Efficient Leakage-free Authentication of Trees, Graphs and Forests

Ashish Kundu*
IBM T J Watson Research Center
Hawthorne, New York, USA
akundu@us.ibm.com

Mikhail Atallah, Elisa Bertino
Department of Computer Science and CERIAS,
Purdue University.
West Lafayette, Indiana, USA
{mja,bertino}@cs.purdue.edu

ABSTRACT

Leakage-free authentication of trees and graphs have been studied in the literature. Such schemes have several practical applications especially in the cloud computing area. In this paper, we propose an authentication scheme that computes only one signature (optimal). Our scheme is not only super-efficient in the number of signatures it computes and in its runtime, but also is highly versatile – it can be applied not only to trees, but also to graphs and forests (disconnected trees and graphs). While achieving such efficiency and versatility, we must also mention that our scheme achieves the desired security – leakage-free authentication of data objects represented as trees, graphs and forests. This is achieved by another novel scheme that we have proposed in this paper – a secure naming scheme for nodes of such data structures. Such a scheme assigns "secure names" to nodes such that these secure names can be used to verify the order between the nodes efficiently without leaking information about other nodes. As far as we know, our scheme is the first such scheme in literature that is optimal in its efficiency, supports two important security concerns – authenticity and leakage-free (privacy-preserving/confidentiality), and is versatile in its applicability as it is to trees, graphs as well as forests. We have carried out complexity as well as experimental analysis of this scheme that corroborates its performance.

1. INTRODUCTION

In the emerging cloud computing paradigms, hosting and distribution of data is carried out by third party infrastructures and servers, which may not be trusted (e.g., Amazon EC2, Amazon Web Services AWS, "Database as a Service"[8]). In such third-party data distribution settings, an important requirement is to assure data authenticity. Data authenticity must be assured even when the data that a user can access is a subset of the signed data, as users maybe authorized to only access a subset of the data. In the cloud-computing paradigms, which are increasingly being employed in order to store and publish sensitive information belonging to individuals (such as healthcare) and enterprises, protection of privacy and confidentiality are as important as verifying authenticity of data [1]. Such leakages can be used to infer sensitive information that is not part of the received data, which in turn would lead to privacy and confidentiality breaches.

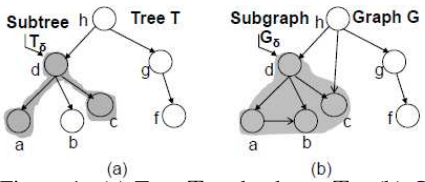

Figure 1: (a) Tree T and subtree T_δ. (b) Graph G and subgraph G_δ.

Leakage-free authentication of data in cloud have several applications such as in assuring healthcare, biological, and financial data (XML, graphs) while protecting privacy and confidentiality (for HIPAA compliance), and in authenticating XPath query results. Healthcare-specific examples, where leakages during authentication of trees and graphs lead to privacy breaches are given in [11] and [12], respectively.

The problem is how does *Alice* (the trusted data owner) sign the data (which is basically, a rooted directed tree or a directed graph) *once*, so that the authenticity of a portion of the data (subtree or a subgraph) received by a user can be verified without leaking any information about the remaining part of the data. An authentication scheme for a tree or a graph must allow one to verify the integrity of its content (*content integrity*) as well as its structure (*structural integrity*). Information that is not in a subtree/subgraph but is in its source graph is referred to as *extraneous information*.

Leakage-free property: As an example, consider the tree in Figure 1(a) and suppose that the user receives the subtree T_δ. The user should neither receive nor should be able to infer anything about the extraneous information: nodes b, f, g and h, and edges $e(d, b)$, $e(h, d)$, $e(h, g)$, and $e(g, f)$. The user should not be able to learn about the existence of b from the process that verifies the order between siblings a and c. In the case of graphs, another basic form of leakage is about immediate ancestors of a node. For example, assume that the user receives G_δ, subgraph of G in Figure 1(b). The user should not learn that node c has another immediate ancestor other than d (which is h), which is in G_δ. For nodes a, b and d, the user should not learn whether they have any other immediate ancestors in G.

Threat Model. Signer of a tree/graph/forest is trusted. The third-party server (cloud server) that processes the queries from clients is not authorized to sign query results on behalf of the trusted data owner, and the user may carry out carry out inference attacks on the data received.

*Part of this work was carried out by the author at CERIAS and Department of Computer Science, Purdue University

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$10.00.

Our Contributions. In this paper, we propose an authentication scheme for data objects represented as trees, graphs or even as forests (disconnected trees/graphs). The proposed scheme supports two important security properties: it (1) can be used to sign and verify authenticity of data objects, and (2) does not leak information during authentication of data. The proposed scheme (1) is optimal in the number of signatures it computes for any tree/graph/forest: one signature; (2) can be used to authenticate not only trees but also graphs and forests. We have also proposed a notion of secure names that can be used to verify order between two siblings of a tree/graph without leaking information as described earlier. Two schemes are proposed – the first one is in-efficient but provides an understanding on how an effcient scheme can be devised. The second naming scheme that we have proposed is both secure and efficient. Next, we propose signature schemes for trees and graphs based on these secure names and Condensed-RSA signature scheme. Both schemes are secure with respect to the two properties – authenticity and leakage-free properties. The CRSA schemes for trees and graphs compute $O(m)$ number of signatures computed for trees, where m is the number of nodes.

In the next part of the paper, we show that we can achieve better efficiency (than CRSA signatures) in leakage-free authentication of trees, graphs and forests: only one signature is computed and the number of expensive cryptographic operations such as modular exponentiations is reduced to as many as two. We are not aware of any other scheme in the literature that is more efficient and/or more secure (unforgeable and leakage-free). Later in the paper, we present experimental results related to computation of secure names, Condensed-RSA (CRSA) based signature schemes, and the single signature scheme (leakage-free redactable signature scheme – LFR scheme). Our schemes are highly scalable. The scheme for trees can accommodate trees with a high branching factors and very large number of nodes in the order of millions, which is corroborated by our experimental results. Branching factors of 100 and 300 (which result in trees with nodes as many as 1 million and 27 millions, respectively, with the height being 3) are handled by the proposed scheme quite efficiently. We also show how replay attacks can be prevented, how dynamic modifications to trees and graphs are handled.

Related Work. One of the most widely applied and extended techniques for authentication of tree-structured data is the Merkle hash technique [14] (MHT), which is binding (that is, integrity-preserving) but it has a major drawback in that it is not hiding (*i.e.*, it leaks information) [5]. It has been used in many scenarios such as: integrity assurance of XML [7], selective dissemination of XML data [2], integrity assurance of DAGs [13] and verifying completion of query results [16]. Merkle hash technique is integrity-preserving, but at the same time leaks [5]. Such a technique is not suitable for integrity assurance in high assurance environment and in privacy-preserving environments. The MHT and the techniques derived from it leak not only the Merkle hash of the nodes that the user does not have access to, but also the structural relationships, such as the parent-child relationships, and the sibling relationships as well as the structural ordering pertaining to nodes that the user does not have access to. Redactable and sanitizable[1] signature schemes have been developed in order to verify integrity of parts of a linear structured message that has been signed. A message in this context is a linear sequence of sub-messages. In this paper, our focus is on more complex structures: trees, graphs and forests. Moreover, all the schemes (except one mentioned later) leak structural information of the parts that are not part of the sub-message (such parts are called redacted or sanitized ones) being validated.

There is little work concerning the problem of leakage-free integrity verification of trees and graphs.

Kundu and Bertino recently proposed the scheme for trees [11] and graphs [12], which overcome such drawback of the MHT. However, these schemes use order between random numbers to verify order between siblings. However, the structural signature scheme for graphs is more expensive than the one proposed in this paper in terms of the number of integrity verifiers (per node and back-edges): it requires more than one depth first-traversals of graphs with cycles (proportional to the number of back-edges), where as the proposed scheme requires only one depth-first traversal of the graph (irrespective of the number of back-edges). Brzuska et al [4] proposed formal definitions of redactable structural signatures and described a scheme for trees. However, their scheme is an expensive one for trees: it computes quadratic number of signatures for the children of each non-leaf node (in number of siblings) in order to verify ordering among siblings. In contrast, in our scheme that is presented in this paper, we just compute *only one* signature for any tree, graph or forest. Moreover, our single signature scheme can be easily applied to not only trees, but also graphs and forests.

[1]Sanitizable signatures require a designated sanitizer, and redactable signatures allow for publicly sanitizable.

2. SECURE NAMES

In this section, we describe a notion of "secure names" that are assigned to the nodes in a tree. The purpose of secure names is to convey the order of siblings (which node is to the left of which other node) without leaking anything else (e.g., whether they are adjacent siblings, how many other siblings are between them, etc). For example, in Figure 1(a), a, b, and c are siblings such that $a \prec b \prec c$. Secure names η_a, η_b, and η_c are assigned to a, b, and c, respectively. Given η_a, and η_c, alongwith a and b, a user can establish the fact that $a \prec c$. But it cannot learn anything about b, or its existence (extraneous information).

The signing procedure traverses a tree $T(V, E)$ bottom-up, and assigns an N-bit secure name η_x to each node x in the tree, and then computes the signature σ_T of the tree using these secure names.

2.1 Preliminary Scheme (Scheme-1)

Our approach for generating secure names follows a bottom-up strategy. Let $v_1, \ldots, v_k$ be a list of siblings listed in left to right order. Let $lsb(s)$ denote the least significant bit of the bit-string s. The secure names of siblings v_i and v_{i+1} are computed such that the least significant bits of the hash of $\eta_{v_i} \| \eta_{v_{i+1}}$ *and* the hash of $\eta_{v_{i+1}} \| \eta_{v_i}$ are 1 and 0, respectively. We call this as the *ordering property* of secure names. This scheme is given in Figure 2.

Example: In the tree in Figure 1(a), N-bit secure names η_a, and η_b, are assigned to a, and b, respectively. η_a is a assigned as a random. η_b is computed such that $lsb(\mathcal{H}(\eta_a \parallel \eta_b))$ = 1 and $lsb(\mathcal{H}(\eta_b \parallel \eta_a))$ = 0. This process is repeated for each set of siblings.

2.1.1 Complexity

Scheme-1 takes exponential amount of time in terms of the number of children of a tree. The probability that a particular choice of r is found suitable for $\eta_{v_{\pi(i)}}$ is 4^{-i+1}, and the average number of r values generated for the selection of such an η_{v_i} is 4^{i-1}. The expected time to compute the secure names all k siblings is therefore: $\sum_{i=1}^{k} 4^{i-1} = (4^k - 1)/3$, and the average time to compute

`rNameGen`: Compute the secure names for siblings $v_1, v_2, \ldots, v_k$, children of node x, where $v_i \prec v_j, i < j$.

1. For the root node $root$ of T, assign a random to $\eta_{\hat{p}_{root}}$.
2. Repeat the following statements for each $x \in V$. Let $v_1, v_2, \ldots, v_k$ be the set of the children of x.
3. Generate a random permutation π of the integers $\{1, \ldots, k\}$.
4. Set $\eta_{v_{\pi(1)}}$ to be any random.
5. For $i = 2, \ldots, k$, compute $\eta_{v_{\pi(i)}}$ as follows.
   (a) Choose a random r.
   (b) For $j = 1, \ldots, i-1$, do the following:
      i. $\lambda_{\prec} \leftarrow \mathcal{H}(\eta_{v_{\pi(j)}} \parallel r)$.
      ii. $\lambda_{\succ} \leftarrow \mathcal{H}(r \parallel \eta_{v_{\pi(j)}})$.
      iii. If $v_{\pi(i)}$ is to the left (resp., right) of $v_{\pi(j)}$, then check whether $lsb(\lambda_{\prec})$ is 1 (resp., 0) and $lsb(\lambda_{\succ})$ is 0 (resp., 1).
      iv. If the answer is "yes" for all j, then $\eta_{v_{\pi(i)}} \leftarrow r$.
      v. Else go back to Sub-step 5(a).

`rNameVrfy`: Verify the order between two nodes $v_i \prec v_j$, using their secure names η_{v_i} and η_{v_j}.

1. $v_i \prec v_j \Leftrightarrow lsb(\mathcal{H}(\eta_{v_i} \parallel \eta_{v_j})) = 1 \wedge lsb(\mathcal{H}(\eta_{v_j} \parallel \eta_{v_i})) = 0$.

Figure 2: Algorithm to compute secure names for $T(V, E)$ (Scheme-1).

all secure names is $(n - \ell) * (4^k - 1)/3$. Moreover, although it is quite unlikely to happen, it is nevertheless possible that two non-sibling nodes receive the same secure name. In such a case, step 5(a) should be repeated.

Given its high cost, we need to find a more efficient solution – perhaps of linear complexity.

2.2 Efficient Scheme (Scheme-2)

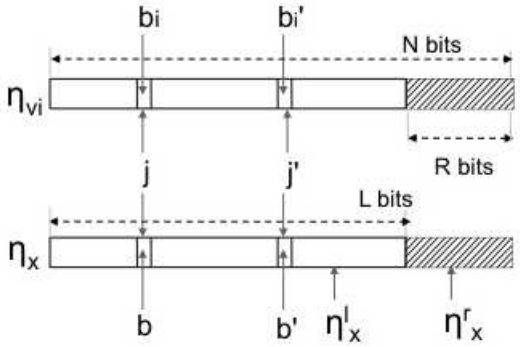

Figure 3: Secure names η_{V_i} and η_x of siblings V_i and x in the context of the efficient naming scheme.

The main drawback of the Scheme-1 is the fact that the worst-case time to compute an η_x, when x is the $(j+1)$'th leftmost child of its parent, is exponential in j (Step 3 in Section 2.1). This section describes an improved scheme that does not suffer from this drawback. As earlier, a non-leaf node in a tree has k number of children. In what follows, we describe a more efficient scheme for constructing a secure scheme.

The idea is, as before, to compute the η_x's (secure names) bottom-up and, within a set of siblings, in left-to-right order. The main difference is how a secure name η_x is computed. This scheme (Scheme-2) is given in Figure 4.

In this approach, we split the N-bit long secure name η_x of a node x into two disjoint parts: η_x^l and η_x^r of sizes L and R refer to the left and right parts of η_x, respectively (Figure 3). If x is the leftmost child (i.e., the first child) of its parent then η_x is selected randomly. If x is $(m+1)$'th leftmost child of its parent, then η_x depends on the secure names of its left siblings. Let w be a left sibling of x: $w \prec x$. Two bits (b_w and b'_w) in η_w^l and two bits (b_x and b'_x) in η_x^l are selected and their values are set such that $b_w \oplus b_x < b'_w \oplus b'_x$. b_w and b_x are the j'th leftmost bit in w and x, respectively, where j is computed using η_w^r and η_x^r in this order (because $w \prec x$). Similarly b'_w and b'_x are the (j')'th leftmost bit in w and x, respectively, where j' is computed using η_w^r and η_x^r in the reverse order. Alongwith N, $(L - 2 * k)$, and R are sufficiently large as security parameters.

Example: For $N = 512$, choose L=360 for $k \leq 100$, and R=152 in the context of current computational power. In the tree in Figure 1(a), secure names η_a, and η_b, are assigned to a, and b, respectively. η_a is an N-bit random and each bit of η_a^l is marked as *not-used*. η_b is computed as follows. η_b^r is an R-bit random and η_b^l is initialized to 0. Each bit of η_b^l is marked as *not-used*. j is computed as $(1 + \mathcal{H}(\eta_a^r \parallel \eta_b^r) \bmod L)$. Since j'th leftmost bits of η_a^l and η_b^l referred to as b_i and b, respectively, are marked as *not-used*, j' is computed as $(1 + \mathcal{H}(\eta_b^r \parallel \eta_a^r) \bmod L)$. If $(j \neq j')$ and the (j')'th leftmost bits of η_a^l and η_b^l referred to as b'_i and b', respectively, are marked as *not-used*, then proceed as follows. Assign either (0,0) or (1,1) to (b_i,b'_i) in η_a^l, and (0,1) or (1,0) to (b,b') in η_b^l. Such an assignment assures that $b_i \oplus b(=0) < b'_i \oplus b'(=1)$. The j'th and (j')'th bits of η_a and η_b are marked as *used*. η_c depends on both η_a and η_b. This process is repeated for each set of siblings.

2.2.1 Complexity

The above algorithm translates into a simple and constant-time test of which of two given siblings is to the left of the other. But we need to analyze the expected number of re-starts. Suppose that the size L of the left part (η_x^l) of a secure name is 500. The probability of a "collision" and re-start at Step (1) is the probability that $2k$ numbers drawn randomly from the 500 choices $[1, 500]$ are not all distinct, i.e., that at least 2 of them are equal. This is the classic birthday problem, and the probability of a re-start is (assuming $2k \leq 500$): $1 - \prod_{j=1}^{2k-1}\{(1 - (j/500))\} \approx 1 - e^{-(2k)(2k-1)/1000}$. For $2k = 50$ this probability is 0.91, hence the expected number of re-starts is $(1/(1 - 0.91) = 11$, which is much better than the preliminary scheme where the expected number of re-starts would have been proportional to 4^{25}. Scheme-2 incurs linear cost $O(n)$ in terms of the number of nodes in the tree.

3. TREES

In this section, we describe the signature, distribution and verification protocols for trees. Prior to computing the signatures, a dummy node is inserted by splitting an edge: if $e(x, y)$ is an edge in the original tree, add a node w such that $e(x, w)$ and $e(w, y)$ are the new edges in the modified tree. Secure name η_w of each inserted node w is a random. Such node w when given to a user only when the user has access to *both* x and y. The ordering between them is not needed to be verified by secure names.

3.1 Leakage-Free Signatures of Trees (`rSign`)

An integrity verifier (IV) of a node is the hash of the secure name of its parent, its secure name and its contents. In case of

`rNameGen`: Compute the secure name for x such that $v_1, v_2, \ldots, v_k, x$ are siblings, where $v_i \prec v_j \prec x, i < j$.

1. Choose a sufficiently large N. Choose L and R are such that (a) $N = L + R$, (b) $R \geq \log(L)$, and (c) $(L - 2 * k)$ and R are sufficiently large as security parameters. Let η_x^r and η_x^l refer to the right-part consisting of R and left-part consisting of L bits of the secure name η_x of x.
2. Assign random values to η_x^r, and zero values to η_x^l. Associate with each bit of η_x^l a status that is initially set to *not-used*.
3. Compute secure names of siblings in the left-to-right order of the siblings. Let $v_1, \ldots, v_k$ be the siblings to the left of x, where v_i is the ith leftmost one. (Each of the η_{v_i}'s of these k siblings of x have already been computed.)
4. For $i = 1$ to k do the following.
   (a) $\lambda_\prec \leftarrow \mathcal{H}(\eta_{v_i}^r \parallel \eta_x^r)$; $j \leftarrow 1 + (\lambda_\prec \mod L)$. Let b_i (resp., b) denote the jth leftmost bit of $\eta_{v_i}^l$ (resp., η_x^l). If the status of b is *not-used* then continue with the next step, else go back to step (2).
   (b) $\lambda_\succ \leftarrow \mathcal{H}(\eta_x^r \parallel \eta_{v_i}^r)$; $j' \leftarrow 1 + (\lambda_\succ \mod L)$. Let b_i' (resp., b') denote the (j')'th leftmost bit of $\eta_{v_i}^l$ (resp., η_x^l).
   (c) If $(j \neq j')$ and the status of b' is *not-used* then proceed to the next step, else go back to step (2).
   (d) Set b and b' such that $b_i \oplus b < b_i' \oplus b'$.
   (e) Change the status of b and b' from *not-used* to *used*.

`rNameVrfy`: Verify whether $y \prec z$, using their secure names η_y and η_z.

1. $j \leftarrow 1 + (\mathcal{H}(\eta_y^r \parallel \eta_z^r) \mod L)$
2. $j' \leftarrow 1 + (\mathcal{H}(\eta_z^r \parallel \eta_y^r) \mod L)$
3. Let b_y and b_z be the j'th, and b_y' and b_z' are the (j')'th bits in η_y and η_z, respectively.
4. Check the following: $y \prec z \Leftrightarrow b_y \oplus b_z < b_y' \oplus b_z'$.

Figure 4: Efficient algo to compute secure names for a tree (Scheme-2).

inserted nodes, no contents is used in IV. Using the IVs, we define a signature $\sigma_{T(V,E)}$ (also referred to as σ_T) for $T(V, E)$. In cases when "the received subtree (sent to the user) is the same as the original tree" is a sensitive information, the signature of a tree may be salted using a random value in order to protect this fact. The (salted) tree signature is publicly available or passed to the user alongwith the subtree that the user has access to. $\sigma_{T(V,E)}$ is an aggregate signature, computed over the IVs of its nodes. We define a signature for trees based on the condensed-RSA (CRSA) signatures [15]. A brief summary of CRSA is given in [12].

DEFINITION 3.1 (INTEGRITY VERIFIER). *Let x be a node in tree $T(V, E)$, and c_x be the content of node x. Its integrity verifier (IV) denoted by ξ_x, is defined as: $\xi_x \leftarrow \mathcal{H}(\eta_{\hat{p}_x} \| \eta_x \| c_x)$.*

In what follows, we define the signature of a tree based on Condensed-RSA signature scheme [15] and aggregate signatures [3].

`rSign`: Sign tree $T(V, E)$.

1. For each node $x \in V$, compute its secure name η_x, and compute its IV : $\xi_x \leftarrow \mathcal{H}(\eta_{\hat{p}_x} \| \eta_x \| c_x)$.
2. Assign a salt ω_T to T.
3. Compute the "signature of the tree" $\sigma_{T(V,E)}$ using CRSA as follows:
   (a) For each $x \in V$, $\sigma_x \leftarrow (\xi_x)^{\bar{d}} \ mod \ \bar{n}$.
   (b) Compute the signature σ_T by evaluating Eq. 1, where $\Omega_T \leftarrow {\omega_T}^{\bar{d}} \ mod \ \bar{n}$.

Figure 5: Algorithm to sign a tree using CRSA.

`rRedact`: Computed signature of the redacted subtree $T_\delta(V_\delta, E_\delta) \subseteq T(V, E)$.

1. $\sigma'_{T_\delta} \leftarrow \langle \sigma_{T_\delta}, \mathcal{VO}, \Theta_{T_\delta} \rangle$, computed as follows.
2. Θ_{T_δ} is the set of all secure names of the nodes and their respective parents in T_δ: $\Theta_{T_\delta} \leftarrow \{(\eta_x, \eta_{\hat{p}_x}) | x \in V_\delta\}$.
3. Compute the collective integrity verifier $\mathcal{VO}$ as follows.
4. CRSA: $\mathcal{VO} \leftarrow \omega_T \prod_{x \in (V - V_\delta)} \xi_x \ mod \ \bar{n}$; $\sigma_{T_\delta} \leftarrow \prod_{x \in V_\delta} \sigma_x \ mod \ \bar{n}$.

Figure 6: Algorithm to redact a subtree using CRSA.

DEFINITION 3.2 (SIGNATURE OF TREES USING CRSA). *Let $T(V, E)$ be a tree. Let $\mathcal{H}$ denote a random oracle. Let the RSA signature σ_x of each node x be defined as follows $\sigma_x \leftarrow \xi_x^{\bar{d}} \ mod \ \bar{n}$, where ξ_x is the IV of x. Let the salt be ω_T be a random, and let $\Omega_T \leftarrow {\omega_T}^{\bar{d}} \ mod \ \bar{n}$. The signature of T, denoted by σ_T, is defined as*

$$\sigma_T = \Omega_T \prod_{x \in V} \sigma_x \ mod \ \bar{n}. \quad (1)$$

3.2 Distribution (`rRedact`)

The distributor *D* sends the following items to Bob, who has access to $T_\delta(V_\delta, E_\delta)$, a subtree of tree $T(V, E)$: ($T_\delta(V_\delta, E_\delta)$,$\mathcal{VO}_{T_\delta}$,$\sigma_T$), where $\mathcal{VO}_{T_\delta(V_\delta, E_\delta)}$ (also referred to as $\mathcal{VO}_{T_\delta}$) is the verification object of T_δ, and σ_T the signature of the $T(V, E)$. The following steps show how to compute $\mathcal{VO}_{T_\delta}$. *D* computes two collective integrity verifiers σ_{T_δ} and Δ_{T_δ} as part of $\mathcal{VO}_{T_\delta}$ over the integrity verifiers of all the nodes that are not in the subtree and also includes the salt.

$\mathcal{VO}$ is used to verify the signature of the tree, and is used to detect if any node(s) has been dropped form T_δ in an unauthorized manner. σ_{T_δ} is used to verify the signature of all the nodes in the subtree in an aggregate manner, and is used to detect if any node(s) has been injected form T_δ in an unauthorized manner. η_x is the secure name of x.

3.3 Authentication (`rVrfy`)

Bob receives the subtree $T_\delta(V_\delta, E_\delta)$, the signature of the tree σ_T, and the verification object $\mathcal{VO}$. As part of the content authentication process, Bob computes the integrity verifiers of the nodes in V_δ and combines them with the received collective integrity verifier $\mathcal{VO}$. If the contents of the nodes are valid, the structural integrity

`rVrfy`: Verify authenticity of subtree $T_\delta(V_\delta, E_\delta)$.
Authentication of nodes:

1. For each node $y \in V_\delta$, compute $\xi_y \leftarrow \mathcal{H}(\eta_{\hat{p}_y} \| \eta_y \| c_y)$.
2. CRSA: Verify (a) and (b):
   (a) $(\sigma_{T_\delta})^{\bar{e}} \stackrel{?}{=} \prod_{y \in V_\delta} \xi_y \ (mod\ \bar{n})$, *and*,
   (b) $(\sigma_T)^{\bar{e}} \stackrel{?}{=} \mathcal{VO} \prod_{y \in V_\delta} \xi_y \ (mod\ \bar{n})$.
3. If (a) and (b) are valid, then the contents and secure names of T_δ are authenticated. Otherwise, if (b) is invalid and (a) is valid, then the received nodes are authenticated, but either some nodes have been dropped, or $\mathcal{VO}$ and/or σ_T have been tampered with.

Verification of edges and ordering among siblings:

1. Carry out a depth-first traversal on T_δ.
2. *Parent-child relationship*: Let x be the parent of y in T_δ; if ($\eta_x \neq \eta_{\hat{p}_y}$), then this relationship is incorrect.
3. *Order among siblings*: For ordered trees, in T_δ, let y and z are children of x, and let $y \prec z$.
   (a) For scheme-1 (Section 2.1): $y \prec z \Leftrightarrow (lsb(\mathcal{H}(\eta_y \parallel \eta_z)) = 1) \wedge (lsb(\mathcal{H}(\eta_z \parallel \eta_y)) = 0)$.
   (b) For scheme-2 (Section 2.2):
      i. $j \leftarrow 1 + (\mathcal{H}(\eta_y^r \parallel \eta_z^r) \bmod L)$
      ii. $j' \leftarrow 1 + (\mathcal{H}(\eta_z^r \parallel \eta_y^r) \bmod L)$
      iii. b_y and b_z are the j'th, and b'_y and b'_z are the (j')'th bits in η_y and η_z, respectively.
      iv. $y \prec z \Leftrightarrow b_y \oplus b_z < b'_y \oplus b'_z$.

Figure 7: Algorithm to verify a subtree using CRSA.

is verified with the help of secure names: the parent-child relationship, and the order among the siblings. Authentication of contents and structural positions of the subtree received includes (1) verification of integrity and, (2) verification of the source of the subtree. The integrity verification of structural relations in a tree involves traversing the tree and using the secure-name of two siblings of its parent or its sibling. The user can carry out verification of integrity of a n'-node subtree in $O(n')$-time. The verification procedure is given in Figure 7.

4. GRAPHS

Consider a simple graph G shown in Figure 1(b). It is a directed acyclic graph (DAG) with node c having two immediate ancestors – d and h Our solution for trees described earlier, does not work for graphs. In case of graphs, a node may have multiple incoming edges (i.e., multiple immediate ancestors such as c), whereas in case of trees, a node has only one parent (immediate ancestor) except for the root, which does not have any parent. Therefore, in the context of graphs, we cannot use the notion of integrity verifiers that is used for trees (Definition 3.1). The challenge in designing leakage-free signatures for graphs arises from the fact that the set $\alpha_\delta(x)$ of immediate ancestors of a node x in a subgraph G_δ is a (possibly empty) subset of the set $\alpha(x)$ of immediate ancestors of x in G. The question is how to verify the authenticity of $\alpha_\delta(x)$ without leaking any information about $(\alpha(x) \backslash \alpha_\delta(x))$: whether it is empty or non-empty, what is its size, etc? For example, c has

`rSign`: Sign a graph $G(V, E)$.

1. For each node $x \in V$,
   (a) For each node x, compute its secure name η_x.
   (b) For each node x, compute its integrity verifier $\xi_x \leftarrow \mathcal{H}(\eta_x \| c_x)$; For each edge $e(x, y)$, compute its integrity verifier $\xi_{(x,y)} \leftarrow \mathcal{H}(\eta_x \| \eta_y)$.
2. Assign a salt ω_G to G.
3. Compute the signature using CRSA σ_G:
   (a) For each $x \in V$, $\sigma_x \leftarrow (\xi_x)^{\bar{d}} \ mod\ \bar{n}$; For each edge $e(x, y) \in E$, $\sigma_{(x,y)} \leftarrow (\xi_{(x,y)})^{\bar{d}} \ mod\ \bar{n}$.
   (b) $\Omega_G \leftarrow {\omega_G}^{\bar{d}} \ mod\ \bar{n}$. Compute $\sigma_{G(V,E)} \leftarrow \Omega_G \prod_{x \in V} \sigma_x \prod_{e(x,y) \in E} \sigma_{(x,y)} \ mod\ \bar{n}$.

Figure 8: Algorithm to sign a graph using CRSA.

only d as its immediate ancestor G_δ, whereas it has d and h as the immediate ancestors in G. How to authenticate the fact that d in fact is a correct immediate ancestor of c in G_δ, without leaking any information about h. Figures 8, 9 and 10 describe schemes to sign, redact and verify graphs using CRSA, respectively. In what follows, ordering among siblings makes sense for graphs with cycles if and only if, the back-edge (or the edge which can be removed to break a given cycle among nodes) need to be of different semantics.

4.1 Leakage-Free Signatures for Graphs (`rSign`)

Our proposed scheme computes the integrity verifier of a node independent of the secure name of the parent, and integrity verifiers for edges. It computes secure names for nodes that have specific ordering with their siblings. If in a graph, the ordering between some siblings is not possible, then the secure names of such nodes are just randoms.

DEFINITION 4.1 (INTEGRITY VERIFIER: NODE). *Let x be a node in graph $G(V, E)$, and c_x be the content of node x. Its integrity verifier (IV) denoted by ξ_x, is defined as: $\xi_x \leftarrow \mathcal{H}(\eta_x \| c_x)$.*

DEFINITION 4.2 (INTEGRITY VERIFIER: EDGE). *Let $e(x, y)$ be an edge in graph $G(V, E)$. Its integrity verifier (IV) denoted by $\xi_{(x,y)}$, is defined as: $\xi_x \leftarrow \mathcal{H}(\eta_x \| \eta_y)$.*

Signature of a graph is then computed as the aggregate signature of the integrity verifiers of nodes and edges. Distribution is similar in the case of trees: if a user has access to a specific set of nodes and edges, signatures of the integrity verifiers of the edges and nodes as well as the secure names of the nodes are given to the user alongwith the nodes and edges. Also the aggregate signature of these IVs are given to the user alongwith the signature of the source graph and the verification object.

The signature scheme has a complexity of $O(|V|+|E|)$. It computes as many signatures as the number of nodes and edges in the graph (as in the case of trees) and another signature for the whole graph.

4.2 Distribution of Graphs (`rRedact`)

The distributor *D* sends the following items to Bob, who has access to $G_\delta(V_\delta, E_\delta)$, a subgraph of graph $G(V, E)$: (G_δ,$\mathcal{VO}_{G_\delta}$,σ_G),

rRedact: Computation of the redacted signature of $G_\delta(V_\delta, E_\delta)$:

1. Compute σ_{G_δ} and Δ_{G_δ} as follows.
2. CRSA: (a)$\sigma_{G_\delta} \leftarrow \prod_{y \in V_\delta} \sigma_y \ \prod_{e(x,y) \in E_\delta} \sigma_{(x,y)} \ mod \ \bar{n}$.
   (b) $\Delta_{G_\delta} \leftarrow \omega_G \prod_{y \in V - V_\delta} \xi_y \prod_{e(x,y) \in E - E_\delta} \xi_{(x,y)} \ mod \ \bar{n}$.
3. $\Theta_{G_\delta} \leftarrow \{\eta_x \mid x \in\}$; $\mathcal{VO}_{G_\delta} \leftarrow \langle \sigma_{G_\delta}, \Delta_{G_\delta}, \Theta_{G_\delta} \rangle$.

Figure 9: Algorithm to redact a subgraph using CRSA.

rVrfy: Verification of authenticity of subgraph $G_\delta(V_\delta, E_\delta)$
Authentication of contents:

1. For each node $x \in V_\delta$ in a subgraph $G_\delta(V_\delta, E_\delta)$, compute its integrity verifier: $\xi_x \leftarrow \mathcal{H}(\eta_x \| c_x)$.
2. For each edge $e(x, y)$, compute its integrity verifier: $\xi_{(x,y)} \leftarrow \mathcal{H}(\eta_x \| \eta_y)$.
3. CRSA: Compute (a) $((\sigma_{G_\delta})^{\bar{e}} \stackrel{?}{=} \prod_{x \in V_\delta} \xi_x \ (mod \ \bar{n}))$ *and*, (b) $((\sigma_G)^{\bar{e}} \stackrel{?}{=} \Delta_{G_\delta} \prod_{x \in V_\delta} \xi_x \ (mod \ \bar{n}))$.
4. If (a) and (b) are valid, then the contents and secure names of G_δ are authenticated. Otherwise, if (b) is invalid and (a) is valid, then the received nodes are authenticated, but either some nodes have been dropped, Δ_{G_δ} and/or σ_G have been tampered with. *Parent-child relationship* is verified during this process.

Verification of ordering among siblings:

1. Carry out a depth-first traversal on G_δ.
2. *Order among siblings*: In G_δ, let y and z are children of x, and let $y \prec z$.
   (a) For scheme-1 (Section 2.1): $y \prec z \Leftrightarrow (lsb(\mathcal{H}(\eta_y \| \eta_z)) = 1) \wedge (lsb(\mathcal{H}(\eta_z \| \eta_y)) = 0)$.
   (b) For scheme-2 (Section 2.2):
      i. $j \leftarrow 1 + (\mathcal{H}(\eta_y^r \| \eta_z^r) \bmod L)$
      ii. $j' \leftarrow 1 + (\mathcal{H}(\eta_z^r \| \eta_y^r) \bmod L)$
      iii. b_y and b_z are the j'th, and b'_y and b'_z are the (j')'th bits in η_y and η_z respectively.
      iv. $y \prec z \Leftrightarrow b_y \oplus b_z < b'_y \oplus b'_z$.

Figure 10: Algorithm to verify a subgraph using CRSA.

where $\mathcal{VO}_{G_\delta}$ (also referred to as $\mathcal{VO}_{G_\delta}$) is a verification object, and σ_G is the signature of G.

Δ_{G_δ} is used to verify the signature of the graph, and is used to detect if any node(s) has been dropped form G_δ in an unauthorized manner. σ_{G_δ} is used to verify the signature of all the nodes in the subgraph in an aggregate manner, and is used to detect if any node(s) has been injected form G_δ in an unauthorized manner. η_x is the secure name of x.

Example: *D* has to send G_δ in our example to Bob. σ_G is a CRSA-signature. *D* computes the Δ_{G_δ} as a modular multiplication of the salt ω_G, and the integrity verifiers of f, g, and h, because f, g, and h are not in G_δ. Now $\mathcal{VO}_{G_\delta}$ is the tuple consisting of σ_{G_δ}, Δ_{G_δ} and a set consisting of an element for each node in G_δ. *D* then sends the signature of the graph σ_G and $\mathcal{VO}_{G_\delta}$ alongwith G_δ, to the user.

4.3 Authentication (rVrfy)

Bob receives the subgraph $G_\delta(V_\delta, E_\delta)$, the secure name η_x of each node x, verification object $\mathcal{VO}_{G_\delta}$, and the signature of the graph σ_G. It verifies the authenticity of the contents; if they are authentic then the structural integrity is verified.

4.3.1 Authentication of the contents of subgraph G_δ

By contents, we mean the contents of each node x as well as η_x. In order to authenticate contents of $G_\delta(V_\delta, E_\delta)$, Bob first computes the integrity verifiers ξ_x for each node, and then combines them appropriately with Δ_{G_δ} in order to verify the signature σ_G. If the signature verifies, the edges and ordering among siblings are also verified. Authentication of contents of $G_\delta(V_\delta, E_\delta)$ has a complexity of $O(|V_\delta| + |E_\delta|)$.

Example: Bob computes the integrity verifiers of a, b, c and d in G_δ in our example. Consider CRSA signatures. Bob computes a modular multiplication of these integrity verifiers together with Δ_{G_δ} received as part of $\mathcal{VO}_{G_\delta}$. Then Bob applies the signature verification process of CRSA on the result of this multiplication and the received signature σ_G of the graph. If the verification turns out to be valid, the contents are authenticated.

5. SINGLE SIGNATURE SCHEME

In this section, we propose a construction of leakage-free redactable signaturesfor trees that is secure as well as highly efficient (computes *only one* signature). The LFR signature scheme is based on the notion of secure namesdeveloped earlier in the paper, and the redactable set signaturesdeveloped by Johnson, Molnar, Song and Wagner (JMSW) [9].

Review of JMSW Redactable Set Signatures: Johnson etal. [9] developed a redactable set signaturescheme based on RSA primitives and random oracle. The signature secure (EU-CMAover $\subset$ and $\cup$ operation). Since it is redactable, given thesignature of a set, and the elements that are to be removed from the set (thatresults in the subset), anyone who knows the public key, can efficiently computethe signature of the subset. The signature is history-independent, and thus canbe shown to be a leakage-free redactable signature for sets. Let the public keybe $\bar{e}$ and the RSA modulus be $\bar{n}$. Some constraints on RSA are that $\bar{n} = \bar{p}.\bar{q}$, where $\bar{p}$ and $\bar{q}$ are "safe primes".Given a set $S = \{s_1, s_2, \ldots, s_n\}$, its signature$\sigma_S$ is computed as follows: compute $H(S) = \prod_{1 \le i \le n} \mathcal{H}(S_i) \ mod \ \bar{n}$, and $I(S) = H_S^{-1} \ mod \ \phi(\bar{n})$, where$\phi(\bar{n}) = (\bar{p} - 1)(\bar{q} - 1)$. Signature σ_S is computed as$(\bar{e})^{I(S)}$. Verification proceeds as follows: Given a set S', and asignature σ, one computes the $H(S') = \prod_{1 \le i \le |S'|} \mathcal{H}(S'_i)$, where S'_i is the i'th element in S'; And then it is checkedif $(\sigma)^{H(S')}$ is equal to $\bar{e}$. If the equality holds, σ is a validsignature of the set S'. In order to compute the signature of a subset$S'' \subset S$, one redacts the hashes of the elements $S \setminus S''$ fromthe signature σ_S as follows: compute $H(S \setminus S'')$ and$\sigma_{S''}$ is computed as ${\sigma_S}^{H(S \setminus S'')}$.

5.1 LFR Signature: $r\Pi$

In this section, we present an LFR signature scheme $r\Pi$ for trees that can be easily extended to graphs/forests. Ordered graphs and forests can be signed using theabove scheme. The number of signatures computed is optimal: 1, and the number of hashingscarried out is $O(|V| + |E|)$. The signing, distribution, and verification schemes are given in Figures 11, 13, and 12, respectively.

5.2 Complexity

rSign: Sign tree $T(V, E)$. Let $p(x)$ be the parent of node x.

1. For each node w with m children, let x_i be the i'th child,$1 \leq i \leq m$, and $x_j \prec x_{j+1}$, $1 \leq j \leq m-1$.
2. Let η_x refer to the secure name assigned to node x.
3. Add a dummy node $dxy$to each edge $e(x, y)$, thereby splitting the edge to two edges$e(x, dxy)$ and $e(dxy, y)$. Assign a random η_{dxy} to each dummy node dxy. $V' \leftarrow V \cup \{dxy | e(x, y) \in E\}$.
4. For each node x in V', compute $\theta_x \leftarrow \mathcal{H}(\eta_{p(x)} \parallel \eta_x \parallel c_x)$.
5. $H(V) \leftarrow \prod_{x \in V} \theta_x \ mod \ \bar{n}$.
6. $\sigma_T \leftarrow (\bar{e})^{I(V)} \ mod \ \bar{n}$, where $I(V) \leftarrow H_V^{-1} \ mod \ \phi(\bar{n})$, $\phi(\bar{n}) = (\bar{p}-1)(\bar{q}-1)$.

Figure 11: LFR: Algorithm to sign a tree.

rRedact: Compute signature of subtree(s) $T_\delta(V_\delta, E_\delta) \subset T(V, E)$.

1. $\sigma_{T_\delta} \leftarrow (\sigma_T)^{H(V \setminus V_\delta)} \ mod \ \bar{n}$, where $H(V \setminus V_\delta) \leftarrow \prod_{x \in V \setminus V_\delta} \theta_x \ mod \ \bar{n}$.

Figure 12: LFR: Algorithm to redact a subtree.

Single signature scheme: Our single signature scheme takes a singletraversal on a tree/graph/forest, and incurs cost of $O(n + m)$, where n and m are number of nodes and edges,respectively. The number of signatures computed is 1, and the number oforder-preserving encryptions that are carried out is n/k, as that many groupsof siblings are there. The cost of computing a subtree/subgraph/sub-forest ofn' nodes and m' edges is $O(n - n' + m - m')$, because $n - n'$ nodes and $m - m'$edges have to be redacted from the signature. The cost of verifying theintegrity of a subtree/subgraph/sub-forest is $O(n' + m')$, but needs to verifyonly 1 signature. There is no decryption of the encrypted integers.*Brzuska et al's scheme*: For trees, it computes n signatures for thetree, and quadratic number of signatures for each group of siblings: $O(k^2)$(and there are n/k such groups of siblings), k is the arity of the tree.More precisely, it computes $O(n+nk)$ signatures, and incurs cost of $O(n+nk)$. The cost of computing a subtree ofn' nodes is $O(n'+n'k)$, because $n'k$ orderings and their signatures have tobe selected. The cost of verifying the integrity of a subtree/subgraph/sub-forestis $O(n' + n'k)$: these many signatures are verified.*Comparison*: In comparison to our scheme, signing by Brzuska et al's is$n + nk$ times more expensive. For computation of signature of redactedsignatures, our scheme requires one modular exponentiation, whereas the other scheme doesnot need any such operations; however, their scheme incurs $O(nk)$ moretraversal cost. Verification of the integrity of a subtree using Brzuska et al's is$(n' + n'k)$ times more expensive than our scheme.

6. PERFORMANCE RESULTS

We carried out experiments over the two schemes proposed in Sections 2.1 and 2.2.We implemented these two techniques in Java 1.6 and JCA 6.0 (Java Cryptography Architecture) APIs. The experiments were carried out on a IBM Thinkpad with the following specification: Linux (Ubuntu 8.10) on Intel Core 2 Duo CPU2.2GHz with 2.98GB RAM. SHA-512 is used as the hash function.

Verification (rVrfy): Verify $(\sigma, T_\delta(V_\delta, E_\delta))$; verifier receives$\theta_x$ for each node x in V_δ, and θ_{dxy} for each edge$e(x, y)$ in V_δ. *Verification of Contents*:

1. For each node x in V', compute $\theta_x \leftarrow \mathcal{H}(\eta_{p(x)} \parallel \eta_x \parallel c_x)$.
2. $H(V_\delta) \leftarrow \prod_{x \in V_\delta} \theta_x \ mod \ \bar{n}$, $\theta_x = \mathcal{H}(\eta_{p(x)} \parallel \eta_x \parallel c_x)$.
3. If $\sigma_{T_\delta}{}^{H(V_\delta)} \ mod \ \bar{n} = \bar{e}$, all nodes are authenticated.

Verification of Edges and Ordering:

1. Carry out a depth-first traversal on T_δ.
2. *Parent-child relationship*: Let x be the parent of y in T_δ; if $(\eta_x \neq \eta_{\hat{p}_y})$, then this relationship is incorrect.
3. *Order among siblings*: For ordered trees, in T_δ, let y and z are children of x, and let $y \prec z$.
   (a) For Scheme-1 (Section 2.1): $y \prec z \Leftrightarrow (lsb(\mathcal{H}(\eta_y \parallel \eta_z)) = 1) \wedge (lsb(\mathcal{H}(\eta_z \parallel \eta_y)) = 0)$.
   (b) For Scheme-2 (Section 2.2):
      i. $j \leftarrow 1 + (\mathcal{H}(\eta_y^r \parallel \eta_z^r) \bmod L)$
      ii. $j' \leftarrow 1 + (\mathcal{H}(\eta_z^r \parallel \eta_y^r) \bmod L)$
      iii. b_y and b_z are the j'th, and b'_y and b'_z are the (j')'th bits in η_y and η_z respectively.
      iv. Check: $y \prec z \Leftrightarrow b_y \oplus b_z < b'_y \oplus b'_z$.
4. If contents, the edges, and the orderings among all siblings are verified to be authentic, return 1, else return 0.

Figure 13: LFR: Algorithm to verify a subtree.

Computing Secure Names: Our performance results corroborate the theoretical analysis and show that the second technique outperforms the first technique both in the number of attempts and the time required to successfully assign a secure name to a nodes especially when the breadth of a tree is as high as in the order of hundreds. We considered trees of branching factor (the number of children a non-leaf node can have) ranging from 1 to 100. Considering the upper limit, a tree that has a breadth of 100 and a height as small as 3, can have as many as 1 million nodes. In the plots, the rank of a child among its siblings (other children of the same parent) is i, $0 \leq i \leq 99$. Figures 14 and 15refer to the performance results with respect to (1) and (2),respectively.Scheme 1 and 2 respectively refer to the preliminary technique(Sections 2.1) and the better technique(Section 2.2).

We have also carried out the experiments for branching factor 1 to 300, which requires modification of the size of the secure names (only in the case of the second technique) in order to accommodate the breadth of 300, which could not have been possible with the size of 512-bits for the secure name; 300 is a very large branching factor – a tree of a small height 3 and branching factor 300 has as many as 27 million nodes. The plots in Figures 16 and17 refer to the performance results with respect to the number of attempts to compute a secure name of a node and the time to compute such a secure name, respectively with respect to the position of a node among its siblings.

Performance of Signature Schemes

Both the CRSA-based and single-signature schemes that we have implemented use the efficient secure naming scheme in order to as-

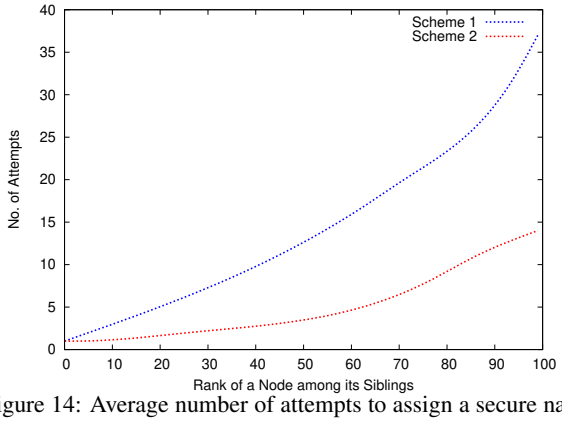

Figure 14: Average number of attempts to assign a secure name to a node; branching factor ≤ 100.

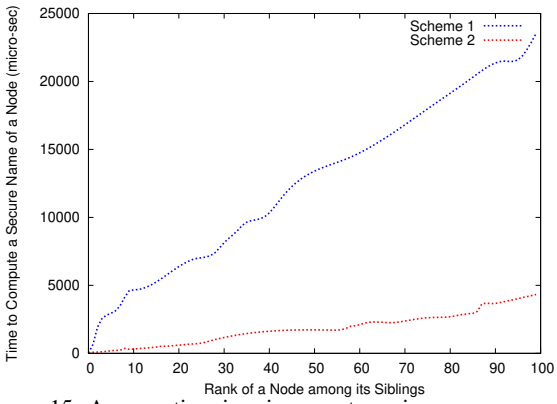

Figure 15: Average time in micro-sec to assign a secure name to a node; branching factor ≤ 100.

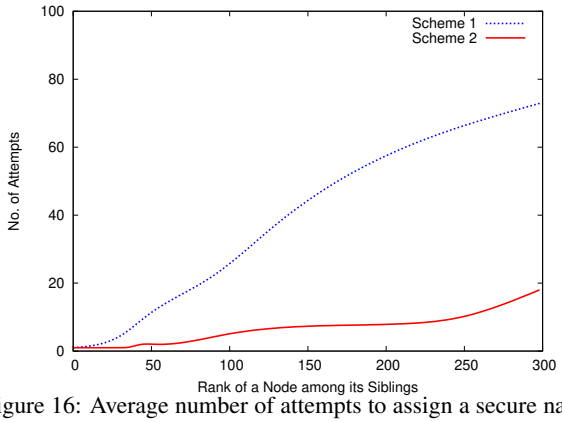

Figure 16: Average number of attempts to assign a secure name to a node; branching factor ≤ 300.

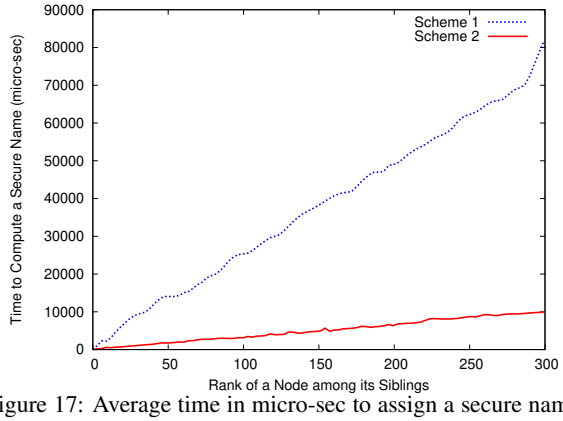

Figure 17: Average time in micro-sec to assign a secure name to a node; branching factor ≤ 300.

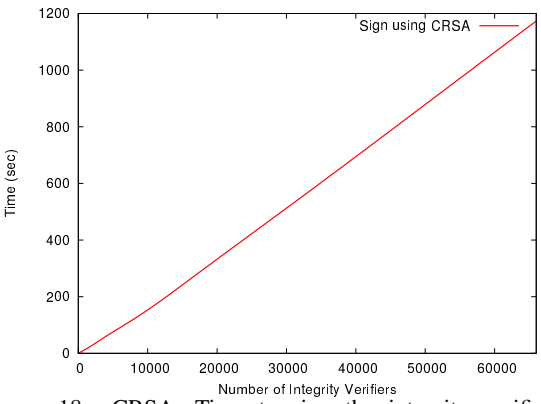

Figure 18: CRSA: Time to sign the integrity verifiers of a tree/graph/forest.

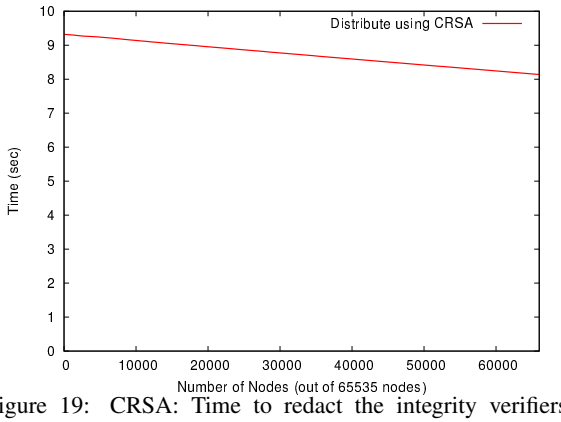

Figure 19: CRSA: Time to redact the integrity verifiers of a tree/graph/forest.

sign secure names to the siblings prior to signing the data. The time to compute signatures, distribute and verify them include the time to compute secure names. These performance results are applicable for forests as well, as the schemes for them are only different from the tree in the sense of representation of the edges. The experimental results for sign, verify and distribute hold for not only tree but also for graphs and forests. The X-axis for trees represents the number of integrity verifiers for nodes, whereas the graphs and forests, it represents the number of integrity verifiers for nodes *and* graphs. The Y-axis represents the time taken to compute the secure names *and* sign, distribute and verify.

CRSA-based Signature Scheme: We have carried out experiments using the efficient naming scheme. The plots for time to sign, distribute and verify trees are given in Figures 18, 19 and 20. The performances of these algorithms are similar to the performances of the algorithms in structural signatures. The dominant cost factor in signing, and verification is the modular exponentiation (modular multiplication for distribution) for CRSA. Verification of structural relationships is quite fast (less expensive than the cost of computing the secure names in the efficient scheme as shown in Figure 15), and do not affect the verification cost in any significant way; such

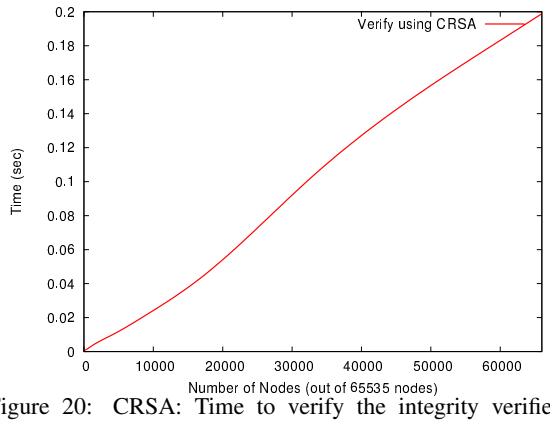

Figure 20: CRSA: Time to verify the integrity verifiers of a tree/graph/forest.

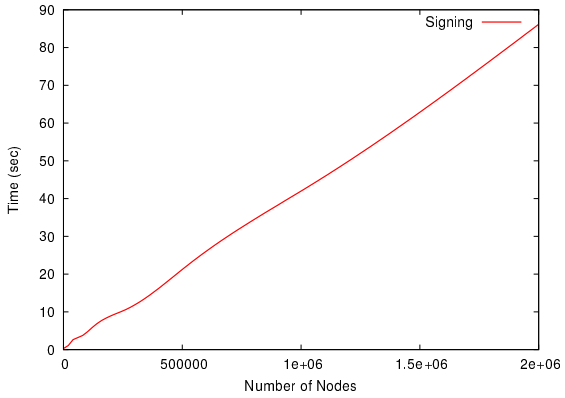

Figure 21: LFR: Signing a tree/graph/forest.

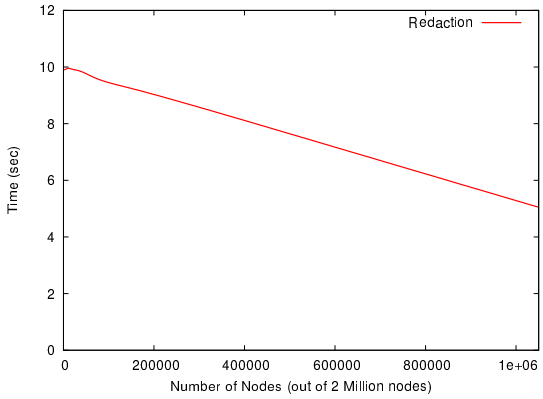

Figure 22: LFR: Computation of redacted signature of a subtree/subgraph/subforest.

cost is not included in the plot for verification.

LFR: Single Signature Scheme

Signing a tree using our scheme (that deals with sibling ordering as well) of $2 * 2^{20}$ (more than 2 Million) nodes requires about 70 seconds (Figure 21), which is in fact quite efficient. In contrast, for a tree with 65535 nodes, computing the RSA signatures for Brzuska et al's signature scheme [4] takes more than 1100 seconds (even without sibling ordering). It is significantly expensive than our signing scheme. The time to compute signatures of redacted

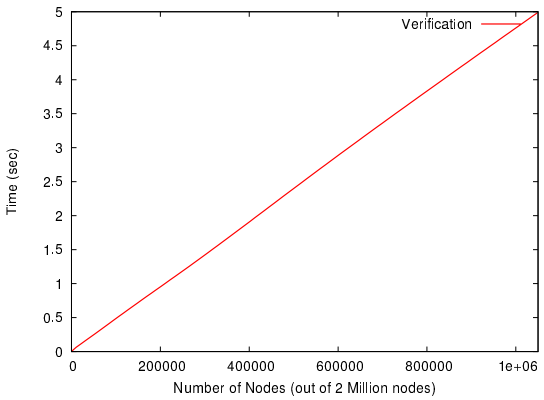

Figure 23: LFR: Verification of a subtree/subgraph/subforest.

subtrees decreases as the size of the redacted subtree increases (Figure 22), because the number of nodes to be redacted decreases with the increase in the size of the subtree. It takes about 5 seconds to compute the redacted signature of a 1-Million-node subtree (of a 2 Million node tree). It corroborates the fact that redaction of signatures is more efficient than re-computing them. Verification of the signature of a redacted subtree of 1 Million nodes (of a 2 Million node tree) requires about 5 seconds (Figure 23) , which is significantly less expensive than Brzuska et al's scheme: that has to verify 1 Million signatures just for parent-child relationships.

7. SECURITY ANALYSIS

We prove the security of the signature schemes by proving the leakage-free property of the secure names, unforgeability and leakage-free property of the signature schemes, for both trees and graphs. In what follows, $\mathcal{A}$ denotes a probabilistic polynomial time (PPT) adversary.

7.1 Secure Names

LEMMA 7.1 (LEAKAGE-FREE PROPERTY). *Under the random oracle hypothesis, secure names computed by Scheme-2 are leakage-free.*

PROOF. Sketch: Consider that an adversary $\mathcal{A}$ can determine with non-negligible probability whether a given set of secure names for a subset of siblings V_δ have been computed as part of the computation of secure names of V ($V_\delta \subset V$), or have been computed afresh. Consider the sets of nodes as output from $\mathcal{A}$: V_0 = {x,z} and V_1 = {x,y,z}, $x \prec y \prec z$. b is drawn uniformly and randomly from $\{0, 1\}$. The adversary then receives the challenge (V_0, Θ_0). Consider that $b = 1$, and $\mathcal{A}$ outputs $b' = 1$. $\mathcal{A}$ determines that there are one or more siblings in between x and z. It implies that the $j'th$ and (j')'th bit positions for (x,y) pair and/or for (y,z) pair are known to $\mathcal{A}$, which implies that $\mathcal{A}$ has been able to carry out second-preimage attack on $\mathcal{H}$. The bits of the secure names are assigned using $\oplus$ operation, and the probability of each bit being assigned 0 or 1 is $\frac{1}{2}$. R: the number of bits on the η_w^r, w is either of x,y, or z, is a security parameter (λ_2). Therefore, $\mathcal{H}$ is not a random oracle, which contradicts our assumption. Otherwise, $\mathcal{A}$ has carried out a brute-force attack by enumerating all possible secure names; however, the number of bits that are never used for any pair of siblings ($L - 2 * k$) is a large value - a security parameter (λ_1). It implies that $\mathcal{A}$ can carry out such brute force search over an exponential search space, a contradiction to the assumption that $\mathcal{A}$ is a probabilistic-polynomial adversary.

As earlier, let all secure names be in the interval $[1 : U]$. To prove that a secure name η_x reveals nothing about its rank i among its siblings, it suffices to prove that the process for secure-name assignment is such that the probability of a bit in η_x being either 0 or 1 is $\frac{1}{2}$, and it is true for all the bits in η_x. We give a proof by induction.
Basis: Case I: x is the left-most child of its parent: η_x is randomly chosen.
Case II: x is the second left-most child of its parent: Let v_1 be the left sibling of x. The R bits are randomly chosen. Two out of the remaining bits referred to as b and b' are chosen such that $(b_1 \oplus b) < (b'_1 \oplus b')$ (Step 2 of the scheme). However, b_1 and b'_1 are bits in the random v_1, i.e. the probability that the value of b_1 (or b'_1) is either 0 or 1 is $\frac{1}{2}$. Result of the XOR ($\oplus$) of a random number with another (possibly non-random) number is also a random number [10]. Thus b and b' are also random bits. The remaining bits of x are "not used" and randomly chosen. Thus the number $n(x))$ is a random.
Inductive step:
If v_k is the k'th left-most child of its parent and η_{v_k} is a random number, then η_x is also a random number where x is the $(k+1)$'st leftmost child of its parent. $r(v_k) + 2(k-1)$ number of bits in η_x are already "used". By Step 2 in the scheme, two bits at positions j and j' that are still unused in η_{v_k} are chosen. The $r(x)$ bits are randoms as well as the two bits at j and j' leftmost positions in η_x are also randoms. The remaining bits are "not used" and chosen randomly. Thus η_x is also a pseudo-random. □

7.2 Trees

LEMMA 7.2. *The signature scheme $r\Pi \equiv$ (*`rGen`*,* `rSign`*,* `rRedact`*,* `rVrfy`*) for trees using the CRSA scheme is existentially unforgeable under the adaptive chosen-message attack over the subset operation.*

PROOF. Sketch: Unforgeability of the signature is due to the unforgeability of CRSA. If the signature of a tree can be forged by an $\mathcal{A}$, then $\mathcal{A}$ has managed to solve the RSA problem or the Computational Diffie-Hellman problem, which however are assumed to be hard problems.

Given the correctness of the secure names (scheme-1 or scheme-2), the order between siblings can be verified. In case the order between siblings or edge relationship in a tree has been forged, then the hash function $\mathcal{H}$ is not a random oracle, which contradicts our assumption. □

LEMMA 7.3. *The signature scheme $r\Pi \equiv$ (*`rGen`*,* `rSign`*,* `rRedact`*,* `rVrfy`*) for trees using the CRSA scheme is leakage-free.*

PROOF. Sketch: $r\Pi$ is leakage-free if and only if the secure naming scheme is leakage-free, which is proven in Lemmas **??** and **??**. □

7.3 Graphs

LEMMA 7.4. *The signature scheme $r\Pi$ = (*`rGen`*,* `rSign`*,* `rRedact`*,* `rVrfy`*) for graphs using the CRSA scheme is existentially unforgeable under the adaptive chosen-message attack over the subset operation.*

Proof is similar to the proof of Lemma 7.2.

LEMMA 7.5. *The signature scheme $r\Pi$ = (*`rGen`*,* `rSign`*,* `rRedact`*,* `rVrfy`*) for graphs using the CRSA scheme is leakage-free.*

Proof is similar to the proof of Lemma 7.3.

7.4 Single Signature Scheme

The following lemmas state the security of the proposed construction $r\Pi$.

LEMMA 7.6. *Under the random oracle hypothesis, and the assumption that the RSA problem is hard, and that the secure naming scheme Scheme-2 is secure, $r\Pi$ is existentially unforgeable under chosen-message attack over subset (and union) operation over trees/graphs/forests.*

PROOF. Suppose that $r\Pi$ can be forged for a subtree T_δ. In one scenario, either a node x can be substituted by another node y in the subtree such that $\theta_x = \theta_y$. It implies that $\mathcal{H}$ has encountered a collision, contradicting the assumption that $\mathcal{H}$ is a random oracle. Similarly, forging a wrong parent-child relationship is not feasible under the random oracle hypothesis. If forging is carried out by forging the signature of a set (other than subset and union operations), then the JMSW signature scheme has been broken, which implies that the adversary has solved the RSA problem efficiently [9]. The order between two siblings cannot be forged under the random oracle hypothesis (because the $\mathcal{H}$ involves θ_x of each node x). □

LEMMA 7.7. *Under the random oracle hypothesis, and the assumption that the RSA problem is hard, then $r\Pi$ is leakage-free.*

PROOF. Suppose that $r\Pi$ is not leakage-free for a subtree T_δ of tree T. In other words, if T_δ and T are given to the $r\Pi$, the signature that one receives from T_δ leaks the fact whether it was computed from scratch or by redaction from the signature of T. If existence of a sibling in T, but not in T_δ, is leaked by the signature of T_δ, then the plaintext values of the position of a sibling among other siblings has been recovered, i.e., the secure naming scheme Scheme-2 has computed secure names that are not leakage-free, which however is not true. If a parent-child (edge) relationship between two nodes x and y in T_δ is leaked, then θ_{dxy} is not distinct, which is a contradiction. □

8. DISCUSSION

We would now describe how the schemes presented in this paper can be used for certain other scenarios.

Forests

Our single-signature scheme for graphs can be used to sign and authenticate forests *as-it-is*, i.e., a set of dis-connected trees/graphs in a leakage-free manner. Our scheme for graphs does not depend on connected-ness of graphs.

8.1 Encrypted Trees, Graphs and Forests

In cloud computing, often plaintext data is not delivered to the cloud servers. If the contents of the nodes are encrypted (but not the structure), out schemes (including the structural signature schemes) can be used directly to such scenarios. The only changes that are needed are (1) Share with the server the integrity verifiers, which are hashed values. (2) use a perfect one-way hash functions [6] to compute the hashes, which can then be hashed again to be converted to full-domain hashes. Perfect one-way hash functions do not leak contents of the message being hashed, whereas standard (such as SHA1 or SHA2) hash functions leak information.

8.2 Dynamic Trees, Graphs and Forests

In order to incrementally compute the signature of the updated tree, an insertion (resp., deletion) of a new node requires a new secure name, and leads to a modular multiplication (resp., division) in case of CRSA and an group addition (resp., subtraction) on the

elliptic curve followed by a bilinear operation. In an updated graph, the signature of immediate ancestors has also to be updated appropriately. Unlike in the MHT, in our schemes, the updates do not get propagated up in a tree. They do not affect the secure name of other siblings or nodes in Scheme-1; however, in Scheme-2, they affect the secure name of other siblings. The single signature scheme supports both redaction and union of two signatures, which is why, it also supports dynamic updates on trees, graphs and forests.

Computation of a new secure name in the j'th rank among its siblings does not affect any secure names of other siblings in case of Scheme-1. However, in the case of Scheme-2, it affects the secure names of all other siblings.

9. CONCLUSION AND FUTURE WORKS

In this paper, we propose an authentication scheme for data objects represented as trees, graphs or even as forests (disconnected trees/graphs). The proposed scheme supports two important security properties: it (1) can be used to sign and verify authenticity of data objects, and (2) does not leak information during authentication of data. The proposed scheme (1) is optimal in the number of signatures it computes for any tree/graph/forest: one signature; (2) can be used to authenticate not only trees but also graphs and forests. In order to achieve such level of security as well as efficiency, we developed the notion of secure names, and proposed an efficient scheme for computation of secure names. Experimental results corroborate our complexity analysis, and show that our schemes are highly scalable, which is essential in a cloud computing paradigm. We are in the process of implementing this scheme as part of a security stack of a cloud computing platform that is planned to host biological databases used by both academia and industry.

10. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, Technical Report No. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

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

[3] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of the 22nd Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2003.

[4] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)*, volume 6123 of *Lecture Notes in Computer Science*, pages 87–104. Springer Berlin / Heidelberg, June 22-25, 2010.

[5] A. Buldas and S. Laur. Knowledge-binding commitments with applications in time-stamping. In *Proceedings of the 10th International Conference on Practice and Theory in Public Key Cryptography (PKC)*, pages 150–165, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 131–140, New York, NY, USA, 1998. ACM.

[7] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of xml documents. *J. Comput. Secur.*, 12(6), 2004.

[8] H. Hacigumus, S. Mehrotra, and B. Iyer. Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 29–38, Washington, DC, USA, 2002. IEEE Computer Society.

[9] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Conference on the Cryptographers' Track (CT-RSA)*, pages 244–262, London, UK, 2002. Springer-Verlag.

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

[11] A. Kundu and E. Bertino. Structural signatures for tree data structures. *Proceedings of the VLDB Endowment*, 1(1):138–150, 2008.

[12] A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT 2010)*, volume 426 of *ACM International Conference Proceeding Series*. ACM, March 22-26, 2010.

[13] 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.

[14] R. C. Merkle. A certified digital signature. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[15] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *ACM Transactions of Storage*, 2(2):107–138, 2006.

[16] H. Pang and K.-L. Tan. Verifying completeness of relational query answers from online servers. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):1–50, 2008.