Structural Signatures for Tree Data Structures

Ashish Kundu
Department of Computer Science, Purdue University, West Lafayette, IN, USA
ashishk@cs.purdue.edu

Elisa Bertino
Department of Computer Science, Purdue University, West Lafayette, IN, USA
bertino@cs.purdue.edu

ABSTRACT

Data sharing with multiple parties over a third-party distribution framework requires that both data integrity and confidentiality be assured. One of the most widely used data organization structures is the tree structure. When such structures encode sensitive information (such as in XML documents), it is crucial that integrity and confidentiality be assured not only for the content, but also for the structure. Digital signature schemes are commonly used to authenticate the integrity of the data. The most widely used such technique for tree structures is the Merkle hash technique, which however is known to be "not hiding", thus leading to unauthorized leakage of information. Most techniques in the literature are based on the Merkle hash technique and thus suffer from the problem of unauthorized information leakages. Assurance of integrity and confidentiality (no leakages) of tree-structured data is an important problem in the context of secure data publishing and content distribution systems.

In this paper, we propose a signature scheme for tree structures, which assures both confidentiality and integrity and is also efficient, especially in third-party distribution environments. Our integrity assurance technique, which we refer to as the "Structural signature scheme", is based on the structure of the tree as defined by tree traversals (pre-order, post-order, in-order) and is defined using a randomized notion of such traversal numbers. In addition to formally defining the technique, we prove that it protects against violations of content and structural integrity and information leakages. We also show through complexity and performance analysis that the structural signature scheme is efficient; with respect to the Merkle hash technique, it incurs comparable cost for signing the trees and incurs lower cost for user-side integrity verification.

1. INTRODUCTION

Data sharing among multiple parties with high integrity assurance is an important problem which has been widely investigated. An integrity assurance technique provides mechanisms using which a user can verify that the data has not been tampered with. Specific integrity assurance requirements and techniques depend on the structure according to which the data are organized. Because one of the most widely used data organization structures is the tree structure (see the XML-based example in Section 2), the development of techniques specifically suited for data organized according to such tree structures is crucial. When addressing the problem of integrity for tree structures it is important to notice that each node typically contains some content and that the structural relationships between the nodes may establish some relationships between the contents in these nodes. Such relationships may be defined according to properties such as classification, indexing, temporal-orientation and sensitivity of the contents [10]. Integrity of such relationships is referred to as *structural integrity*, whereas the integrity of the contents is referred to as *content integrity*. An integrity mechanism for tree structures must thus preserve both content and structural integrity. In many cases such as healthcare and military scenarios, an additional requirement is to maintain the confidentiality of the content and the structural information [26]. By confidentiality we mean that: (i) a user receives only those nodes and the structural information that the user is allowed to access, according to the stated access control policies; (ii) a user should not receive nor should be able to infer any information about the content and presence of nodes and structural information that the user is not authorized to access.

The Merkle hash technique [21] is the most well known integrity assurance technique for tree structures and has been widely extended for use in high-assurance integrity content distribution systems and in secure data publishing [4, 8, 12, 14, 18, 20]. A drawback of such technique is that the integrity verification process does not preserve confidentiality. Merkle hash is binding (integrity) but not hiding (confidentiality) [5]; therefore it is vulnerable to inference attacks[1]. The signature of a non-leaf node combines the signatures of its children nodes in a particular order. Further, in order to allow the user to compute the hash of a node during integrity verification, the signatures of a set of nodes/subtrees in the tree has to be revealed to the user, even if the user does not have access to these nodes/subtrees. By the mere

[1]The inference problem is a widely investigated problem in computer and information security [22]. An important issue is to avoid that mechanisms designed to address one security requirement, i.e. integrity, undermine the other relevant security requirement, i.e. confidentiality

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-305-1/08/08

fact that such signatures are received, the user may infer that a given node, to which the user has access, has a sibling/parent/child node, even though the user does not have access to it. Such an inference may lead to confidentiality breaches, as we will show through an example in Section 2.

More specifically, the integrity verification of a subtree S, which belongs to a tree T by using the Merkle hash technique reveals: (1) the signature (Merkle hash) of some nodes which are in T but not in S; (2) the structural relationship between a node x in S and some node y which is in T but not in S; and (3) the relative (structural) order between a node x which is in S and y, which is in T but not in S. One approach to avoid such information leakage is to pre-compute and store a separate Merkle signature for every distinct subtree that may be the result of a query or a request to access the tree. However such an aproach is impractical because the result of a query can be an arbitrary subtree and there can be an exponential number of such subtrees in a tree.

The problem that the paper addresses is as follows: The trusted owner Alice of a data item organized as a (rooted) tree T wants to digitally sign T *once* so that it can be queried or accessed many times. A user Bob should be able to verify the integrity of the content and structure of a subtree S (of T) that Bob is authorized to access. Any information about a node which is in T but not in S, its signature, its structural relationship with any other node in T should *not* be revealed to Bob. Obviously the Merkle hash technique cannot be effectively used for this purpose. In this paper, we propose an integrity assurance technique for tree structures which is secure against the above information leakages and is also efficient.

The distribution of data is often carried out through third parties, which are not completely trusted in the sense that the trusted owner relies on the third party D for the distribution but does not wish to provide D the authority to sign on its behalf. This may not be due to a lack of trust in D but merely a recognition of the fact that typical systems such as D are more vulnerable (to breakdowns, spy wares, insider misbehavior, or simply accidents) than the trusted owner. This model offers many advantages, but also offers a challenge: How does the third-party distributor D, which does not have the authority to sign (only Alice does), prove to a user the integrity of the data provided to the user without leading to leakage of information related to structure as well as content?

The obvious answer is to sign the data (tree) once and store at D a number of integrity verification items that D can later on provide to any user who is legitimately requesting a subset of the data (a subtree) which the user is authorized to access. These integrity verification items are cryptographic hashes or signatures that enable the user to verify the integrity of the subtree with respect to both content and structure that it receives.

Our integrity assurance technique, which we call the "structural signature scheme", overcomes the drawbacks of Merkle hash technique and related techniques. In this scheme, the signature of a tree, called "structural signature", is based on the structure of the tree as defined by tree traversals - post-order, pre-order and in-order. It is well-known that a non-binary tree can uniquely be re-constructed from its post-order and pre-order sequences of vertices [7] and a binary tree from its in-order and pre-order/post-order sequences of vertices [16]. Our tree signature is based on such property and is defined using a randomized notion of such traversal numbers, which are extensions of the respective traversal numbers. The structural signature scheme further helps in reducing the computational cost at the trusted owner, which signs the tree, and at the user side. It also helps in reducing the communication cost between the distributor and the user. In the paper, in addition to formally defining the technique, we prove that it protects against violations of content and structural integrity and information leakages. We also carry out a performance analysis and show that our technique is competitive in terms of performance with respect to the Merkle hash technique.

Novelties of the Structural Signature Scheme. The structural signature scheme possesses several novel features over existing techniques:

- It provides stronger security guarantees in terms of integrity and confidentiality.
- It simplifies the transmission of tree-based data from a distributor to a user and improves the efficiency of such transmission. It facilitates sending *only* the nodes of a subtree to a user; no structural information about the parent-child and the ordering between the siblings needs to be sent. Note that the Merkle hash technique and the related techniques require sending the subtree as it is - the nodes and the structural information, which incurs more cost.
- Like the Merkle hash technique, it facilitates precise detection of integrity violations in the sense that it precisely identifies the node or the structural relationship that has been compromised.
- While its signature generation time is comparable to that of the Merkle hash technique, the user-side integrity verification time that it requires is less than the time taken by the Merkle hash technique.

Organization of the Paper. The paper is organized as follows. Section 2 presents a running example. The threat model considered in the paper is described in Section 3. The notations used in the paper and a background on tree traversal numbers are given in Section 4. Section 5 introduces the notion of randomized traversal numbers and two key lemmas. These lemmas are used to define the notion of structural signature in the next section - Section 6; this section also defines the steps for signing a tree and verifying the content and structural integrity of subtrees received by a user. Section 7 illustrates our signature scheme using the running example. Security analysis and performance of the structural signatures are discussed in Section 8 and 9, respectively. Section 9.4 discusses its non-repudiation property, management of dynamic updates, and its advantages. Related work is presented in Section 10. Finally, Section 11 concludes the paper.

2. RUNNING EXAMPLE

Our running example is in the area of XML data management. XML organizes data according to the tree structure; integrity and confidentiality of XML data is an important

```
<HealthRecord>
    ...
    <PatientID id=2345S>
    ...
    </PatientID>
    <Contact>
        ...
    </Contact>
    <CriticalDiseases>
        <Disease name=Cancer>
            <Surgery>
            ...
            </Surgery>
            <Chemotherapy>
                <Treatment instance=1>
                    <Doctor name=Dr. S. Stevens/>
                    <DateTime date=20Feb2003 time=2PM/>
                </Treatment>
                <Treatment instance=2>
                    <Doctor name=Dr. M. Paul/>
                    <DateTime date=17Jan2003 time=12PM/>
                </Treatment>
            </Chemotherapy>
            <Medication>
            ...
            </Medication>
        </Disease>
        <Disease name=KidneyFailure>
            ...
        </Disease>
    </CriticalDiseases>
    ...
</HealthRecord>
```

(a)

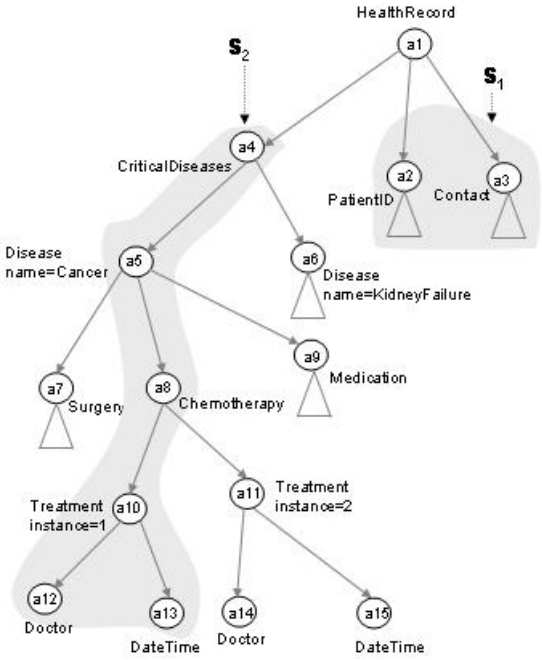

Figure 1: (a) XML-based Health-care record of a patient. (b) The tree representation of the HealthRecord.

requirement, given the widespread adoption of XML for distributed web-based applications. As such, XML is an important application domain for the techniques presented in the paper.

The XML document in Figure 1 is the health-care record of a patient and thus contains sensitive information. Assume that such record is stored in a hospital database and that its schema, referred to as *HealthRecord*, is defined as follows. The *HealthRecord* element, that is, the root of the tree has a child for each of the following elements: *CriticalDiseases*, *PatientID* and *Contact*. The *CriticalDiseases* element is used to list all the critical diseases a patient suffers from; information about a specific critical disease is specified as its child by the element *Disease*. Inside each *Disease* element, the types of treatment that the patient has gone through for that same disease are listed. For *Cancer*, the types of treatment are specified by the following elements: *Surgery*, *Chemotherapy*, and *Medication*. Each type of administered treatment is specified as a child node of the node specific to the treatment type and is an instance of *Treatment* element. It contains an attribute *instance*, which refers to the specific instance of the treatment, and child elements to specify the date and time of administering (*DateTime*), and the name of the doctor who administered the treatment (*Doctor*). A patient may have received treatments from different doctors, each related to a different instance of the same type of treatment or instance of a different type of treatment. For expository purposes, we have associated a label with each node in the health record in Figure 1(b); for example, the node *Chemotherapy* is labeled by a_8.

The hospital database, which can be accessed remotely, stores all such patient health records. The Merkle hash technique is used to sign the tree and support integrity verification by the data consumer. Table 1 (in the Appendix) presents the details of how to compute and verify the Merkle hash for each node in the tree in Figure 1(b). The third column of such table also lists the information which is leaked when the integrity of a node is verified.

Consider the following scenario. A cashier has access to the subtree S_1, shown in Figure 1, including the root a_1, that are essential for financial and administrative purposes. She does not have access to a_4 and its content that refers to *CriticalDiseases*. An access to the health-record in Figure 1(a) leads to the integrity verification of this portion of the health record at the side of the cashier. During this process, the cashier receives the Merkle hash of a node x and she also receives the following information: x is a child of a_1 and is on the left side of a_2 and a_3. By knowledge of the schema, the cashier determines that a node at such position must be the *CriticalDiseases* node. Thus the cashier infers that the patient is definitely suffering from some critical disease. If the hospital specializes in some specific critical disease(s), the cashier can further infer which (possible) disease the patient is suffering from. Each of these inferences leads to

disclosure of information that is sensitive for the patient.

We now consider another scenario, which leads to leakage of more detailed sensitive information. A nurse has access to S_2 and a_1 from the record in Figure 1. He has access to S_2, because he works with the doctor *S. Stevenson*, who prescribed the administering of this treatment. The nurse receives S_2 and a_1 and the corresponding signature information from the remote database. In order to be able to verify the integrity of S_2, he also receives the signature of a_6, a_7, a_9 and a_{11}.

The schema of *HealthRecord* specifies that a child of *CriticalDiseases* refers to a critical disease from which a patient suffers from. By receiving the signature for a_6, which is a child of the node a_4 (element *CriticalDiseases*), the nurse infers that the patient is suffering from another critical disease different from cancer. This is a disclosure of private information, to which the nurse does not have access to. Assume that the hospital of our example specializes on the treatment of only a limited number of critical diseases. It is thus easy to infer what the other disease is. Furthermore, by inferring that a_5 has two siblings, that is, a_7 and a_9, the nurse is able to infer that the patient has gone through two other treatments other than chemotherapy. Such inference may easily lead to determine the seriousness of the illness. In addition, from the schema, the nurse can infer that these nodes refer to *Surgery* and *Medication*, which reveals that the patient has been received either or both of these treatments. If the hospital has two doctors who specialize in *Surgery* or *Medication*, then the knowledge that the patient has been treated with *Surgery* or *Medication* leads to more information, such as that he has been treated by more than one doctors and who (possibly) has been his doctor.

Furthermore, by the disclosure of the signature of node a_{11} and its structural relationship with a_8 as its child, and by knowing that children of a *Chemotherapy* element refer to the treatment instances, the nurse is sure that the patient went through another Chemotherapy treatment and possibly with another doctor. Additional knowledge about doctors and the hospital could lead to more leakage.

3. THREAT MODEL

There are two threats that we want to protect against: data tampering attack that compromises data integrity and inference of information to which a user is not authorized.

Data tampering attack. An attacker wishes to tamper the content and/or the structural order between two or more nodes of a subtree S of a tree T that a receiver has access to.

Inference attack. The user Bob, who has access to the subtree S of a tree T is aware of the context, the schema of the tree-based representation of S and T, and can exploit the information leaked along-with S to infer sensitive information from each or a combination of the information units leaked:

- Leakage of signature of a node/subtree (y) in T but not in S: By comparing the signature of such a node with that of a node/subtree (x) in S, the attacker determines if x and y are identical subtrees (the hash operation used to compute Merkle hash is assumed to be collision resistant). He also learns about the existence of another node/subtree in T but not in S.
- Leakage of structural relationship between x in S and y: By knowing the specific relationship between two nodes, which may be either parent-child, ancestor- descendant or sibling-sibling, the attacker infers the nature and type of the sensitive information embedded in y in the context of x. As we saw in the health-care example earlier, such leakage leads to inference of exact information (Table 2 in the Appendix).
- Leakage of structural order between x and y: By knowing the specific order of x and y, which is either y is to the left of x or right of x, the attacker infers sensitive information such as a temporal relation between x and y.

4. BACKGROUND

In this section, we present the notations used in the paper and a review of tree traversals, which are central to the proposed signature mechanism.

4.1 Notations

A (rooted) tree is referred to as T or $T(V, E)$, where V and E denote the set of vertices and edges, respectively. S refers to a subtree. $S_x(V_x, E_x)$ refers to a subtree with root x and the set of vertices and edges V_x and E_x, respectively. A binary tree is a tree, in which each node has at most two children and each child is either a left or a right child [17]. The term "tree" in this paper refers to a "non-binary" tree, unless otherwise explicitly referred to as "binary" tree. C_x refers to the content of a node x.

h refers to a collision-resistant one-way hash function and $|r|$ denotes the number of bits used to represent a floating point number r.

We introduce a few more notations that are dependent on the context, as we proceed in the paper.

4.2 Review of Tree Traversals

Post-order, pre-order and in-order tree traversals are defined in [17]. While post-order and pre-order traversals are defined for all types of trees, in-order traversal is defined only for binary trees. In each of these traversals, the first node visited is assigned 1 as its *visit count*. For every subsequent vertex visited, the *visit count* is incremented by 1 and is assigned to the vertex. This sequence of numbers is called the sequence of post-order (PON), pre-order (RON) or in-order (ION) numbers for the tree T, depending on the particular type of traversal.

Properties of traversal numbers: The post-order number of a node is smaller than that of its parent. The pre-order number of a node is greater than that of its parent. The in-order number of a node in a binary tree is greater than that of its left child and smaller than that of its right child. A specific traversal number of a node l is always smaller than that of its right sibling r; i.e., p_l is smaller than p_r. The distribution and range of the traversal numbers are uniform and deterministic ($[1, 2, \ldots, |V|]$). The determinism of the distribution and range of the traversal numbers make them unsuitable for our purposes as they reveal information about the approximate size of the data and the position of the subset of data in the data set. The structural signature as defined later is based on these traversal numbers. It is possible for an adversary to exploit this information and replace a signed node with a compromised or a differ-

ent node altogether by assigning it to the original pre-order number. Siblings can be interchanged and the corresponding visit counts could also be interchanged while satisfying the specific properties.

In order to overcome the above limitations of the traversal numbers, we propose the notion of *randomized traversal numbers - randomized* post-order, pre-order and in-order numbers (referred to as RPON, RRON, and RION, respectively).

5. RANDOMIZED TRAVERSAL NUMBERS

The definition of randomized post-order numbers is a generalization of the one proposed in [19], which proposes the notion of encrypted post-order numbers (not pre-order or in-order) for distribution of content; the encrypted post-order numbers are used as the indexing and routing parameter for the content.

We transform a traversal number into a unique random number such that the order between the traversal numbers (of a specific traversal) is preserved. By preserving the order of their original counterparts, the randomized traversal numbers preserve their properties. For an unordered tree, we transform a traversal number into a unique random number such that the order between the traversal numbers (of a specific traversal) for ancestors and descendants is preserved, whereas the order between such numbers among siblings does not need to be preserved. The distribution and range of randomized traversal numbers is non-uniform and non-deterministic.

DEFINITION 5.1. *The set of randomized traversal numbers of a tree T is defined as the set of distinct real numbers chosen randomly through a transformation of the set of traversal numbers, that is, $\mathcal{T}^e = \zeta(\mathcal{T})$, where: $\mathcal{T}$ and $\mathcal{T}^e$ refer to the set of traversal numbers and their randomized counterparts, respectively; ζ is a random transformation function such that for ordered trees, the order among all traversal numbers is preserved, and for unordered trees, the order among such numbers assigned to ancestors and descendants are preserved, while the order among those assigned to siblings does not need to be preserved.*

The randomized transformations of post-order, pre-order and in-order numbers are called as randomized post-order (RPON), randomized pre-order (RRON) or randomized in-order (RION) numbers. RPON, RRON and RION for a node x are referred to as p_x, r_x and i_x, respectively.

The following lemmas provide the basis for defining the notion of structural signature for trees using randomized traversal numbers in the next section.

LEMMA 5.2. *The pair of randomized in-order number and either post-order or pre-order number for a node in a binary tree correctly and uniquely determines the position of the node in the structure of the tree, where the position of a node is defined by its parent and its status as the left or right child of that parent.*

PROOF. From the in-order and either post-order or pre-order traversal sequences of the vertices, it is possible to *uniquely* re-construct a binary tree [16]. Thus from these sequences or from their randomized counterparts, for a node, it is possible to correctly identify its parent and its status as left or right child of that parent in the tree. Thus the lemma is proved. □

LEMMA 5.3. *The pair of randomized post-order number and pre-order number for a node in a (non-binary) tree uniquely determines its position in the structure of the tree, where the position of a node is defined by its parent and its siblings to its immediate left and right in the tree.*

PROOF. It follows from [7]. □

6. STRUCTURAL SIGNATURES FOR TREES

In this section, we develop structural signatures for trees, which makes use of the Lemma 5.3 and are defined using the post-order and pre-order traversals. Structural signatures for binary trees are defined identically except that in-order traversals are used as one of the components in place of either the post-order and pre-order traversals (Lemma 5.2). For simplicity of presentation, we focus only on non-binary trees.

6.1 Notion of Structural Signatures

A structural position uniquely identifies a node in a tree structure. For non-binary trees, it is defined as a pair of the RPON and the RRON of a node and for binary trees it is defined as a pair of the RION and RPON (or RRON) of the node.

The structural signature for the tree $T(V, E)$ denoted by G_T, is a hash of the structural position and content of all the nodes in the tree, taken in a particular sequence of the vertices, such as a post-order sequence. The hash is further certified by a trusted entity (the owner or a certifying authority). The signature can be salted by the addition of a random value if the fact that the received subtree (sent to the user) is same as the original tree needs to be protected as a sensitive information. The (salted) tree signature is publicly available or passed to the user alongwith the subtree it has access to.

The structural signature of a node x in tree T is defined as a hash of the structural position and content of x and the (salted) signature of the tree G_T. The hash is further certified by a trusted entity (the owner or a certifying authority). The use of the structural position in the computation of the hash binds the content and the tree signature to the node x, because the structural position (pair of RPON and RRON) of a node is unique in a given tree. The formal definitions of these notions are as follows. $\|$ denotes to the string concatenation operation.

DEFINITION 6.1. *Let x be a node in tree $T(V, E)$. Its structural position, denoted by ρ_x, is defined as a pair of its RPON p_x and RRON r_x, that is, $\rho_x = (p_x, r_x)$.*

DEFINITION 6.2. *Let the nodes in tree $T(V, E)$ be referred to as 1, 2, ..., n, where $n = |V|$. Let h denote a one-way cryptographic hash function. The structural signature of the tree T, denoted by G_T, is defined as $G_T = h(\rho_1 \| C_1 \| \rho_2 \| C_2 \| \ldots \| \rho_i \| C_i \| \ldots \| \rho_n \| C_n)$.*

DEFINITION 6.3. *Let x be a node in tree $T(V, E)$. The structural signature of x, denoted by G_x, is defined as $G_x = h(G_T \| \rho_x \| C_x)$.*

Any modification of the content or the tree-structure, after the tree is signed, would be detected. If x has not been tampered with, the signature G_x with which the node is signed is same as the hash of the following elements: G_T (publicly available or provided with the subtree), ρ_x and C_x. If they are not equal, then x has been tampered with.

6.2 Signing a Tree

The steps that the trusted owner Alice follows in order to sign and share a tree $T(V, E)$ with Bob and other clients are given below.

1. Compute the post-order and pre-order numbers for each node in T.
2. For each node x in V: transform the post-order and pre-numbers into randomized post-order and pre-order numbers denoted respectively as p_x and r_x, such that
   (a) for unordered trees, RPON's and RRON's among the siblings do not need to preserve any order, while for ancestors and descendants, they need to preserve the order;
   (b) for ordered trees, RPON's and RRON's for all nodes, need to preserve the order.
3. Assign (p_x, r_x) to x as its structural position ρ_x.
4. Compute the structural signature of the tree T, G_T from a specific sequence of vertices (such as the post-order sequence which can be available from the steps 1 and 2).
5. For each node x in V: compute the signature G_x.

After the signatures are generated, Alice or a Certificate authority certifies[2] G_T and the signature G_x of each node x. Computation of randomized post-order and pre-order numbers (Steps 1 and 2 in the above algorithm) can be carried out in one traversal (instead of two) by processing a node x for its pre-order number, then recursively processing all its children; after all the children of x are processed, x is processed for its post-order number.

How to compute randomized traversal numbers: The interval between two adjacent randomized traversal numbers is a random. It is either a single random η or is chosen as follows: draw a random number m_i, then draw m_i number of randoms and compute η as follows: $\eta = \sum_{1 \leq j \leq m_i} \eta_j$. Order-preserving encryption (e.g. [3]) may instead be used for the computation. A set of sorted random numbers are input to such a technique. The numbers in the output are used as randomized traversal numbers.

After the tree is signed and the signatures are certified, the tree is ready to be shared with Bob and other users. Suppose that Bob wishes to access T and thus sends such a request to a distributor. Bob has the authorization to access the subtree $S_z(V_z, E_z)$. $S_z(V_z, E_z)$ can be shared with Bob according to two different strategies - (1) by sharing the signed subtree - its nodes and the structure or (2) by sharing the signed nodes in the subtree and letting Bob reconstruct the subtree using the RPON's and RRON's of the nodes. We describe both of these options in the following sections.

6.3 Sharing the Subtree along with its Structure

The distributor sends to Bob: S_z along with its structure (e.g. in the form of an adjacency matrix) and the structural signature of each node in this subtree (and the structural signature of the tree if it is not publicly available). Bob receives the subtree and it refers to it as $R(V_r, E_r)$ (with a different name in order avoid any ambiguity). Bob is aware of the h function used. He verifies the integrity of each node. Next, he verifies the integrity of structural relationships among all those nodes in $R(V_r, E_r)$, whose integrity and authenticity have been correctly verified.

Authentication of Signatures. Bob verifies the certificate of the signature of the tree T and the signature of each node in S_z. If the certificate is valid, then the signature of the node is valid. A spurious node would not be certified by a trusted entity; so such a node would be detected during this process.

Integrity Verification for Content and Signature

1. For each node y in the set of received nodes V_r, Bob computes its structural signature from its position and content and compares it with the signature G_y with which y has been signed:
   (a) $temp \leftarrow h(G_T \parallel \rho_y \parallel C_y)$
   (b) If $temp$ is same as G_y then the integrity of the node y is verified.

Integrity Verification for Structural Relations. Integrity verification of structural relations in a tree involves traversing the tree and comparing the RPON (RRON) of each node with the RPON (RRON) of its parent or its sibling. The steps are as follows.

1. Carry out a pre-order traversal on R.
2. Let x be the parent of z; if $((p_x \leq p_z)$ or $(r_x \geq r_z))$, then parent-child relationship between x and z is incorrect.
3. For ordered trees, let y be the right sibling of z; if $((p_z \geq p_y)$ or $(r_z \geq r_y))$, then the left-right order among the siblings y and z is incorrect.

6.4 Sharing a Subtree - only the Nodes

An advantage of the structural signatures is that there is no need to supply the user with the structure of the subtree it is receiving. Our scheme reduces the amount of data that needs to be transmitted from the distributor to the users and thus improves the efficiency of the data distribution. The structure can be reconstructed from the pre-order and post-order traversals (for non-binary trees [7]) (in-order and pre/post-traversals for binary trees [16]). The subtree is shared as a set of nodes signed by the structural signature. Such a set of nodes received by the user is processed as follows:

Reconstruction of a Subtree using Structural Signatures. The structural position of a node includes the RPONs and RRONs, which possess the same properties as that of the post-order and pre-order numbers. The subtree reconstruction algorithm [7] can thus use RPONs and RRONs. There is no need to carry out the verification of the structural integrity, as it is automatically taken care of during the subtree re-construction process. For binary trees, the

[2]Structural signatures facilitate verification of both structural and content integrity, while certification of a digest of the content of a node can be used to verify only the integrity of content, not the structural integrity.

algorithm given in [16] can be used, where RIONs would be used.

1. Authenticate the signatures of the tree and the nodes by verifying their certificates.
2. Verify the integrity of content and signatures of the nodes as per the procedure in Section 6.3.
3. Apply the algorithm by Das et al. [7] (Section 3.3)[3] for reconstruction of the sub-tree with the following changes:
   (a) use the RRONs and RPONs of the nodes as post-order and pre-order numbers
   (b) if an edge thus constructed involves a node (or nodes) whose integrity is found to be invalid in the previous step, then this edge is treated as invalid.

In the algorithm by Das et al. [7], consecutive nodes in post-order (pre-order) means the nodes with RPONs (RRONs) that are next to the other.

7. ILLUSTRATION

Consider the tree in our running example (Figure 1) and suppose that the tree has been assigned post and pre-order numbers (Figure 2(a)) and their randomized counterparts (Figure 2(b)). The structural position of each node is represented as (RPON, RRON) in Figure 2(b). Each node has a content that consists of the name of its element and attribute value pairs. Since the partial tree signatures and the hash values are large bit strings (e.g., 160 bits for SHA1), we do not show their values.

The cashier has access to the subtree S_1. The database D sends two nodes - a_2 and a_3, their structural signatures alongwith the (salted) tree-signature G_T. The cashier receives two nodes a_2 and a_3. She applies the integrity verification procedure on each of these nodes. She applies the hash function to the concatenation of the G_T, 66.2, 69.1 and C_2). Then she verifies if the resulting value is equal to the received node signature of a_2; if this is the case, then the integrity of the node is verified. If the content/structural position of the node a_2 has been compromised, then it can be detected during this process. The same procedure is followed for a_3. The cashier verifies the structural integrity. a_2 and a_3 were sent as siblings. The cashier verifies whether the RRON of a_2 p_2 (=66.2) is smaller than that of a_3 p_3(=69.5); if so, then a_3 is an ancestor or a right sibling of a_2. However since p_2 (=69.1) $<$ p_3 (=78.2), a_3 is not an ancestor of a_2. Thus their relationship is correctly verified.

The nurse is authorized to access a S_2; however suppose he receives a S_2 that is tampered, such that in the tampered tree, a_{10} is the child of the node a_5 and a left sibling of a_8. Such a violation of structural integrity can be detected by comparing the structural positions of the nodes as in Section 6.3. The RRON of node a_{10} is greater than that of a_8, which means that a_{10} cannot be a left sibling of a_8. If a_{10} is received as the right sibling of a_8, the structural integrity is violated. Such violation is detected, because the RPON of a_{10} is smaller than that of a_8, which means that a_{10} cannot be a right sibling of a_8.

[3]Due to space limits, we could not replicate the algorithm from [7].

8. SECURITY ANALYSIS

This section analyzes the soundness of the structural signatures in terms of its integrity and confidentiality guarantees with respect to information leakage defined earlier.

8.1 Integrity

LEMMA 8.1. *Subject to the one-way and collision-resistant properties of the hash function h, any integrity violation of a node in a tree can be detected by using structural signatures.*

PROOF. Any compromise of the content C_x or the structural position of a node x ρ_x in T would invalidate the structural signature G_x, which is a hash of a message that contains C_x and ρ_x, unless the hash function h encounters a collision, which contradicts our assumption. Any unauthorized re-ordering between two or more nodes (violation of structural integrity) can be detected using the RPON's and RRON's (Lemma 5.3). Suppose x belongs to another tree T', different from T, but claimed to belong to T. Such a forgery is possible when the signature of T, G_T, is identical to that of T', $G_{T'}$. G_T and T', $G_{T'}$ are identical only when the one-way hash h has encountered a collision, which is a contradiction to our assumption (and the Random Oracle Hypothesis [25]). Can such a tree T' be found such that a collision be deliberately generated? By the property of h, it is "hard" to do so (under the Random Oracle Hypothesis). □

8.2 Leakage

Suppose that Bob has access to a subtree $S_z(V_z, E_z)$ in T. Let x and y be two immediate siblings in S_z, left and right respectively. Can Bob determine the existence of any other node u which he does not have access to, between two siblings x and y, by knowing the RPON's and RRON's of x and y?

In an unordered tree, there is no such leakage of information, as RPONs and RRONs among siblings do not have any order. In an ordered tree, there is no leakage, because the probability of inference is negligible[4] - ($\frac{1}{2^k}$, for a large k).

Without loss of generality, let (p_y - p_x) be smaller than (r_y - r_x). Thus there is a better chance to infer some information from (p_y - p_x) (= *temp*). However, the interval between two RPON's is a random value. There are many possible real numbers between two RPON's, because the RPON's are real numbers chosen randomly and can be represented using r-bits (such as 64-bits or 128-bits). The size of the real numbers is - 64-bits in IEEE 754 double format. In XLC (IBM implementation of C for AIX servers [1]), the size of the double is 128 bits. For 64-bits, the possible number of real values between the RPON's of x and y is in the order of 2^{53} (k is 53). Let *temp* be *i.d*, where i is the exponent and d is the decimal portion of *temp*. The total number of possible real values between the RPON's of x and y are: $i*2^{53}$ + $d*2^{53}$. Similarly for 128-bit double precision, the number of RPON's between x and y and 2^{117} (k is 117), as XLC uses an exponent size of 11-bits. The inference is not

[4]A negligible function is a function that approaches zero faster than the reciprocal of any polynomial. Such functions are typically used to prove the security of cryptographic protocols by showing that the probability that an adversary can carry out an attack successfully is a negligible function of the security parameter k [11].

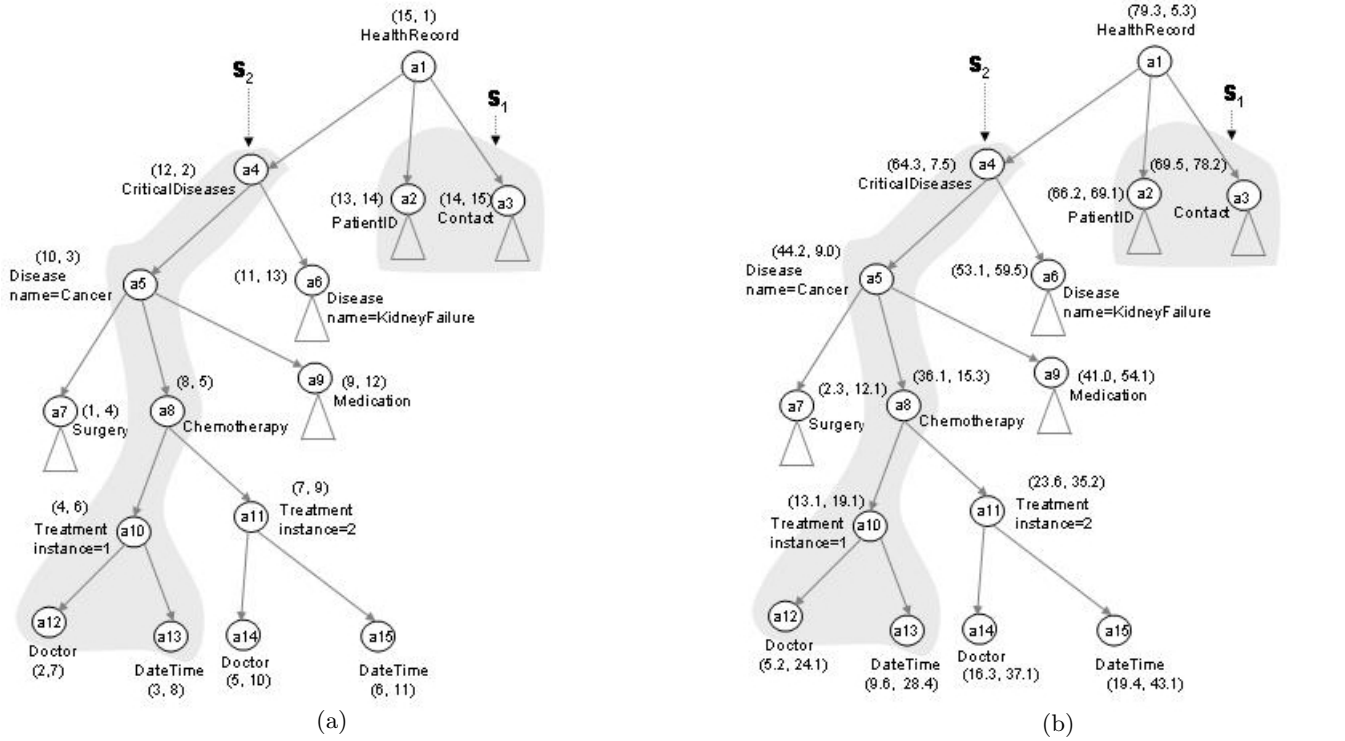

Figure 2: (a) Post-order and pre-order numbers assigned to the Health-care record as (PON, RON). (b) Randomized post-order and pre-order numbers assigned to the Health-care record as (RPON, RRON).

possible because of the large number of real values between the RPON's of x and y - in the order of 2^{53} or 2^{117}.

When the randomized traversal numbers are (treated as) cryptographically secure random numbers (which can be interpreted as real numbers, such as those supported by Java Cryptographic Architecture [2]) the size k of such numbers is 160-bits, or 512-bits. The probability of inference (and leakage) about the existence of node u between two immediate siblings from such randoms, is negligible ($\frac{1}{2^k}$). The same argument can be extended for RRON's and RION's.

Therefore Bob (as an attacker) can not infer any definite information about existence and signature of another sibling in between x and y, given that he has access to x and y. Further, he can not infer the structural relations or ordering that do not entirely belong to S_z, that involve nodes that do not belong to S_z.

LEMMA 8.2. *Subject to one-way and collision-resistant properties of the hash operation h, the structural signatures do not lead to any leakage of (1) node signatures, and information about the (2) existence of nodes, (3) structural relations or (4) structural order among nodes.*

PROOF. Suppose that a user Bob has access to $S_z(V_z, E_z)$, a subtree in T. Bob has access to the subtree, structural signature of T, the node signatures of S_z and their structural positions. Any leakage would be a direct leakage through these information or an inference from them.

Direct leakage: Clearly (as per Definitions 6.2 and 6.3, and the protocols specified in Section 6) Bob does not need the signature of any node (u) that is in T but not in S_z. He therefore does not need to know any of the structural relationships and structural ordering that exist in T, but not in S_z. Therefore none of (1), (2), (3) and (4) is directly leaked to Bob; he does not learn any extra information from the integrity verification process.

Indirect leakage through the signature of the tree and signatures of the nodes in S_z: Under the Random Oracle Hypothesis, the structural signature of the tree does reveals neither (1) - the existence of u nor (2) - the signature of u. Similarly, the structural signature of a node leaks neither (1) and (2). Therefore (3) - the structural relations (edges or paths) and (4) - t he structural order among nodes in S_z and u are not revealed by the signatures.

The structural positions (RPON's and RRON's) of the nodes in S_z: RPON's and RRON's are real numbers chosen randomly and can be represented using r-bits (such as 64-bits or 128-bits). The inference is not possible because of the large number of real values between the RPON's of x and y - in the order of 2^{53} or 2^{117} (k is 53 or k is 117). When the randomized traversal numbers are (treated as) cryptographically secure random numbers (which can be represented as real numbers) the size of such numbers k is 160-bits, or 512-bits, the probability of inference (and leakage) about (2) - the existence of node u between two immediate siblings from such randoms is negligible ($\frac{1}{2^k}$) [11].

The structural positions of nodes - the RPON's and RRON's cannot be used to determine the structural signature of u. Since (1) and (2) cannot be inferred from the RPON's and RRON's, (3) and (4) also cannot be inferred from the structural position of a node. □

Comparison with Merkle hash. In Merkle hash techniques and its derivatives, there is a definite release of $log(n)$ information, both in terms of content and structure. So the probability information leakage is 1. For structural signatures, there is no direct release of signatures of nodes and relationship among nodes that the user do not have access to. Moreover, as we quantified above, in our technique, inference attacks are also not practical to determine the existence of any sibling among the two other siblings.

9. PERFORMANCE AND DISCUSSIONS

In this section, we analyze the performance of the structural signature scheme with respect to the Merkle hash technique through complexity analysis and experiments.

9.1 Complexity Analysis

Cost of Signature Generation. The pre-order and post-order numbers can be generated by a single traversal of the tree $T(V, E)$. The traversal complexity is thus $O(|V|)$. The complexity for signing a tree according to the signing procedure in Section 6 is $O(|V|)$.

The storage complexity of structural signatures is: $|\rho_x|+|G_T|+|G_x|$, which turns out to be $(2*k+|h|+|h|)$, where k and $|h|$ are the number of bits used to represent a random number (RPON/RRON/RION) and the output of the hash respectively - a constant factor $O(1)$.

Cost of Distribution. If the distribution strategy is to share the signed subtree $S_R(V_R, E_R)$ including its structure, the distributor has to send $|V_R|$ nodes and information about $|V_R|$-1 edges. If the distribution strategy is to share only the signed nodes in the (signed) subtree and the user reconstructs the subtree using the RPON's and RRON's of the nodes, then the distributor has to send $|V_R|$ nodes only. The latter reduces the communication (sending) cost on the side of the distributor by about 50% than the former strategy.

Cost of Integrity Verification. If the distribution strategy is to share the signed subtree including its structure, the procedure for verification of content integrity incurs a cost linear in the size of the received subtree $S_R(V_R, E_R)$, that is, $O(|V_R|)$. It accounts for one hashing for each node. The cost of verification of structural integrity is also linear in terms of the size of the received subtree, that is, $O(|V_R|)$; the costs of comparison of RPON's and RRON's (and RIONs for binary trees) is constant. If the distribution strategy is to share only the signed nodes in the subtree (and no structure), the integrity verification cost comprises the cost of the integrity verification for the content and the signature, that is, $O(|V_R|)$, and the cost of reconstruction of the subtree using [7, 16], which is of linear order, that is, $O(|V_R|)$. The reconstruction process verifies the structural integrity as well. Such a distribution mode reduces the communication (receiving) cost on the side of the distributor by about 50% than the former strategy.

9.2 Comparison with the Merkle hash technique

Cost of Signature Generation. The complexity of generating Merkle signature for a tree $T(V, E)$ is $O(|V|)$, which is identical to that of the structural signatures. Our experiments show that structural signature scheme has almost the same performance as that of the Merkle hash technique; the difference is only marginal. The latter takes 0.13 seconds less for 65535 nodes than the former.

The storage complexity of the Merkle signature per node is $|G_T|$, which is $|h|$ bits, which is of $O(1)$ cost. The storage requirement of the structural signature scheme is higher in terms of a constant factor, which is $(2*k+|h|)$. However this constant factor difference in the storage requirement helps in improving other costs.

Cost of Distribution. Integrity verification of a subtree $S_R(V_R, E_R)$ in the Merkle hash technique involves computing the hashes of: (1) the nodes that are connected (adjacent) to a node in $S_R(V_R, E_R)$, but not part of $S_R(V_R, E_R)$, and (2) the ancestors of R, which is the root of $S_R(V_R, E_R)$. Such values are not necessary for integrity verification in case of structural signatures. Let μ refer to the set of the hashes as specified by (1) and (2) together. Therefore the distributor has to send all the nodes in the subtree V_R, the hashes in μ, respective structural relations, and ordering. Thus the communication cost on the side of the distributor is higher than what it would be if one were to send only S_R: it is in the order of $2 * (|\mu| + |V_R|) - 1$. Moreover (if structural order is necessary), the distributor also incurs the cost of sending information about the structural order among all these nodes. In the case of structural signatures, the communication cost is either $2 * (V_R) - 1$ (when the nodes and edges are shared) or V_R (when only the nodes are shared). The structural position of a node takes care of the structural order among nodes. Obviously communication (sending) cost for the structural signature is almost 50% of such cost incurred by Merkle hash technique.

Cost of Integrity Verification. Similarly, the user incurs less communication cost in case of the structural signature scheme than in the Merkle hash technique, while receiving the data sent by the distributor. Further in case of the Merkle hash technique, the user has to verify integrity by using a higher number of entities, that is, $2 * (|\mu| + |V_R|) - 1$, and the proportional amount of information about the structural order among all these nodes (if structural order is important). The best case of Merkle hash technique is when the integrity of the whole tree $T(V, E)$ is to be verified; in such case, obviously, no hash of any node is required and integrity verification using the Merkle hash technique has complexity $O(|V|)$, which is same as the integrity verification complexity using the structural signatures, $O(|V|)$. Thus for integrity verification in comparison to the Merkle hash technique, the structural signature scheme incurs less cost, except in the infrequent case in which the user has access to the whole tree (in such a case, the costs for both the schemes is identical - equivalent to the cost of signature generation). Our experimental results corroborate the fact that integrity verification at the user side using structural signature is more efficient than using Merkle hash technique.

9.3 Experimental Results

We implemented our structural signature scheme and Merkle hash technique for trees. Our implementation uses Java ($J2SE5.0$) on a IBM $T42$ Thinkpad with Windows XP, Intel Pentium M 1.60GHz and 512MB RAM. With no loss of

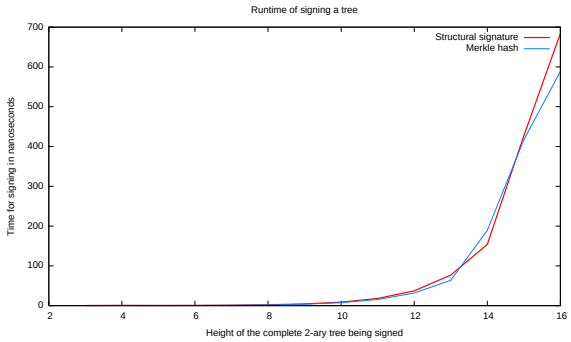

Figure 3: Time taken for signature generation of a complete 2-ary tree with respect to its height while using the Structural signature scheme and the Merkle signature scheme.

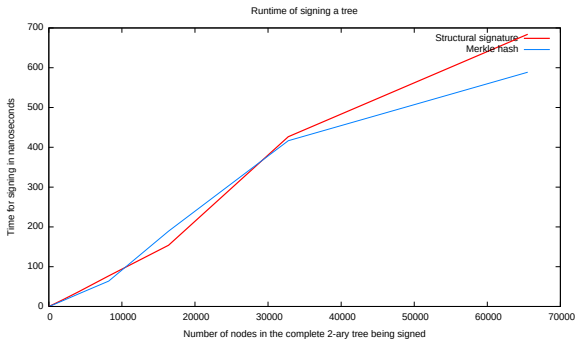

Figure 4: Time taken for signature generation of a complete 2-ary tree with respect to its number of nodes.

generality, we carry out our experiments on complete trees; the trees are 2-ary with 2 to 65535 nodes (in other words, the height is from 2 to 16).

Our experimental results show that the amount of time taken to generate the structural signatures for a tree is practically the same as the time required by the Merkle hash technique (Figures 3 and 4). The structural signature takes around 0.10 seconds more for a tree with 65535 nodes; it is quite negligible especially when signatures are generated *once* and re-used *many times*.

The time taken for user-side integrity verification is a significant factor because it affects the end-to-end response time at the user side and since integrity verification would be carried out by many users, the collective overhead would be very high. Our experimental results also show that the amount of time taken for integrity verification using our structural signature scheme is less than the time required by the Merkle hash technique (Figures 5 and 6). The subtree whose integrity verification has been carried out is a complete left-most subtree in a complete 2-ary tree of height 16. Our technique also behaves more efficiently as the size of the tree increases.

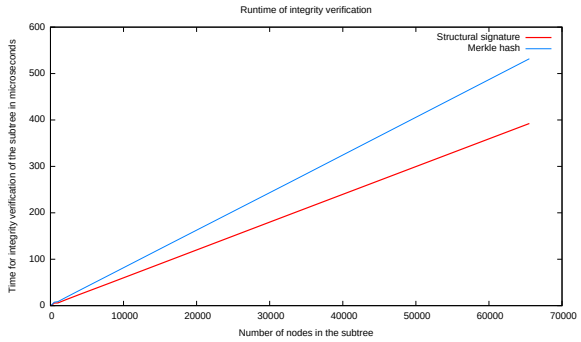

Figure 5: Time taken for integrity verification of a subtree with respect to its number of nodes.

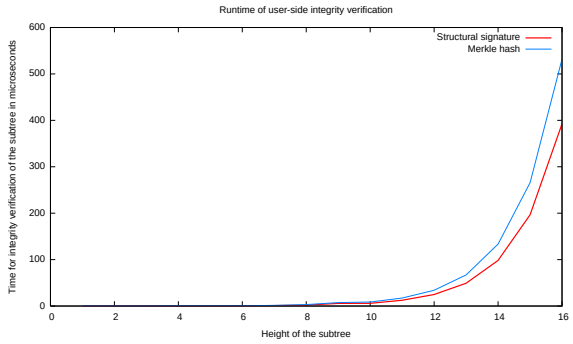

Figure 6: Time taken for integrity verification of a subtree with respect to its height.

9.4 Discussions

Dropping of Nodes. In order to detect that one or more nodes have been dropped in an unauthorized manner from the subtree, the distributor can create a hash of the structural position (or signature) of all the nodes in an order (such as BFS-order) known to the user and send it alongwith the data. The user then re-computes this hash from the nodes it has received. If both of the hashes match, then no node has been dropped.

Non-repudiation. The signatures (of a tree and its nodes) are certified by the trusted owner Alice or a trusted certificate authority (which binds the owner's name to the signatures during certification). Therefore Alice nor the certificate authority can refute the signature or the certificate of the tree and its nodes.

Dynamic Updates. In the structural signature scheme, there is no need to compute the signature of the tree at the user-side in order to verify the integrity of a node or a subtree. Therefore the signature of the old tree can be used for creating the structural signatures of the nodes that are inserted or updated. Use of the signature of the old tree prevents leakage of the fact that the tree has been updated. Such a leakage would occur, if a new signature of the tree is computed.

Upon insertion of a node, its structural position is computed such that the RPON and RRON preserve correct relationships with the parent and siblings of the node. Then the node, its content and the structural position are signed using the signature of the (original) tree, which is followed by the certification of the signature of the node. Insertion of a subtree is carried out as a sequence of insertions. Upon the deletion of a node or a subtree, the signatures of the nodes and the tree need not be re-computed. Insertion of an edge is generally followed by a deletion of an old edge and vice versa, in order to maintain the properties of a tree - connected and $n-1$ edges for n nodes. The nodes involved in the insertion of an edge from x to y (followed by deletion of z to y) requires re-computation of the signature of y with respect to x and discarding its earlier signature. Update of a node/edge and a subtree of m nodes incurs $O(1)$ and $O(m)$ cost, respectively. In order to accomodate insertions and maintain appropriate randomness ($\frac{1}{2^k}$), the randomized traversal numbers should be generated using η (Section 6.2) as the summation: $\sum_{1 \leq j \leq m_i} \eta_j$, where m_i is chosen randomly.

10. RELATED WORK

Information leakage is an important problem in data sharing and data analysis. Some of the contexts in which information leakage is currently being addressed are privacy-preserving databases [6, 24, 27, 29], automated trust negotiation [15], and error correction [9]. However, there is little work on information leakage through integrity verification and signature of data, especially trees.

Merkle [21] proposed a digital signature scheme based on a secure conventional encryption function over a hierarchy (tree) of document fragments. Since then, this technique has been used widely, but always with an authentication path of logarithmic size to verify even a single document fragment. It also leads to leakage of information (discussed in Section 2). Buldas and Laur [5] have also found that Merkle trees are binding (integrity-preserving) but *not* hiding (confidentiality-preserving).

The use of commutative hash operations (one-way accumulators [13]) to compute the Merkle hash signature prevents leakage related to the ordering among the siblings. However it cannot prevent the leakage of signatures of a node and the structural relationships with its descendants or ancestors. Moreover, one-way accumulation is very expensive (due to modular exponentiation) in comparison to the one-way hash operation.

The Merkle hash technique has been widely used in data authentication. Devanbu *et al.* used the Merkle hash technique for authenticating XML data [8]. Bertino *et al.* [4] proposed a technique based on the Merkle hash technique for selective dissemination of XML data in a third party distribution framework . Kocher [18] proposed to use Merkle hash trees for distribution to third parties. Goodrich and Tamassia [12] proposed authenticated dictionaries using skip lists and commutative hashing (one-way accumulators). Goodrich *et al.* [14] proposed techniques to authenticate graphs with specific path queries and geometric searching. Martel *et al.* [20] proposed a general model for authenticated data structures. For secure multicast, Perrig uses static data ordering over symmetric encryption [23].

Chatvichienchai and Iwaihara [6] proposed mechanisms for secure updates, without leading to information leakages. However such mechanism does not address the problem of information leakages during verification of integrity of partial XML documents. Wang *et al.* [26] treat structure and content as first-class protection units. However they focus on a sharing model, in which the receiver of the data has access to only the content (nodes) and *not* to the structural relationship between them. The paper proposed a scheme for securing structural information in XML databases: how to process queries on an encrypted XML database such that individual element content and structural relations are kept confidential if the security constraint specified requires so. In our case, we allow the receiver to have access to both nodes and the structural relationships between them.

Traversal numbers have been used for querying and navigation of XML data by Zezula *et al.* [28]. However they do not address any security issues. They use the non-randomized version of traversal numbers, which is unsuitable for security purposes. Traversal numbers have also been used for secure querying of data [26]. However they have not been used to define signatures for trees and graphs. Wang *et al.* [26] have used a notion similar to traversal numbers in defining the structural index in XML databases in order to be able to locate encryption blocks as well as their unencrypted data nodes that satisfy user query. They use real intervals $[0, 1]$ for root and every child of the root being assigned a sub-interval such as $[0.5, 0.6]$. The first entry in the interval can be assumed to be referring to the pre-order number and the second one to the post-order number. However they do not derive such an interval from traversal numbers nor do they use traversal numbers for signing trees. None of the previous approaches propose the use of randomized traversal numbers for the signature of trees. As such, the previous approaches do not include security analysis, performance evaluations nor detailed comparison with the Merkle hash technique and other secure data publishing techniques derived from the Merkle hash technique.

11. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed the notion of structural signatures for signing trees in order to assure integrity and maintain confidentiality, thereby providing stronger security guarantees than those provided by the Merkle hash trees and related techniques, which are widely used to assure data integrity. The notion of structural signatures is based on the simple notion of tree traversals and the fact that a combination of two tree traversals - post-order and pre-order can be used to uniquely reconstruct a tree (and any of its subtree). We also showed that the performance of structural signatures is as good as that of the Merkle hash technique both for signature generation and integrity verification. Thus with the equivalent cost, our technique supports stronger security guarantees than the Merkle hash. Moreover, structural signatures reduce the amount of data that need to be sent from a distributor to a user, by at the same time allowing the user to reconstruct the subtree from the nodes and to easily verify the integrity of the data. Like Merkle trees, structural signatures facilitate precise detection of integrity violations - the compromised nodes.

As future work, we plan to extend the structural signature scheme to pervasive devices so that the integrity of a tree can be verified efficiently at the device-side with less energy consumption. Further, we plan to explore signature schemes based on Zero-knowledge proofs; our preliminary

analysis indicates however that such techniques are much more expensive than the structural signature scheme.

Acknowledgement. We thank Prof. Mike Atallah for his comments on the paper. The authors have been partially supported by AFOSR grant FA9550-07-1-0041 - Systematic Control and Management of Data Integrity, Quality and Provenance for Command and Control Applications.

12. REFERENCES

[1] IBM XL C/C++. http://www-306.ibm.com/software/awdtools/xlcpp/.

[2] Java cryptographic architecture. http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2004. ACM.

[4] E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta. Selective and authentic third-party distribution of XML documents. *IEEE TKDE*, 16(10):1263–1278, 2004.

[5] A. Buldas and S. Laur. Knowledge-binding commitments with applications in time-stamping. In *Public Key Cryptography*, pages 150–165, 2007.

[6] S. Chatvichienchai and M. Iwaihara. Detecting information leakage in updating XML documents of fine-grained access control. In *Database and Expert Systems Applications*, 2006.

[7] S. K. Das, K. B. Min, and R. H. Halverson. Efficient parallel algorithms for tree-related problems using the parentheses matching strategy. In *Proceedings of the 8th International Symposium on Parallel Processing*, pages 362–367, Washington, DC, USA, 1994. IEEE Computer Society.

[8] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of XML documents. In *CCS '01*, pages 136–145, New York, NY, USA, 2001. ACM.

[9] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *STOC '05*, pages 654–663, New York, NY, USA, 2005. ACM.

[10] S. K. Goel, C. Clifton, and A. Rosenthal. Derived access control specification for XML. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 1–14, New York, NY, USA, 2003. ACM Press.

[11] O. Goldreich. *Foundations of Cryptography*, volume 1, Basic Tools. Cambridge University Press, 2001.

[12] M. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing, 2000.

[13] M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, pages 372–388, London, UK, 2002. Springer-Verlag.

[14] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Lecture Notes in Computer Science*, pages 295–313, Berlin / Heidelberg, 2003. Springer.

[15] K. Irwin and T. Yu. Preventing attribute information leakage in automated trust negotiation. In *CCS '05*, pages 36–45, New York, NY, USA, 2005. ACM.

[16] V. Kamakoti and C. Pandu Rangan. An optimal algorithm for reconstructing a binary tree. *Inf. Process. Lett.*, 42(2):113–115, 1992.

[17] D. E. Knuth. *The Art of Computer Programming*, volume 1. Pearson Education Asia, third edition, 2002.

[18] P. C. Kocher. On certificate revocation and validation. In *FC '98: Proceedings of the Second International Conference on Financial Cryptography*, pages 172–177, London, UK, 1998. Springer-Verlag.

[19] A. Kundu and E. Bertino. Secure dissemination of XML content using structure-based routing. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 153–164, Washington, DC, USA, 2006. IEEE Computer Society.

[20] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[21] R. C. Merkle. A certified digital signature. In *CRYPTO '89*, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[22] Matthew Morgenstern. Security and inference in multilevel database and knowledge-base systems. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 357–373, New York, NY, USA, 1987. ACM.

[23] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *CCS '01*, pages 28–37, New York, NY, USA, 2001. ACM.

[24] V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data publishing. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 531–542. VLDB Endowment, 2007.

[25] D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press, third edition, 2005.

[26] H. Wang and L. V. S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *VLDB'06: Proceedings of the 32nd international conference on Very large data bases*, pages 127–138. VLDB Endowment, 2006.

[27] R. C. Wong, A. W. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 543–554. VLDB Endowment, 2007.

[28] P. Zezula, G. Amato, F. Debole, and F. Rabitti. Tree signatures for XML querying and navigation. In *Database and XML Technologies*, pages 149–163, 2003.

[29] N. Zhang and W. Zhao. Distributed privacy preserving information sharing. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 889–900. VLDB Endowment, 2005.

APPENDIX

Table 1: Computation/verification of Merkle hash signature of S_2 in the health record in Figure 1(b)

Node a	*Nodes whose Merkle hash used in this particular order to compute/verify Merkle hash of a*	*Distinct information leakages during verification of a*
a_{13}	a_{13}	none
a_{12}	a_{12}	none
a_{10}	a_{12}, a_{13}, a_{10}	none
a_8	a_{10}, a_{11}, a_8	signature of a_{11} , a_{11} as sibling of a_{10}, a_{11} as child of a_8 , a_{11} as to the right of a_{10}
a_5	a_7, a_8, a_9, a_5	a_7-specific *leakage*: signature of a_7 , a_7 is sibling of a_8 , a_7 is child of a_5), a_7 is to the left of a_8); a_9-specific *leakage*: signature of a_9 , a_9 is sibling of a_8 , a_9 is child of a_5 , a_9 is to the right of a_8
a_4	a_5, a_6, a_4	signature of a_6 , a_6 is sibling of a_5 , a_6 is child of a_4 , a_6 is to the right of a_5

Table 2: Inference of sensitive information from the leakage during the integrity verification of S_2 in Figure 1(b)

Leaked information during verification of a node	*Inference from the leakage in the health-care context*
signature of a_{11} `AND` (a_{11} as sibling of a_{10} `OR` a_{11} as child of a_8)	Patient has gone through another Chemotherapy.
a_{11} as to the right of a_{10}	If sibling order represents more information such as temporal order, then more sensitive information can be derived such as it can be inferred if the chemotherapy referred to by node a_{11} was administered earlier or later than the one referred to by a_{10}.
signature of a_7 `AND` (a_7 is sibling of a_8 `OR` a_7 is child of a_5)	Patient has gone through another type of treatment; also inferred is - it to be either Surgery or Medication
a_7 is to the left of a_8	More leakage related to the order such as temporal order
signature of a_9 `AND` (a_9 is sibling of a_8 `OR` a_9 is child of a_5)	Patient has gone through another type of treatment; also inferred is - it to be either Surgery or Medication
a_9 is to the right of a_8	More leakage related to the order such as temporal order
signature of a_6 `AND` (a_6 is sibling of a_5 `OR` a_6 is child of a_4)	Patient suffers from another critical disease; can be determined which disease it is from the specialty of the hospital
a_6 is to the right of a_5	More leakage related to the order such as temporal order: time of treatment of this disease in this hospital relative to the time of treatment of Cancer