Leakage-Free Integrity Assurance for Tree Data Structures

Ashish Kundu and Elisa Bertino
Department of Computer Science, Purdue University, USA.
{ashishk, 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 the 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 leakage of information. Most existing techniques for the integrity of hierarchical data structures are based on the Merkle hash technique and thus suffer from the problem of information leakages. We describe the types of leakages and inference attacks that can be carried out on the Merkle hash technique, in the context of integrity assurance. 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 an integrity assurance scheme for tree data 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 integrity assurance 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. Techniques for computing randomized traversal numbers are also described 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 show through complexity and performance analysis that the structural integrity assurance scheme is efficient; with respect to the Merkle hash technique, it incurs comparable cost for signing trees and incurs lower cost for user-side integrity verification. Further, we extend the proposed technique in order to assure integrity of weighted trees and dynamic updates. We also show how the proposed structural integrity assurance technique can be applied in order to precisely detect integrity violations as well as to efficiently recover data. Such techniques have applications in digital forensics and efficient data transmission.

1 Introduction

Data sharing among multiple parties with high integrity assurance is an important problem [10, 12, 11, 18, 28, 24, 3]. 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 is organized. Because one of the most widely used data organization structures is the tree structure (see the XML-based example in Section 4), 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 [14]. 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 application domains, such as healthcare and military, an additional requirement is to maintain the confidentiality of the content and the structural information [35]. 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 [30] is the most well known integrity assurance technique for tree structures and has been widely extended for use in content distribution systems and data publishing [3, 10, 15, 18, 23, 28]. A drawback of such technique is that the integrity verification process does not preserve confidentiality. The

Merkle hash technique is binding (integrity) but not hiding (confidentiality) [5]; therefore it is vulnerable to inference attacks[1]. The Merkle hash 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 Merkle hashes of a set of nodes/subtrees in the tree has to be provided to the user, even if the user does not have access to these nodes/subtrees. By the mere fact that such hashes 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 4.

More specifically, the integrity verification of a subtree T_δ, which belongs to a tree T by using the Merkle hash technique reveals: (1) the Merkle hash of some nodes which are in T but not in T_δ; (2) the structural relationship between a node x in T_δ and some node y which is in T but not in T_δ; and (3) the relative (structural) order between a node x which is in T_δ and y, which is in T but not in T_δ. 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 approach 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 T_δ (of T) that Bob is authorized to access. Any information about a node which is in T but not in T_δ, its signature, its structural relationship with any other node in T should *not* be revealed to Bob. Obviously the Merkle hash technique cannot be 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 (signed) 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.

In this paper, we propose an integrity assurance 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 integrity assurance scheme", is based on the structure of the tree as defined by depth first traversals (pre-order, post-order, in-order) of the tree. The scheme assigns randomized traversal numbers (post-, pre- and in-order numbers) to each node, which uniquely represents the relative position of the node in the tree. Techniques on how to compute randomized traversal numbers are also presented in the paper. We formally prove that the structural integrity assurance scheme is secure, that is, it not only protects against violations of content and structural integrity, but also against information leakages. Complexity and performance analysis shows that our 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. Further, we extend the proposed technique in order to assure integrity of weighted trees and dynamic updates without leaking weights or the updates. As an application to digital forensics, we show how the proposed structural integrity assurance technique can be used in order to precisely and efficiently detect integrity violations. Our technique incurs $O(n)$ cost while the state-of-the art technique [16] incurs $O(n^3 logn)$. Such a capability further enables efficient recovery of data, which is often a requirement

[1]The inference problem is a widely investigated problem in computer and information security [31]. 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

in secure and unreliable data transmission channels, such as satelite communication.

Novelties of the structural integrity assurance scheme The structural integrity assurance 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 the time it requires to compute integrity verifiers 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.

Outline of the Paper The paper is organized as follows. The security model is described in Section 2. The notations most commonly used in the paper are reported in Table I. The data distribution and security model is presented in Section 2. The Merkle hash technique and related inference attacks are briefly summarized in the Section 3. Section 4 presents a running example. Section 5 introduces the notion of randomized traversal numbers and two key lemmas. These lemmas are used to define the notion of structural integrity verifiers in the next section (Section 6). This section also defines the algorithm for signing a tree and verifying the content and structural integrity of subtrees received by a user. Schemes for distribution of subtrees are described in Section 7. Section 8 illustrates our scheme using the running example. Security analysis and performance of the structural integrity verifiers are discussed in Section 9 and 10, respectively. Sections 11 and 12 extend the structural integrity assurance scheme to dynamic updates and weighted trees. In Section 13, applications of the proposed technique to digital forensics and efficient data recovery are discussed. Related work is presented in Section 14. Finally, Section 15 concludes the paper. How to compute randomized traversal numbers and the security of addition of random numbers are described in Appendix A.

2 Model

The most commonly used notations in the paper are defined in Table I.

Data: A directed (rooted) tree $T(V, E)$. A node represents an atomic unit of the data that can be shared with a user. A structural relationship is either a relationship represented by an edge between two nodes or a structural order between the two nodes. In a tree, siblings, that is, nodes with a common parent node, are structurally ordered.

Trusted Owner: The trusted owner Alice ($\mathcal{A}$) of a data object organized as tree wants to sign it *once* so that the data object can be searched/queried/published *many times*.

Third-party Distributors: After signing T, Alice may delegate the job of publishing T or processing queries over T to third-party distributors, which do not have signature authority. Each distributor D stores all the pre-computed integrity verifiers (also referred to as IVs) for the data; D is also responsible for sending appropriate IVs to users alongwith the subtree sent to the user. D does not have the authority to sign, and D need only be trusted so much that it acts correctly with respect to query processing.

Users: A user Bob ($\mathcal{B}$) receives a subtree of a tree as a result of publication of the data, or a search/query initiated by $\mathcal{B}$. Any non-empty subtree of a tree can be a valid data object returned to $\mathcal{B}$ by a distributor. Each user runs a software process, called (integrity) prover, that is capable of integrity verification of received

Table 1: Commonly used acronyms and notations.

Notation	*Meaning*
PON, RON, ION	Post-order number, Pre-order number In-order number, respectively.
RPON, RRON, RION	Randomized PON, Randomized RON Randomized ION, respectively.
IV	Integrity Verifier (*e.g.* a signed hash).
ψ_x	structural integrity verifier (IV) of node/tree x.
ρ_x	structural position of node x.
$\mathcal{H}$	Cryptographically secure one-way hash function: $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{\vert\mathcal{H}\vert}$.
$T(V, E)$	A directed rooted tree with a set of vertices V and set of edges E.
$T_\delta(V_\delta, E_\delta)$	A subtree with a set of vertices V and set of edges E.
$e(x, y)$	A directed edge from node x to y.
c_v	Content of node v.

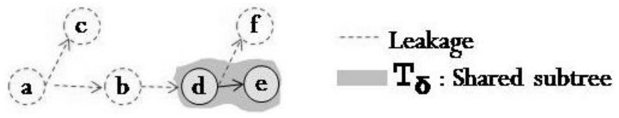


Figure 1: An example tree with each node having some content. The leakages are with respect to Merkle hash technique (Section 3).

subtrees.

Integrity requirements: A user Bob ($\mathcal{B}$) (its integrity prover) can verify the integrity of the subtree that it receives. Definition 2.1 defines the semantics of integrity of a subtree.

Definition 2.1 (Integrity of Trees). Integrity of a subtree of a tree has not been compromised if and only if none of the following entities in the subtree has been modified:

- the content of each node;
- each (directed) edge;
- each structural order existing between nodes.

An attack that modifies the contents of nodes or the structural information is called a data tampering attack. The above integrity requirements are to detect any such attacks carried out on data.

Confidentiality requirements: User $\mathcal{B}$ must not be able to infer any extraneous information in a tree with respect to the received subtree T_δ of tree T of (defined by Definition 2.2 below) as part of the integrity verification process.

Definition 2.2 (Extraneous Information). Extraneous information in a tree $T(V, E)$ with respect to a subtree $T_\delta(V_\delta, E_\delta)$ comprises: (1) each node y (referred to as extraneous node) such that $y \in V$ and $y \notin V$; and (2) each edge e (referred to as extraneous edge) such that $e \in E$ and $e \notin E$.

The auxiliary information in the context of Merkle hash technique is also extraneous information. For example, consider Figure 1 and suppose that the user receives subtree T_δ. The extraneous information in T with respect to the T_δ comprises of the nodes a, b, c and f, and edges $e(a, b)$, $e(b, d)$, and $e(d, f)$, and the structural order between a pair of nodes such as (b, c) and (e, f).

3 Merkle Hash Technique

The Merkle hash technique [30] works bottom-up. For a node x in tree $T(V, E)$, it computes a Merkle hash (MH) $mh(x)$ as follows: if x is a leaf node, then $mh(x) = \mathcal{H}(c_x)$; else $mh(x) = \mathcal{H}(mh(y_1)\|mh(y_2)\|\ldots\|mh(y_m))$, where y_1, y_2, $\ldots$,y_m are the m children of x in T in that order from left to right. For example, consider the tree in Figure 1. The Merkle hash for this tree is computed as follows. The MH of e and f are computed as $\mathcal{H}(c_e)$ and $\mathcal{H}(c_f)$, respectively, which are then used to compute the MH of d as $mh(d) = \mathcal{H}(mh(e) \| mh(f))$. The MH of b is computed as $\mathcal{H}(mh(d))$. Similarly the MH of c and a are computed as $\mathcal{H}(c_c)$ and $\mathcal{H}(mh(b) \| mh(c))$, respectively.

By using such technique, only the contents of the leaf nodes can be authenticated. In order to account for the contents in non-leaf nodes, two simple variants of the Merkle hash technique can be used to compute the MH of a non-leaf node from the MH of its children and the contents (or hash of the content) of the non-leaf node itself. Suppose x to be a non-leaf (thus a root/intermediate) node in T. The MH of x is defined as follows: $mh(x) = \mathcal{H}(\mathcal{H}(c_x)\|mh(y_1)\|mh(y_2)\|\ldots\|mh(y_m))$.

Consider again the tree in Figure 1. MH of d, b and a are computed respectively as $\mathcal{H}(\mathcal{H}(c_d)\|mh(e)\|mh(f))$, $\mathcal{H}(\mathcal{H}(c_b)\|mh(d))$, and $\mathcal{H}(\mathcal{H}(c_a)\|mh(b)\|mh(c))$.

3.1 Integrity Verification

Let T_δ be a subtree of tree T to be shared with a user. The following *auxiliary information* is also sent to the user, for integrity verification of T_δ (Consider the subtree T_δ in Figure 1:

1. With respect to each node in T_δ, MH of its siblings[2] that are in T but not in T_δ. For example, in Figure 1, $mh(f)$ (with respect to e) is sent.

2. With respect to each node x in T_δ, the MH of each sibling of each ancestor of x, if that sibling is not in T_δ. In our example, $mh(a)$ and $mh(b)$ are also sent.

3. With respect to each node in T_δ, the hash of the content of each of its ancestor. In our example, $\mathcal{H}(c_a)$ and $\mathcal{H}(c_b)$ are sent to the user.

4. The structural order between a node in T_δ and its sibling(s) that are not in T_δ, and the structural order between the sibling nodes that are not in T_δ. In our example, the order between e and f, and the order between b and c are sent to the user.

5. Parent-child/ancestor-descendant relationship(s) between a node in T_δ and another node not in T_δ (such as the relationship between b and d), and those between the nodes that are not in T_δ (such as between a and b, and between a and c).

6. The fact that a given node is the root of T (even if it is the root of T_δ). In our example, a is the root of the tree and this fact is conveyed to the user.

The user then computes the MH of the whole tree using such information (the subtree and auxiliary information) and compares it with the received signed MH of the root. If they are equal, the integrity of the subtree is validated. Moreover, this process authenticates the subtree against the original tree.

3.2 Inference Attacks

The attacks on the Merkle hash technique are based on the set of *auxiliary information* sent to the user. By exploiting the knowledge of these information, inference attacks described below can be carried out on the MHT.

- *(plaintext, ciphertext)-inference attack*: It exploits the information (1), (2), (3), and (6). By comparing the Merkle hash of a node e in the shared subtree with the MH of another node f received as part of the auxiliary information, the user can infer whether contents of e is same as that of f and if the subtree with root e is identical to the subtree with root f. With the auxiliary information (6), the user

[2]Nodes that are siblings in a tree have a common parent.

can also infer (a) whether the received subtree is in fact the original tree, and (b) whether the root of received subtree is in fact the root of the tree.

- *(ciphertext, ciphertext)-inference attack*: It exploits the information (1), (2), (3), and (6). By comparing the Merkle hashes of two nodes (c and f) that are received as part of the auxiliary information, the user can infer whether the contents of c is same as that of f and whether the subtree with root c is identical to the subtree with root f. With the auxiliary information (6), the user can also infer whether the received subtree is in fact the original tree.

- *Structural inference attack*: It exploits the information (1), (2), (3), and (5). The user infers the number of nodes that are not in the received subtree, and the structure of the original tree from the auxiliary information and shared subtree. If the user receives non-empty auxiliary information, then it infers that the shared subtree is a proper subtree of the original tree and there are nodes it has not received. In some cases, the user can also learn about the exact size of the original tree (such as in the case of our example in Figure 1).

- *Missing-siblings inference attack*: It exploits the information (1) and (2). From the auxiliary information, the user infers the number of siblings of a received node e that are not in the shared subtree. If the shared subtree has x and y as siblings, the user also infers the number of siblings that are not in the shared subtree but are to the right of x and to the left of y in the original tree.

- *Structural-order inference attack*: It exploits the information (4) and (5). The user infers structural order between siblings involving one or more nodes that the user does not have access to. In our example, the user learns that b and e are left siblings of c and f, respectively.

- *Parent-child inference attack*: It exploits the information (4) and (5). The user infers the parent-child relationships involving one or more nodes that the user does not have access to. In our example, the user learns that b and d are the parents of e and f, respectively. The user also learns that b is an ancestor of e and f.

3.3 Variants of Merkle Hash Technique

By using one-way accumulators instead of one-way hashes in order to compute the Merkle hash of an intermediate node, the order between either siblings or ancestor-descendants can be factored out of the construction of the digital signature. However, such a technique when applied to siblings of a node cannot prevent leakage of ancestor-descendant information and vice versa.

By salting [21] or by MD-strengthening [29, 8] the Merkle hash of each node one is secured against the known-plaintext and known-ciphertext inference attacks. However, salting or MD-strengthening cannot prevent structural inference attacks completely even when coupled with one-way accumulators. Another consideration would be to concatenate the depth of a node to its contents when computing its Merkle hash. However, the knowledge of depth of a node is a leakage of structural properties and makes the technique vulnerable to structural inference attacks.

By its inherent nature, the Merkle hash technique, leaks certain information about the contents and/or the structure of the signed tree. It is thus necessary to develop a new integrity assurance mechanism that can prevent such leakages and attacks. In the next section, we give an example in order to illustrate the significance of such leakages and inference attacks.

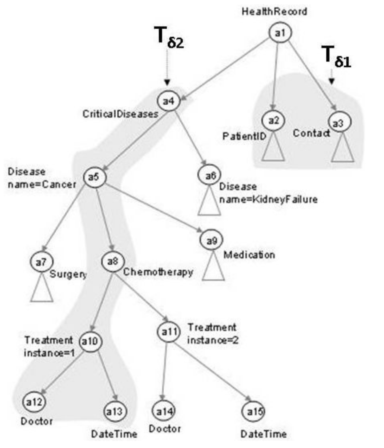
4 Running Example

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 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 2 is a fictitious 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>
    . . .
    <PatientID id=2345S>
    . . .
    </PatientID>
    <Contact>
        . . .
    </Contact>
    <CriticalDiseases>
        <Disease name=Cancer>
            <Surgery>
            . . .
            </Surgery>
            <Chemotherapy>
                <Treatment instance=1>
                    <Doctor name=Dr. S. Stevens/>
                    <DateTime date=. . . time=. . . />
                </Treatment>
                <Treatment instance=2>
                    <Doctor name=Dr. M. Paul/>
                    <DateTime date=. . . time=. . . />
                </Treatment>
            </Chemotherapy>
            <Medication>
            . . .
            </Medication>
        </Disease>
        <Disease name=KidneyFailure>
            . . .
        </Disease>
    </CriticalDiseases>
    . . .
</HealthRecord>
```

(a)



(b)

Figure 2: (a) XML-based Health-care record of a patient. (b) The tree representation of the HealthRecord.

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 associate a label with each node in the health record in Figure 2(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 II lists the details about how to compute and verify the Merkle hash for each node in the tree in Figure 2(b). The third column of such table also lists the information which is leaked when the integrity of a node is verified. Table III lists the inference attacks that can be carried out due to the leakages.

Consider the following scenario. A cashier has access to the subtree $T_{\delta 1}$, shown in Figure 2, including the root a_1, that is 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 2(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 a_4 and she also receives the following information: a_4 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 $T_{\delta 2}$ and a_1 from the record in Figure 2. He has access to $T_{\delta 2}$, because he works with the doctor *S. Stevenson*, who prescribed the administering of this treatment. The nurse receives $T_{\delta 2}$ and a_1, and the corresponding signature information from the remote database. In order to be able to verify the integrity of $T_{\delta 2}$, 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_8 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.

Table 2: Computation/verification of Merkle hash signature of T_δ in the health record in Figure 2(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

5 Randomized Traversal Numbers

In this section, we review tree traversals and define the notion of randomized traversal numbers.

Table 3: Inference of sensitive information from the leakage during the integrity verification of T_δ in Figure 2(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

5.1 Review of Tree Traversals

Post-order, pre-order, and in-order tree traversals are defined in [22]. 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. 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. It is possible for an adversary to exploit this information and replace a signed node with a compromised or a different node altogether by assigning to it 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.*

5.2 Randomized Traversal Numbers

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 is preserved, while the order among those assigned to siblings does not need to be preserved.*

Techniques to compute randomized traversal numbers are described in Section 6.3. The randomized transformations of post-order, pre-order and in-order numbers are called as randomized post-order (RPON), randomized pre-order (RRON), and randomized in-order (RION) numbers. RPON, RRON, and RION for a node x are denoted by p_x, r_x and i_x, respectively.

The following lemmas provide the basis for defining the notion of structural integrity verifiers 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 [20]. 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 [9]. □

6 Integrity Verification of Trees

In this section, we develop structural integrity verifiers (IVs) for trees based on Lemma 5.3. Structural IVs 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 exposition, we focus primarily on non-binary trees.

6.1 Structural Integrity Verifiers

A structural position uniquely identifies a node in a tree structure. 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 (Definition 6.1). Because the nodes should be bound to a given tree, a unique identifier referred to as ψ_T is assigned to the tree. ψ_T is a random value and can be computed as the 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(Definition 6.2). The hash is further signed by a trusted entity (the owner or a certifying authority). The hash can be salted using a random value if the fact that "the received subtree (sent to the user) is the same as the original tree" is a sensitive information. The (salted) tree IV is publicly available or passed to the user alongwith the subtree the user has access to. The structural IV of a node x in tree T is defined as a hash of the structural position and content of x and the (salted) IV of the tree ψ_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 IV 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 ω be a cryptographically secure random. Let $\mathcal{H}$ denote a one-way cryptographic hash function. The structural integrity verifier of T, denoted by ψ_T, is defined as $\psi_T = \mathcal{H}(\omega \| \rho_1 \| c_1 \| \rho_2 \| c_2 \| \ldots \| \rho_i \| c_i \| \ldots \| \rho_n \| c_n)$.*

Definition 6.3. *Let x be a node in tree $T(V,E)$. The structural integrity verifier of x, denoted by ψ_x, is defined as $\psi_x = \mathcal{H}(\psi_T \| \rho_x \| c_x)$.*

6.2 Computation of Integrity Verifiers

The algorithm that the trusted owner Alice follows in order to compute the IVs for a tree $T(V,E)$ is given below.

Algorithm

1. Compute the post-order and pre-order numbers for each node in T.
2. For each node x in T: 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 IV of the tree T, ψ_T from a specific sequence of vertices (such as the post-order sequence which can be available from steps 1 and 2).
5. For each node x in V, compute the IV ψ_x.

After the IVs are generated, Alice signs[3] ψ_T and ψ_x of each node x. Let *signed*(ψ_T) and *signed*(ψ_x) refer to the signed IVs of T and x, respectively.

6.3 Computation of Randomized Traversal Numbers

Computation of randomized post-order and pre-order numbers (Steps 1 and 2 in the algorithm in Section 6.2) 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.

Randomized traversal numbers can be computed using one of the following three techniques.

6.3.1 Sorted random numbers

For a tree $T(V,E)$, compute n $(=|V|)$ number of secure random numbers, and sort them in a *working list* according to increasing order. Assign the lowest number in the working list to the node being visited in pre-order or post-order and remove it from the list. The cost of such a technique is $O(nlogn)$.

6.3.2 Order-preserving encryption

Assign real numbers to each node as traversal numbers; thus the PON and RON are real numbers such that the order between PONs and RONs are preserved. Apply an order-preserving encryption scheme [1] to the set of PONs/RONs with a unique key for every tree (so that encryption of a given number is unique across trees).

[3]Structural IVs 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.

6.3.3 Addition of random numbers

A randomized traversal number t_2 larger than another randomized traversal number t_1 is computed by adding a random η to t_1. Each random is treated as a non-negative value.

1. Choose a cryptographically secure random η.
2. $t_1 \leftarrow \eta$.
3. For $i = 2$ to n
   (a) Choose a cryptographically secure random η.
   (b) $t_i \leftarrow t_{i-1} + \eta$.

In order to accommodate insertions and maintain appropriate randomness, the randomized traversal numbers should be generated using η as the summation: $\eta \leftarrow \sum_{1\leq\ j\leq\ m_i} \eta_j$, where m_i is chosen randomly.

The following lemma proves the security of such a technique to compute random numbers. The proof is in Appendix A.

Lemma 6.4. *The addition of two cryptographically secure random numbers is a cryptographically secure random number.*

7 Distribution of a Subtree

After the IVs are computed and signed, the tree is ready to be shared with Bob. Suppose that Bob wishes to access T and thus sends such a request to a distributor. Bob has the authorization to access the subtree $T_\delta(V_\delta, E_\delta)$. $T_\delta(V_\delta, E_\delta)$ can be shared with Bob according to two different strategies: (1) by sharing the signed subtree - its nodes and the structure; (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 options in the following sections. Later we show how to use aggregate signatures in conjunction with either of these two strategies to share the subtree. By use of aggregate signatures, the distributor D needs to only send $O(1)$ integrity verifiers to the user.

7.1 Sharing the Subtree along with its Structure

The distributor D sends to Bob, who has access to the tree $T_\delta(V_\delta, E_\delta)$:

- each node x in V_δ;
- $\langle \rho_x, signed(\psi_x) \rangle$ of each node x in V_δ;
- information about the parent-child and ordering between nodes (e.g. in the form of an adjacency matrix);
- the structural IV of the tree if it is not publicly available.

Bob receives the subtree and it refers to it as $T'_\delta(V'_\delta, E'_\delta)$ (with a different name in order avoid any ambiguity). Bob is aware of the $\mathcal{H}$ function used. He verifies the integrity of each node. Next, he verifies the integrity of structural relationships among all those nodes in $T'_\delta(V'_\delta, E'_\delta)$, whose integrity and authenticity have been correctly verified.

7.1.1 Validation of Integrity Verifiers

Bob verifies the signature of the IVs of T and the IV of each node in T_δ. If the signature is valid, then the IV of the node is valid. A spurious node would not be signed by a trusted entity; so such a node would be detected during this process.

7.1.2 Integrity Verification for Contents and Integrity Verifiers

1. For each node y in the set of received nodes V'_δ, Bob computes the structural IV of y from its position, that is, it computes $\mathcal{H}(\psi_T \| \rho_y \| c_y)$. Then it compares this value with the IV ψ_y with which y has been signed. The verification proceeds if the values are equal.

7.1.3 Integrity Verification for Structural Relations

The 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 T'_δ.

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.

7.2 Sharing a Subtree - only the Nodes

An advantage of the use of structural IVs 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 [9]) (in-order and pre/post-traversals for binary trees [20]).

The distributor D sends to Bob, who has access to the tree $T_\delta(V_\delta, E_\delta)$:

- each node x in V_δ;
- $\langle \rho_x, signed(\psi_x) \rangle$ of each node x in V_δ;
- the structural IV of the tree if it is not publicly available.

D does not send any parent-child relationship or ordering between nodes to Bob.

The following section describes how to verify the integrity and re-construct the structure of the received subtree.

7.2.1 Integrity Verification and Reconstruction of a Subtree

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 [9] 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 algorithm given in [20] can be used, where RIONs would be used.

1. Validate the signatures of the tree and the nodes by verifying their certificates.

2. Verify the integrity of content and structural positions of the nodes as per the procedure in Section 7.1.2.

3. Apply the algorithm by Das et al. [9] (Section 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. [9], the consecutive nodes in post-order (pre-order) all the nodes with RPONs (RRONs) that are next to the other.

7.3 Using Aggregate Signatures

In the previous distribution schemes (Section 7.1 and 7.2), the distributor sends $O(n')$ units of IVs (one for each node) to the user, where n' is the number of the nodes in the subtree $T'(V',E')$. In this section, we show how to use aggregate signatures to provide an optimal distribution technique that sends only $O(1)$ units of IV units to the user. The following section gives a brief summary of aggregate signatures adapted from [2].

7.3.1 Review of aggregate signatures

Let $G_1 =< P >$ be an additively-written group of prime order p, and let G_2 be a multiplicatively written group of the same prime order p. A mapping $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map if (i) $e(aX, bY) = e(X,Y)^{ab}$ for all $X, Y \in G_1$ and $a, b \in Z_p^*$; and (ii) $G_2 =< e(P,P) >$. The mapping e is efficiently computable, but given only P, aP, and X (but not a) it is computationally infeasible to compute aX (i.e., the Computational Diffie-Hellman problem is difficult in G_1). This difficulty is what enables the signature and aggregate signature schemes based on bilinear pairings.

In this paper, we use the aggregate signature scheme by Boneh at al. [4]. In such scheme, the signer's secret key is $s \in Z_p^*$, $Q = sP \in G_1$ is public, and the signature for a message m is sM with $M = \mathcal{H}(Q,m) \in G_1$, where $\mathcal{H}$ is a cryptographic one-way hash function; for convenience, we henceforth omit mention of the $M = \mathcal{H}(Q,m)$ and simply say "message M". In the aggregate signatures, given the public P and Q, and given k message-signature pairs $M_i, S_i = sM_i$, $1 \leq i \leq k$, the signature is verified by checking that the following equality holds: $e(Q, \sum_{i=1}^{k} M_i) = e(P, \sum_{i=1}^{k} S_i)$.

7.3.2 Sharing

Distributor D sends the following to the user[4]:

- each node x in V' and ρ_x
- if structure needs to be sent (i.e. strategy (2)), information about the parent-child and ordering between nodes (e.g. in the form of an adjacency matrix)
- the structural IV of the tree if it is not publicly available, D computes $\psi_{T'(V',E')}$ from the aggregate signatures of the tuples associated with each node in T' as follows:

$$\psi_{T'(V',E')} = \sum_{x \in V'} M_x = s \sum_{x \in V'} \mathcal{H}(Q \parallel \psi_T \parallel \rho_x \parallel c_x).$$

7.3.3 Integrity Verification of a Subtree

When a user receives an n''-node subtree $T''(V'', E'')$ (along with the ρ_x' for each node x) from the distributor D, it needs to receive only $O(1)$ items of information, which is the IV $\psi(T'(V',E'))$ that D computes in $O(n'')$ time.

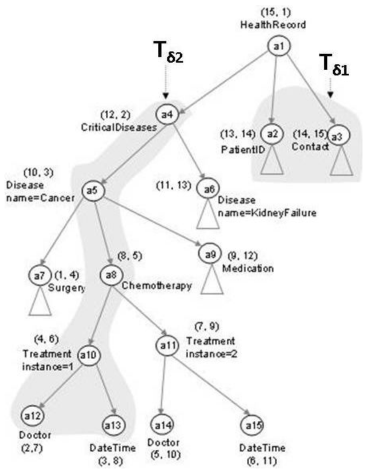
In order to verify the integrity of the contents (as well as the structural position) of the nodes in T'', the user computes

$$\psi_{T''(V'',E'')} = \sum_{y \in V''} (\mathcal{H}(Q \parallel \psi'_T \parallel \rho'_y \parallel c'_y))$$

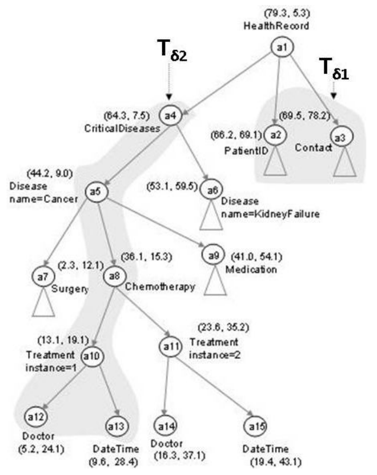
and then checks that the following equality holds:

$$e(Q, \psi_{T''(V'',E'')}) \stackrel{?}{=} e(P, \psi_{T'(V',E')})$$

where $e, P, \mathcal{H}$ and Q are public in an aggregate signature framework.



(a)



(b)

Figure 3: (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).

8 Illustration

Consider the tree in our running example (Figure 2) and suppose that the tree has been assigned post and pre-order numbers (Figure 3(a)) and their randomized counterparts (Figure 3(b)). The structural position of each node is represented as (RPON, RRON) in Figure 3(b). Each node has a content that consists of the name of its corresponding element and attribute-value pairs. Since the IVs 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 subtree $T_{\delta 1}$. The database D sends two nodes - a_2 and a_3, their structural IVs alongwith the (salted) tree-signature ψ_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 ψ_T, 66.2, 69.1 and c_2). Then she verifies whether the resulting value is equal to the received integrity verifier of node a_2; if this is the case, then the integrity of the node is verified. The same procedure is followed for a_3. Then because a_2 and a_3 were sent as siblings, the cashier verifies whether p_2 (=66.2), the RRON of a_2 is smaller than p_3(=69.5), the RRON of a_3; 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 $T_{\delta 2}$; however suppose he receives a $T_{\delta 2}$ 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 discussed in Section 7.1.3. 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.

9 Security Analysis

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

[4]In this section, we have used a slightly different notation $T'(V', E')$ to represent a subtree than the one used in previous section - $T_\delta(V_\delta, E_\delta)$ in order to avoid complicated notation involving double subscripting.

9.1 Integrity

Lemma 9.1. *Given that the hash function $\mathcal{H}$ is cryptographically secure, any integrity violation of the content and/or structural position of a node in a tree can be detected by using structural integrity verifiers.*

Proof. Let x be a node in tree T. Any compromise of the content c_x or the structural position ρ_x of a node x in T would invalidate the structural IV ψ_x, which is a hash of a message that contains c_x and ρ_x, unless the hash function $\mathcal{H}$ encounters a collision, which contradicts our assumption. Any unauthorized re-ordering of 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 tree T', different from T, but claimed to belong to T. Such a forgery is possible when the IV of T, ψ_T, is identical to that of T', $\psi_{T'}$. ψ_T and $\psi_{T'}$ are identical only when the one-way hash $\mathcal{H}$ has encountered a collision, which contradicts our assumption (and the Random Oracle Hypothesis [34]). Can such a tree T' be found such that a collision be deliberately generated? By the property of $\mathcal{H}$, it is "hard" to do so (under the Random Oracle Hypothesis). □

Countering Integrity Attacks

In order to allow the user to detect whether one or more nodes have been dropped in an unauthorized manner from the subtree, the distributors can employ the distribution scheme using the aggregate signatures (Section 7.3). When the distribution scheme does not use that distribution scheme, the distributor creates a hash of the structural position (or the IV) 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 the hashes match, then no node has been dropped.

The distributor can also create an aggregate signature as described in Section 7.3.

A spurious node would not be signed by the owner; so such a node would be detected during this process (Section 7.1.1).

9.2 Leakage

Suppose that Bob has access to a subtree $T_\delta(V_\delta, E_\delta)$ in T. Let x and y be two immediate siblings in T_δ, 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 (as proved by the following lemma).

Lemma 9.2. *Given that the hash function $\mathcal{H}$ is cryptographically secure, the structural integrity verifiers do not lead to any leakage of extraneous information.*

Proof. Suppose that a user Bob has access to $T_\delta(V_\delta, E_\delta)$, a subtree in T. Bob has access to the subtree, the structural IV of T, the IV of each node in T_δ, and the structural position of each node. Any leakage would be a direct leakage through such information or an inference from it. By Definition 2.2, extraneous information can be categorized as follows: (1) node IVs, and information about the (2) existence of nodes, (3) structural relations or (4) structural order among nodes.

Direct leakage: Clearly (as per Definitions 6.2 and 6.3, and the protocols specified in Section 6) Bob does not need to know the IV of any node (u) that is in T but not in T_δ. He therefore does not need to know any of the structural relationships and structural ordering that exist in T, but not in T_δ. 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 IV of the tree and IVs of the nodes in T_δ: Under the Random Oracle Hypothesis, the structural IV of the tree reveals neither (1) the existence of u nor (2) the IV of u. Similarly, the structural IV of a node leaks neither (1) and (2). Therefore (3) the structural relations (edges or paths) and (4) the structural order among nodes in T_δ and u are not revealed by the IVs.

The structural positions (RPON's and RRON's) of the nodes in T_δ: By Lemma 6.4, RPON's and RRON's are cryptographically secure random numbers; therefore the attacker cannot learn anything about (2) - the existence of node u between two immediate siblings.

Therefore the structural positions of nodes, that is, the RPON's and RRON's, cannot be used to determine the structural IV 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. Thus the lemma is proven. □

Comparison with Merkle hash In the Merkle hash technique and its derivatives, there is a release of $log(n)$ information, both in terms of content and structure. So the probability information leakage is 1. For structural IVs, there is no direct leakage of IVs of nodes and relationship between nodes that the user do not have access to. Moreover, as we quantified above, in our technique, inference attacks cannot determine the existence of any sibling among the two other siblings. The structural integrity assurance scheme is not vulnerable to any of the inference attacks that Merkle hash technique is vulnerable to (listed in Section 3.2).

10 Performance

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

10.1 Complexity Analysis

Cost of Integrity Verifier Computation The pre-order and post-order numbers can be generated by a single traversal of the tree. The traversal complexity is thus $O(|V|)$. However if a "sorting of randoms" approach is used to compute randomized traversal numbers, the cost of computing IVs for a tree $T(V,E)$ is $O(|V|log(|V|))$. The use of "order-preserving encryption" for this purpose may result in nonlinear cost; however this is not clear from [1]. The cost in case of "addition of randoms" is $O(|V|)$.

The storage complexity of structural integrity verifiers is: $|\rho_x|+|\psi_T|+|\psi_x|$, which turns out to be $(2{*}k+|\mathcal{H}|+|\mathcal{H}|)$, where k and $|\mathcal{H}|$ are the number of bits used to represent a random number (RPON/RRON/RION) and the output of the hash respectively, thus a constant factor $O(1)$.

Cost of Distribution If the distribution strategy is to share the signed subtree $T_\delta(V_\delta, E_\delta)$ including its structure, the distributor has to send $|V_\delta|$ nodes and information about $|V_\delta|$-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 $|V_\delta|$ 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 $T_\delta(V_\delta, E_\delta)$, that is, $O(|V_\delta|)$. It accounts for one hashing for each node. The verification cost for structural integrity is also linear in terms of the size of the received subtree, that is, $O(|V_\delta|)$; the cost 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, the integrity verification cost comprises the cost of the integrity verification for the content and the IV, that is, $O(|V_\delta|)$, and the cost of reconstructing the subtree using the algorithm proposed in [9, 20], which is of linear order, that is, $O(|V_\delta|)$. 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.

10.2 Comparison with the Merkle hash technique

Cost of Integrity Verifier Computation The complexity of generating the Merkle signature for a tree $T(V,E)$ is $O(|V|)$, which is identical to that of the structural IVs. Our experiments show that structural integrity assurance 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 $|\psi_T|$, which is $|\mathcal{H}|$ bits, which is of $O(1)$ cost. The storage requirement of the structural IV scheme is higher in terms of a constant factor, which

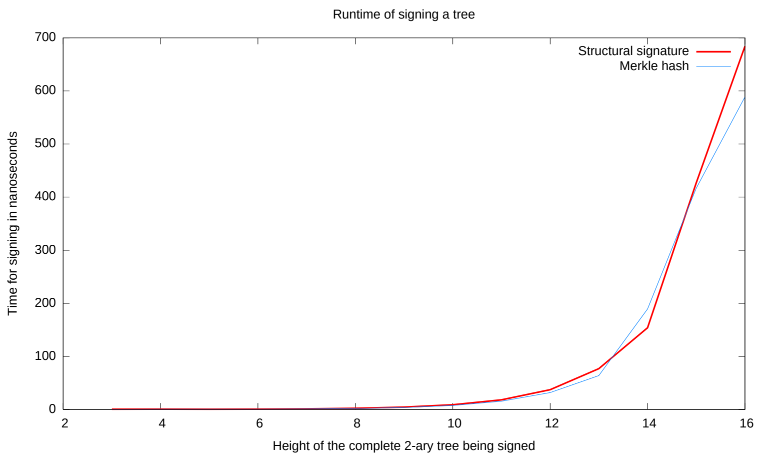


Figure 4: Time taken for generation of integrity verifiers for a complete 2-ary tree with respect to its height while using the Structural integrity verification scheme and the Merkle hash technique.

is $(2{*}k{+}|\mathcal{H}|)$. However this constant factor difference in the storage requirement helps in improving other costs.

Cost of Distribution Integrity verification of a subtree $T_\delta(V_\delta, E_\delta)$ in the Merkle hash technique involves computing the hashes of: (1) the nodes that are connected (adjacent) to a node in $T_\delta(V_\delta, E_\delta)$, but not part of $T_\delta(V_\delta, E_\delta)$, and (2) the ancestors of R, which is the root of $T_\delta(V_\delta, E_\delta)$. Such values are not necessary for integrity verification in case of structural IVs. Let μ refer to the set of the hashes as specified by (1) and (2). The distributor has to send all the nodes in the subtree V_δ, the hashes in μ, all structural and ordering among the nodes. Thus the communication cost on the side of the distributor is higher than what it would be if one were to send only T_δ: it is in the order of $2 * (|\mu| + |V_\delta|) - 1$. 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 IVs, the communication cost is either $2 * (V_\delta) - 1$ (when the nodes and edges are shared) or V_δ (when only the nodes are shared). The structural position of a node takes care of the structural order among nodes. Obviously, the communication cost for the structural IV is almost 50% of the cost incurred by Merkle hash technique.

Cost of Integrity Verification In the case of the Merkle hash technique, the user has to verify integrity by using a higher number of entities, that is, $2 * (|\mu| + |V_\delta|) - 1$, and a 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 when using the structural IVs, $O(|V|)$. Thus for integrity verification in comparison to the Merkle hash technique, the structural integrity assurance 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 integrity verifier cmputation). Our experimental results corroborate the fact that integrity verification at the user side using structural IVs is more efficient than using the Merkle hash technique.

10.3 Experimental Results

We implemented our structural integrity assurance scheme and Merkle hash technique. 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 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).

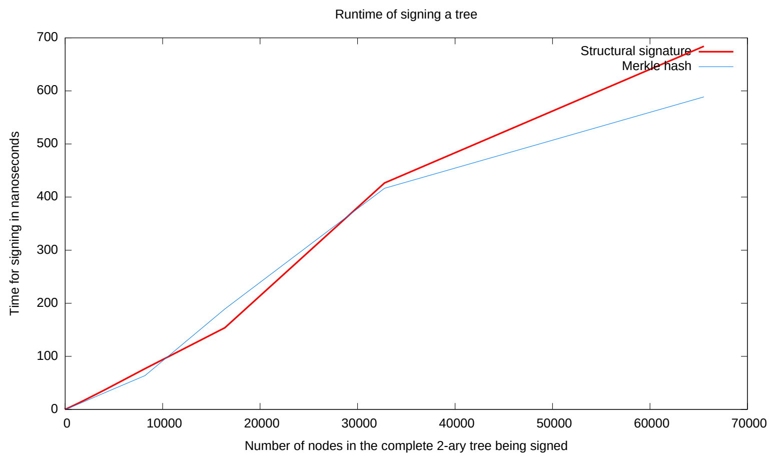


Figure 5: Time taken for generation of integrity verifiers for a complete 2-ary tree with respect to its number of nodes.

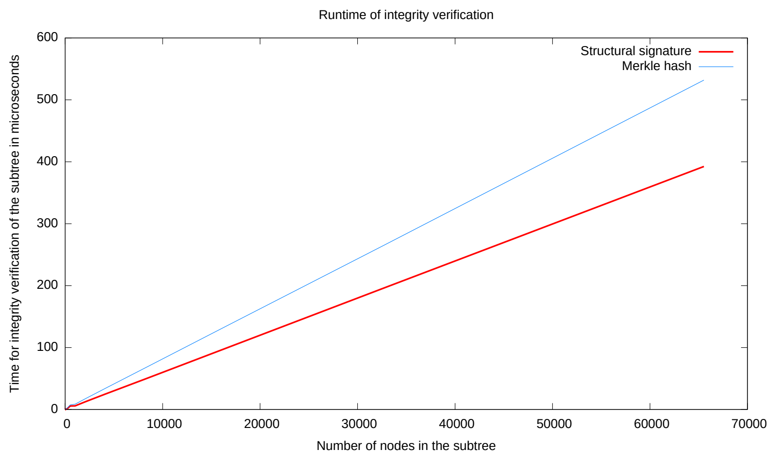


Figure 6: Time taken for integrity verification of a subtree with respect to its number of nodes.

Our experimental results show that the amount of time taken to generate the structural IVs for a tree is practically the same as the time required by the Merkle hash technique (Figures 4 and 5). The structural integrity assurance technique takes about 0.10 seconds more for a tree with 65535 nodes; it is quite negligible especially when IVs 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 integrity assurance scheme is less than the time required by the Merkle hash technique (Figures 6 and 7). 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.

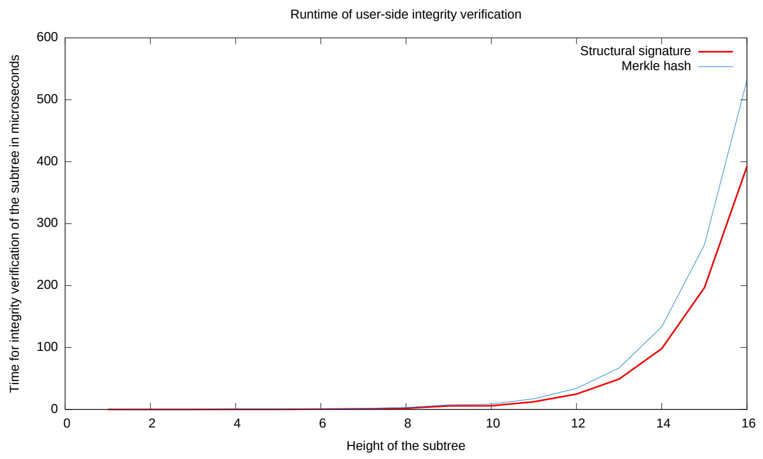


Figure 7: Time taken for integrity verification of a subtree with respect to its height.

11 Dynamic Updates

In the structural integrity assurance scheme, the IV of the tree is used for verifying the structural IVs of the nodes that are inserted or updated. The use of the IV of the old tree prevents leakage of the fact that the tree has been updated outside the subtree that a user has access to. Such a leakage would occur, if a new IV of the tree is computed such as in the Merkle hash technique.

11.1 Insertion

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 content, the structural position of the node, and the IV of the (original) tree are used to compute the IV of the node, which is signed by the owner. Insertion of a subtree is carried out as a sequence of insertions.

1. Downward (Leaf-level) updates: Let x be a leaf node and z be the new (leaf) node added as the child of x. A new RPON p_z is computed according to the following strategies.

   - (insertion sort) repetitively select a secure random η until $\eta < p_x$; η is assigned to p_z;
   - (order-preserving encryption) let z be the $(n+m)$-th node in the updated tree; then order-preserving encryption is applied to $(n+m)$ or a real number that is larger than the real number assigned to the $(n+m-1)$-th real number; such encrypted number is assigned to p_z;
   - (subtraction) draw a secure random number η; if $\eta < p_x$, then η is assigned to p_z; else $(p_x - \eta)$ is assigned to p_z. If such a technique is used, the randomized traversal numbers are treated as nonnegative numbers.

   RRON r_z should be larger than r_x and can be computed according to the following strategies:

   - (insertion sort) repetitively select a secure random η until $\eta > r_x$; η is assigned to p_z;
   - (order-preserving encryption) let z be the $(n+m)$-th node in the updated tree; then order-preserving encryption is applied to $(n+m)$ or a real number that is larger than the real number assigned to the $(n+m+1)$-th real number; such encrypted number is assigned to r_z;
   - (addition) draw a secure random number η; if $\eta > r_x$, then η is assigned to r_z; else $(r_x + \eta)$ is assigned to r_z.

2. Upward (Root-level) updates: Let x be the root node and z be the new root node added as the parent of x. RPON p_z (RRON r_z) is computed in the same way as RRON (RPON, respectively) is computed in the case of (a) downward updates.

3. Let node z be inserted as a sibling of x and y in the order x, z, and y: The process of insertion also computes the structural position p_z, r_z, ρ_z, and ψ_z. Randomized traversal numbers p_z and r_z for z are computed such that $p_x < p_z < p_y$, and $r_x < r_z < r_y$. RPON p_z is computed according to the following strategies.

   - repetitively select a secure random η until $p_x \ < \eta \ < p_y$ and η is assigned to r_z;
   - (order-preserving encryption) let z be the $(n + m)$th node in the updated tree; then order-preserving encryption is applied to $(n + m)$ or a real number that is larger than the real number assigned to the $(n + m + 1)$-th real number (for y) and smaller than the real number $(n + m - 1)$ or another real number; such encrypted number is assigned to p_z;
   - (addition) Let ω be a uniformly chosen random number from $(0, 1)$; $(p_x + \omega$*$(p_y$ - $p_x))$ is assigned to p_z.

   RRON r_z is computed in the same way as RPON p_z.

This technique is also used to handle insertions between a parent and a child. The nodes involved in the insertion of an edge from x to y (followed by deletion of z to y) require re-computation of the IV of y with respect to x.

However, after a certain number of insertions within an interval (i.e. within two nodes), the RPONs, RRONs or RIONs should be re-computed for the whole tree or subtree containing that interval. The reason is to avoid that the numbers too dense. The threshold number of insertions depends on the size of the randomized traversal number, such as 512-bit (for randoms generated by SHA2-512), and the size of the tree. In order to accommodate insertions and maintain appropriate randomness, 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.

11.2 Deletion

Upon the deletion of a node or a subtree, the IVs of the nodes 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 structural properties typical of trees to remain connected and to have $n - 1$ edges for n nodes. Update of a node/edge and a subtree can be carried out as insertion and deletion of a node/edge and a subtree. Insertion, deletion or update of a node/edge and a subtree of m nodes incurs $O(1)$ and $O(m)$ cost, respectively.

12 Weighted Trees

Weighted trees have a weight $w(x, y)$ associated with an edge $e(x, y)$ between two nodes x and y [7]. In this section, we show how to assure integrity of weighted trees without leakage by extending the structural integrity verification technique. Assume that every edge has an assigned weight; such a requirement can be easily fulfilled by assigning a weight of zero to edges that do not have any assigned weight. A naive solution is to split the edges into two with a node inserted in the edge. The content of the new node is the weight of the edge. However, such a solution almost doubles the size of the tree as well as doubles the cost of distribution and user-side integrity verification. We propose to split the weight of an edge $e(x, y)$ between the two nodes x and y such that a user can only learn about the weight if and only if it has access to the $e(x, y)$.

Weight Splitting

Algorithm to securely split weights in a tree $T(V, E)$:

1. Choose a secure random number η.
2. If $x \in V$ is the root of T, then $inw(x) \leftarrow \eta$.
3. If $x \in V$ is a leaf node, then $outw(x) \leftarrow \eta$.

4. For each edge $e(x, y) \in E$:
    (a) If $outw(x)$ has not been assigned any random, then $outw(x) \leftarrow \eta$.
    (b) $inw(y) \leftarrow \eta \oplus w(x, y)$.

Items $inw(x)$ and $outw(x)$ for each node x are treated as part of its contents. Computation of the structural integrity verifier of each node - ψ_x and of the tree - ψ_x includes these items according to the algorithm in Section 6.2.

Distribution

During sharing of a subtree T_δ with Bob, D sends $\langle \rho_x, inw(x), outw(x) \rangle$ alongwith the signed ψ_x and the node x itself.

Integrity verification

For each edge $e(x, y)$ received by the user Bob, he computes the following: $w'(x, y) \leftarrow outw(x) \oplus inw(y)$. By the definition of bit-wise XOR, if x and y are valid nodes received by Bob, $w'(x, y)$ is in fact same as the correct weight $w(x, y)$ between x and y. The fact Bob has received correct $outw(x)$ and $inw(y)$ is verified from verifying the ψ_x and ψ_y, respectively (Section 7). The cost of such an operation is $O(1)$.

Security

Let x and y be two nodes in T, with an edge between them in the tree with weight $w(x, y)$. The knowledge of $inw(x)$ $outw(x)$ does not reveal the value of $w(x, y)$ because Bob does not know $inw(y)$. A similar argument applies to the case in which the user has access only to y and not to x. This is due to Shannon's Theorem of Perfect Secrecy and Vernam cipher [21]. Having access to the edge $e(x, y)$ implies access to both x and y; thus Bob can compute the correct weight whenever it has access to $e(x, y)$.

13 Applications

The structural integrity verification scheme facilitates precise verification of integrity, i.e., pinpointing the nodes or the structural relationships/orderings that have been compromised. Precise detection of compromised nodes or structural information has many applications such as in (1) digital forensics, and (2) content and structural recovery.

Digital Forensics: Determining which nodes/units have been tampered with and whether their structural relationships have also been tampered is an important problem in digital forensics [16]. The technique proposed in [16] requires $O(d^3 log(n))$ hashes be computed for a set of n nodes, such that *at most* d compromised nodes can be deteted. By using our scheme, all compromised nodes can be detected and there is no upper bound on the number of compromises that can be detected. The IV of each node x is signed individually by a non-aggregate signature scheme and the signed value is regarded as the IV for that node. If the set of items is an ordered set, then any compromise of the ordering can also be detected from the structural IVs of nodes.

The cost incurred to provide such stronger security guarantee for an n-node ordered-data item is $O(n)$. Only $O(n)$ IVs are required, which is much less than $O(n^3 log(n))$, if the compromise of some or all of the n nodes need to be detected.

Content and Structural Recovery: The proposed scheme facilitates content-recovery by requesting re-transmission of only those nodes that are compromised. Such a capability helps data recovery with a minimum amount of data re-transmission. The technique used for digital forensics can be used in this case also to detect the data items that have been compromised. If the nodes in a structural ordering are not compromised, but the structural ordering is compromised, then from the IVs, the correct order can be easily computed. Such a capability helps automatic correction of data without any interaction with the distributor.

For example, suppose that two nodes x and y are received and verified to be uncompromised. Suppose that the ordering that y is left of x is the received order for these nodes in a subtree or a subsequence and suppose that it is incorrect. The user can then test if x is left of y. If the test succeeds, the correct order is recovered. If such test also fails, then there is no ordering (that is explicitly) imposed on these two nodes in the tree. For a graph, if a user receives an uncompromised IV for the order between x and y, the user can definitely determine the correct order between them by testing both alternatives. Thus our approach also works as an error correcting technique for recovering the correct orderings among nodes.

Secure publish/subscribe of XML: The structural integrity verification scheme can be used for secure dissemination of XML documents (and of tree-based data objects, in general) in a publish-subscribe model using structure-based routing [26, 25]. An alternative to the encoding scheme proposed in, structural integrity positions and verifiers can be computed for the XML document to be disseminated. Either of the RPON or RRON in the structural position can be used as the *routing parameter*, using which sub-documents (subtrees) can be routed to subscribers and subscribing routers. The integrity verification of a subdocument received by a user can be carried out as specified earlier in Section 7. Moreover, use of structural IVs prevents leakage of the lowest randomized post-order number in a subtree, which occurs in the previous scheme due to its process of encoding.

14 Related Work

Information leakage is an important problem in data sharing and analysis. Some of the contexts in which information leakage is currently being addressed are privacy-preserving databases [6, 33, 36, 38], automated trust negotiation [19], and error correction [13]. However, there is little work on information leakage through integrity verification and signature of data, especially trees.

Merkle [30] 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 4). 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 [17]) 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 [10]. Bertino *et al.* [3] proposed a technique based on the Merkle hash technique for selective dissemination of XML data in a third party distribution framework . Kocher [23] proposed to use Merkle hash trees for distribution to third parties. Goodrich and Tamassia [15] proposed authenticated dictionaries using skip lists and commutative hashing (one-way accumulators). Goodrich *et al.* [18] proposed techniques to authenticate graphs with specific path queries and geometric searching. Martel *et al.* [28] proposed a general model for authenticated data structures. For secure multicast, Perrig uses static data ordering over symmetric encryption [32].

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.* [35] 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.* [37]. 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 [35]. However they have not been used to define signatures for trees and graphs. Wang *et al.* [35] 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 is 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.

This paper is an extension of [27]. The extensions are as follows. We describe in detail how to compute randomized traversal numbers including the proof that randomized traversal numbers computed as addition of two randoms is a random. A distribution scheme of trees based on aggregate signatures is also proposed. Such a scheme requires sending only a constant ($O(1)$) number of integrity verifiers to a user in order to verify the integrity of the node. We discuss how to handle updates such as insertion and deletion of nodes and edges in our scheme. We extend the structural integrity assurance technique for integrity assurance of weighted trees.

15 Conclusions and Future Work

In this paper, we proposed the notion of structural integrity verifiers 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 integrity verifiers 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 our approach is as good as that of the Merkle hash technique both for the generation of integrity verifiers and integrity verification. Thus with the equivalent cost, our technique supports stronger security guarantees than the Merkle hash technique. Moreover, our approach reduces 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 hashes, structural integrity verifiers facilitate the precise detection of integrity violations, *i.e.* the compromised nodes.

As future work, we plan to extend the structural integrity verifier scheme to pervasive devices so that the integrity of a tree can be verified efficiently at the device-side with less energy consumption. Further, given the growing need of graph-structured data, we plan to explore leakage-free integrity assurance schemes for graphs.

Acknowledgement We thank Prof. Mikhail J. Atallah for his valuable comments on the paper. We thank Prof. Samuel S. Wagstaff, Jr. for the discussions and his insights on the security of addition of randoms. We thank the anonymous reviewers of this work for their valuable feedbacks. 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.

References

[1] 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.

[2] M.J. Atallah, YounSun Cho, and A. Kundu. Efficient data authentication in an environment of untrusted third-party distributors. *ICDE'08, Proceedings of the IEEE 24th International Conference on Data Engineering*, pages 696–704, April 2008.

[3] 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.

[4] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Advances is Cryptology – Eurocrypt'03, LNCS. Springer-Verlag*, 2003.

[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] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[8] Ivan Damgård. A design principle for hash functions. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 416–427, London, UK, 1990. Springer-Verlag.

[9] 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.

[10] 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.

[11] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11:2003, 2003.

[12] P. Devanbu, M. Gertz, Ch. U. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, pages 101–112, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

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

[14] 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.

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

[16] M. T. Goodrich, M. J. Atallah, and R. Tamassia. Indexing information for data forensics. In *ACNS*, pages 206–221, 2005.

[17] 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.

[18] 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.

[19] 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.

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

[21] J. Katz and Y. Lindell. *Introduction to Modern Cryptography: Principles and Protocols.* Chapman & Hall/CRC, 1 edition, 2007.

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

[23] 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.

[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 181–192, New York, NY, USA, 2007. ACM.

[25] 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.

[26] A. Kundu and E. Bertino. A new model for secure dissemination of xml content. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(3):292–301, May 2008.

[27] A. Kundu and E. Bertino. Structural signatures for tree data structures. *Proc. VLDB Endow.*, 1(1):138–150, 2008.

[28] 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.

[29] R. Merkle. *Secrecy, Authentication, and Public Key Systems.* Ph.D. Dissertation, 1979.

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

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

A Proof of Lemma 6.4

Proof. Let a and b be two n-bit cryptographically secure random numbers i.e. any bit in either of these numbers is 1 (or 0) with a probability of $\frac{1}{2}$. Let d be the result of the binary addition $a + b$, and c_{n+1} be the final carry of this addition. d_i denotes the addition of i-th bits of a and b with carry-in c_i, which is the carry-out of the addition of of a_{i-1}, b_{i-1} and c_{i-1} for $i > 0$. c_0 is 0.

We prove the lemma by claiming the following:

1. Bit-wise Exclusive-OR (XOR denoted by $\oplus$) of a random bit with another bit results in a random bit (Shannon's Theorem on perfect secrecy and Vernam cipher [21, 34]).
2. An addition of two binary numbers involves two XOR's: one is XOR of a_i and b_i (i-th bit); the other is is XOR of the result of the first XOR with the carry c_i from the addition of the $(i-1)$th bits.
3. The probability that the result of the XOR's d_i is 1 (or 0) is $\frac{1}{2}$, which implies that d_i is secure - it does not leak any information [21]; and the final carry of the binary addition $a + b$ is 1 or 0 with a uniform probability $\frac{1}{2}$. The probability that the carry is 1 is $\frac{2^n-1}{2^{n+1}}$ $(=\frac{1}{2}-\epsilon(n))$, which is same as $\frac{1}{2}$ because $\epsilon(n)=\frac{1}{2^{n+1}}$ is a negligible probability for a large n, which is the number of bits in a cryptographically secure random number.

(1) is straightforward.

(2) is proved as follows. *Binary addition*[5] *of a and b*: The addition of the i-th bits a_i and b_i are carried out as follows. c_{i+1} is the carry-out from the addition of a_i and b_i.

$$d_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = ((a_i \oplus b_i) \wedge c_i) \vee (a_i \wedge b_i)$$

.
Claim (3) is proved by induction.

Basis $i = 0$: The carry-in c_0 for the addition of least-significant bits (0-th) a_0 and b_0 is 0 (as there is no carry-forward). The probability that the sum $d_0 = 1$ is $\frac{1}{2}$, because for d_0 to be equal to 1, (a_0, b_0) should be (1,0) or (0,1), which has a probability $\frac{|\{(0,1),(1,0)\}|}{|\{(0,0),(0,1),(1,0),(1,1)\}|} = \frac{1}{2}$. Consequently the probability that $d_0 = 0$ is also $\frac{1}{2}$.

Carry c_1 is is equal to 1, only when $a_0 = b_0 = 1$; therefore the probability of $c_1 = 1$ is $\frac{1}{4}$.

$i = 1$: The probability that $d_1 = 1$ is $\frac{1}{2}$.

- $d_1 = 1$ when $c_1 = 0$, if $a_1 \oplus b_1 = 0$, that is, both a_1 and b_1 are either 1 or 0; The probability that $a_1 \oplus b_1 = 0$ is $\frac{1}{2}$ $(=\frac{|\{(1,1),(0,0)\}|}{|\{(0,0),(0,1),(1,0),(1,1)\}|})$. Thus the probability that $d_1 = 1$ when carry $c_1 = 1$ is (the probability that c_1 is 1) times (the probability that both a_1 and b_1 are either 1 or 0) $= \frac{1}{4} * \frac{1}{2} = \frac{1}{8}$.
- $d_1 = 1$ when $c_1 = 0$, if $a_1 \oplus b_1 = 1$, which has a probability of $\frac{1}{2}$. Thus the probability of $d_1 = 1$ when $c_1 = 0$ is $\frac{3}{4}*\frac{1}{2} = \frac{3}{8}$, since the probability that $c_1 = 0$ is $\frac{3}{4}$ $(=1-\frac{1}{4})$.

Thus the probability that $d_1 = 1$ (irrespective the value of carry in c_1) is $\frac{1}{8} + \frac{3}{8} = \frac{1}{2}$. The probability that $d_1 = 0$ is $\frac{1}{2}$.

The probability that $c_2 = 1$: $c_2 = 1$ if at least two of a_1, b_1 and c_1 are 1's. That is if either: (a) $c_1 = 1$ and at least one of a_1 and b_1 is 1: $\frac{1}{4} * \frac{3}{4} = \frac{3}{16}$; or (b) $c_1 = 0$ and both a_1 and b_1 are equal to 1: $\frac{3}{4} * \frac{1}{4} = \frac{3}{16}$. Thus the probability that $c_2 = 1$ is $\frac{3}{16} + \frac{3}{16} = \frac{3}{8} = \frac{2^2-1}{2^{2+1}}$.

[5]For a detailed discussion on the full adder of binary system, we refer the reader to *http://en.wikipedia.org/wiki/Half_adder* and the figure of full adder at *http://en.wikipedia.org/wiki/File:Full-adder.svg.*

Inductive Hypothesis: Suppose that the probability that $d_i = 1$ is $\frac{1}{2}$ and the probability that $c_{i+1} = 1$ is $\frac{2^{i+1}-1}{2^{i+2}}$. We have to prove that the probability that $d_{i+1} = 1$ is $\frac{1}{2}$ and the probability that $c_{i+2} = 1$ is $\frac{2^{i+2}-1}{2^{i+3}}$.

1. $d_{i+1} = 1$ if
   - $c_{i+1} = 1$ and $a_{i+1} \oplus\ b_{i+1} = 0$, that is a_{i+1} and b_{i+1} both are either 1 or 0, which has a probability of $\frac{1}{2}$. Thus the probability of $c_{i+1} = 1$ and $a_{i+1} \oplus\ b_{i+1} = 0$ is $\frac{2^{i+1}-1}{2^{i+2}} * \frac{1}{2} = \frac{2^{i+1}-1}{2^{i+3}}$.
   - $c_{i+1} = 0$ and $a_{i+1} \oplus\ b_{i+1} = 1$, which has a probability of $\frac{1}{2}$. Thus the probability of $c_{i+1} = 0$ and $a_{i+1} \oplus\ b_{i+1} = 1$ is (1 - $\frac{2^{i+1}-1}{2^{i+2}}$)$*\frac{1}{2} = (\frac{1}{2} - \frac{2^{i+1}-1}{2^{i+3}})$.

   Thus the probability of $d_{i+1} = 1$ is $\frac{2^{i+1}-1}{2^{i+3}} + (\frac{1}{2} - \frac{2^{i+1}-1}{2^{i+3}}) = \frac{1}{2}$.

2. $c_{i+2} = 1$ if
   - $c_{i+1} = 1$ and at least one of a_{i+1} and b_{i+1} is 1: $\frac{2^{i+1}-1}{2^{i+2}} * \frac{3}{4}$.
   - $c_{i+1} = 0$ and both a_{i+1} and b_{i+1} are 1's: $(1-\frac{2^{i+1}-1}{2^{i+2}}) * \frac{1}{4}$.

Thus the probability of $c_{i+2} = 1$ is
$\frac{2^{i+1}-1}{2^{i+2}} * \frac{3}{4} + (1-\frac{2^{i+1}-1}{2^{i+2}}) * \frac{1}{4} = \frac{3*(2^{i+1}-1)}{4*2^{i+2}} - \frac{2^{i+1}-1}{4*2^{i+2}} + \frac{1}{2^2} = \frac{2^{i+1}-1+2^{i+1}}{2^{i+3}}$
$= \frac{2^{i+2}-1}{2^{i+3}}$.

Thus the lemma is proven. □