, e_i.text_i)$
5: **end for**
6: $k = h_i \bmod g$
7: $\mathcal{G}_k \leftarrow e_i$
8: **for** $j = 1$ to g **do**
9: Watermark verification $(K, h_1, h_2, \cdots, h_{s_\mathcal{G}})$ for $\mathcal{G}_j$ //See algorithm 5
10: **end for**

3.7 Watermark Verification

As we mention earlier, the watermark embedding process and the watermark verification processes have to be synchronized. Algorithm 5 starts by sorting elements e_i in the group according to their relative hash value. It then computes the group $\mathcal{HASH}$ that takes as parameters the watermark key K and the the ordered element $\mathcal{HASH}$ (h'_i). The watermark bits of the group are then extracted. According to the value of the watermark bits, the test for the verification is made. In this case $W[i] == 1$, Algorithm 5 will check to see if there is a white space(or other chosen embedded string character) at the end of the tag name of the element. If $W[i] == 0$, the process checks to see if there is no white space(or other chosen embed-

Algorithm 5 Watermark verification

1: sort elements in the group according to their relative hash value
2: $\mathcal{H} = \mathcal{HASH}(K, h'_1, h'_2, ...h'_{\mathcal{G}_j})$ //h'_i is the element hash of the i^{th} after ordering
3: $W' = \text{extracBit}(\mathcal{H}, s_{\mathcal{G}})$
4: $\mathcal{V} = \text{true}$
5: **for** $i = 1$ to $s_{\mathcal{G}}$ **do**
6: **if** $(W'[i] == 1$ **and not** (spaceEndTag$(e_i)))$ **or** $(W'[i] == 0$ **and** (spaceEndTag$(e_i)))$ **then**
7: $\mathcal{V} = \text{true}$
8: **end if**
9: **end for**
10: spaceEndTag(e) {
11: **if** there is a space at the end of the tag name of the element **then**
12: **return** true
13: **else**
14: **return** false
15: **end if**}

ded string character) at the end of the tag name of the element. If either of the above test fails, Algorithm 5 detects a modification and retrieves the location of the corrupted elements (all elements in the group that fail the verification test).

4 Experiments

We used the programming language Java with the following characteristics: java version 1.5.0_06 Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05).

We also used the Java version of Apache Xerces as XML processor with the following characteristic: Xerces-J 2.8.0 .

During the programming exercise the XML file is represented as DOM tree before manipulation.

The hardware used has the following characteristics: Dell Dimension 8300, Intel(R). Pentium(R) 4 CPU 2.60GHz, 512 MB of RAM running Microsoft Windows XP Professional SP2.

For our experiment, we use an XML file with real data. The characteristics can be seen in the tables 3 and 4.

The results of the tests done on our XML file can be seen in the table 2. For each case(each line of the table), 10 tests were done(for a total number of 200 tests combined) and the changes were applied randomly.

Note: success% = (total number of errors detected) / (total number of expected errors to be detected). Since some groups were actually bigger than others during this

Table 2: Experiment Results

Insertion of a Single Element			
n	g	v	$succes\%$
1	4908	2	30%
1	981	10	100%
1	193	51	100%
1	99	96	100%
Deletion of a Single Element			
n	g	v	$succes\%$
1	4908	2	20%
1	981	10	90%
1	193	51	100%
1	99	96	100%
Modification of a Single Element(att_i and $path_i$)			
n	g	v	$succes\%$
1	4908	2	50%
1	981	10	100%
1	193	51	100%
1	99	96	100%
Modifications of Multiple Elements			
n	g	v	$succes\%$ $(m = 50)$
50	4908	2	20%
n	g	v	$succes\%$ $(m = 10)$
50	981	10	80%
50	193	51	100%
50	99	96	100%

Table 3: Characteristic of the XML Test File

Number of elements	9814
Number of attributes	143
Number of spaces	0

Table 4: Watermark Embedding and Verification Process time

Parsing time XML to DOM object	390 ms
Watermark embedding	11718 ms
watermark Verification	13380 ms

experiment, v was processed as follow: (number of elements in the biggest group + number of elements in the smallest group)/2. n represents the number of modified, inserted or deleted elements.

Also, for the "Modification of Multiple Elements" case, we consider the number of modified groups $m = 10$, and $k_i = k_j$ (the number of modified elements in each modified group is equal). However for the first line $v = 2$, 50 groups were modified (one element in each modified group). For the rest, 50 elements were modified with 5 elements in each of the 10 modified groups.

5 Conclusions

Through this work, we proposed an algorithm for tamper detection and localization for XML data Using fragile watermarks. As the scheme described in [9], there is no distortion of data during the watermark embedding and verification process. As seen in the experimental results, the probability of missing a detection is very low. Also we will mention again the watermark embedding and verification process are done in each group independently and the scheme can detect and localize any modification with a high probability.

References

[1] The W3C - World Wide Web Consortium, was founded in October 1994 to lead the World Wide Web to its full potential. www.w3.org

[2] C. Collberg and C. Thomborson Software watermarking: models and dynamic embeddings In *Annual Symposium on Principles of Programming Languages archive Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1999.

[3] H. Wu and Y. Cheung A New Fragile Mesh Watermarking Algorithm for Authentication In *Proceedings of The 20th IFIP International Information Security Conference, pp. 509–523*, 2005.

[4] C. Rey and J. Dugelay A Survey of Watermarking Algorithms for Image Authentication In *JASP, No. 6, pp. 613-621*, 2002.

[5] R. Agrawal and J. Kiernan. Watermark relational databases. In *Proc. of the 28th Inter. Conf. On Very Large Data Bases*, 2002.

[6] R. Sion, M. Atallah and S. Prahakar Watermarking non-media content: XML and Databases In *CERIAS Security Symposium*, 2001.

[7] X. Zhou, H. Pang, K. Tan and D. Mangla WmXML: A System for Watermarking XML Data In *Proceedings of the 31st VLDB Conference*, 2005.

[8] W. Ng and L. Lam Effective Approaches for Watermarking XML Data In *10th International Conference on Database Systems for Advanced Applications DASFAA*, 2005.

[9] H. Guo, Y. Li, A. Liu and S. Jajodia A Fragile Watermarking Scheme for Detecting Malicious Modifications of Database Relations In *Information Sciences , Vol. 176, No. 10, pp 1350-1378*, 2006.

[10] H. Guo, Y. Li and S. Jajodia Chaining Watermarks for Detecting Malicious Modifications to Streaming Data In *Information Sciences*, to appear.

[11] O. Benedens and C. Busch Towards Blind Detection of Robust Watermarks in Polygonal Models In *Computer Graphics Forum, Vol. 19 Issue 3, p199, 10p*, 2000.

[12] O. Harmanci and M. Mihcak Motion Picture Watermarking Via Quantization of Pseudo-Random Linear Statistics In *Proceedings of Visual Communications and Image Processing Conference (VCIP)*, 2005.

[13] X. Zhou, H. Pang, K. Tan and D. Mangla WmXML: A System for Watermarking XML Data In *Proceedings of the 31st VLDB Conference*, 2005.

[14] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto and H. Nakagawa A Proposal on Steganography Methods using XML In *Symposium on Cryptography and Information Security, pp.301-306* , 2002.

[15] R. Sion, M. Atallah and S. Prabhakar Resilient Information Hiding for Abstract Semi-Structuresa In *Proceedings of the Workshop on Digital Watermarking IWDW, Seoul, Korea*, 2003.