CERIAS Tech Report 2010-06

DATA IN THE CLOUD: AUTHENTICATION OF TREES, GRAPHS, AND FORESTS WITHOUT LEAKING

by Ashish Kundu, Mikhail Atallah, Elisa Bertino

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Data in the Cloud: Authentication of Trees, Graphs, and Forests Without Leaking

Ashish Kundu, Mikhail Atallah, Elisa Bertino
Department of Computer Science & CERIAS
Purdue University
{ashishk,mja,bertino}@cs.purdue.edu

ABSTRACT

In this paper, we address the problem of how to authenticate sub-trees (sub-graphs) without leakage of information. Previous schemes for tree (graph)-organized data, such as XML documents, authenticate information recorded in tree (graph) nodes, but leak structural information that the data receiver is not entitled to access.

This is often unacceptable, as the value of a tree (graph)-organized data is not only in the contents of the tree (graph) nodes but also in the tree (graph) structure (such as in healthcare and military data). A possible approach would be to store a pre-signed hash for each subset of the tree (graph). Such an approach is however not suitable even for moderate-size trees (graphs) because of the exponential number of such subsets. This paper proposes authentication schemes for trees and graphs (with or without cycles). The schemes are provably secure and efficient in that the number of signatures computed for trees is $O(1)$ and for graphs is $O(m)$, where m is the number of nodes. The schemes are highly scalable - they accommodate trees and graphs with high branching factors and extremely large numbers of nodes, such as in the order of millions. The efficiency 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 schemes quite efficiently. We also describe how our scheme for graphs can be used to authenticate forests without leaking.

1. INTRODUCTION

In the emerging cloud computing paradigms, the 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"[10]). In such third-party data distribution setting, an important requirement is to assure data authenticity. Authenticity is typically assured by message authentication codes and signatures computed by the owner of the data, which in third-party distribution setting, are different from the party distributing the data (referred to as distributor). 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. A crucial requirement is however to ensure that the techniques are used for signing the data do not result in data leakages. 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 (A healthcare-specific illustration is given in [12]). Some applications of leakage-free authentication of data are in cloud computing and database-as-a-service, in assuring healthcare data while protecting privacy and confidentiality (for HIPAA compliance), and in authenticating XPath query results.

Data published and distributed through third-party architectures is very often organized as trees or graphs. This is for example the case of data organized according to XML schemas. The question is thus how does *Alice* (the data owner) sign the data (which is basically, a tree or a 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. In what follows, "tree" and "graph" refer to a rooted directed tree and a directed graph (with or without cycles), respectively.

An authentication scheme for a tree or a graph must allow one to verify the integrity of its content as well as its structure. Integrity of the relationships represented by the edges and ordering, if any, between nodes is referred to as *structural integrity*, and the integrity of the contents of the nodes is referred to as *content integrity*. A leakage-free authentication scheme must make it possible for the receiver of a subtree/subgraph to verify its integrity with the following additional requirement: the receiver should neither receive nor should be able to infer any information about the content and presence (for that matter absence) of nodes and structural relationships that are not in its subtree/subgraph.

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 nodes b, f, g and h, and edges $e(d, b)$, $e(h, d)$, $e(h, g)$, and $e(g, f)$. Further, when there is an ordering between the nodes, the user should

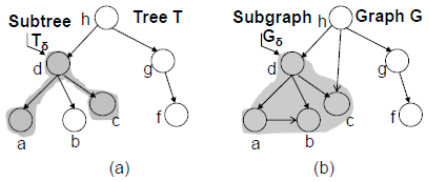

Figure 1: (a) Tree T and subtree T_δ. (b) Graph G and subgraph G_δ.

not be able to infer the ordering between any pair of nodes, when at least one such node is not in the subtree received by the user. This is the case, for example, of the node pairs: (a, b), (b, c) (d, g), and (c, f). Information that should not be leaked is referred to as *extraneous information*. 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 the graph.

One of the most widely applied and extended authentication techniques for tree-structured data is the Merkle hash technique [15] (MHT). The MHT is binding (that is, integrity-preserving) but it has a major drawback in that it is not hiding (*i.e.*, it leaks information) [5]. The MHT (and thus the techniques derived from it, e.g., [6][7][17][18]) leaks 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. The recent "structural signature scheme" for trees proposed by Kundu and Bertino [12] overcomes such drawback of the MHT. However, this scheme computes a signature for each node (hence very inefficient), and is also not suitable for applications to cyclic graphs because such graphs cannot be topologically ordered.

A straightforward scheme for graphs (including trees) is to sign each node as well as each edge. When a user has access to a subgraph (including a subtree), the signatures of the nodes and edges in the subgraph are also sent to the user. However, such a scheme has two major problems - (1) If the graph has an ordering between the siblings, it would not be possible for the user to verify such ordering, (2) the total number of signatures computed and stored in the worst case is $m + m^2$ $(O(m^2))$, where m is the number of nodes in the graph. For a graph of even medium size such as one thousand nodes, about a million signatures need to be stored and managed by the distributor, which is quite large. For a tree, it would have $2m - 1$ signatures. The question is can we support (1) while using less number of signatures? For trees, our scheme achieves m $(O(m))$ number of signatures for all cases while providing leakage-free verification of ordering between siblings and parent-child relationships. For graphs, our scheme achieves $2m$ $(O(()\mathrm{m}))$ number of signatures for all cases while providing leakage-free verification of ordering between siblings and parent-child relationships. For trees, our scheme requires about a 50% less number of signatures. For graphs, our scheme requires linear number of signatures against the quadratic number of signatures in the straightforward scheme.

Our Contributions. In this paper, we propose two provably secure and efficient leakage-free authentication schemes: one for directed rooted trees and another for directed graphs. To that end we define the concept of "leakage-free signatures" for trees and graphs. We defined two security notions for such signatures: (1) existential unforgeability under adaptive chosen-message attack, and (2) indistinguishability under the adaptive "chosen-signature" attack (leakage-free property). Both schemes are secure with respect to these two properties, and efficient in that the number of signatures computed for trees is $O(1)$ and for graphs is $O(n)$ where n is the number of nodes. Our schemes are highly scalable. The scheme for trees is able accommodate trees with a high branching factors and extremely large numbers 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, and how forests (e.g., a set of databases) can be authenticated without leaking using the proposed scheme for graphs.

2. SECURITY MODEL

Data Model. A tree $T(V, E)$ (or a directed graph $G(V, E)$) is a data object with V and E as the sets of vertices and edges, respectively. T_δ or G_δ refer to $T(V, E)$ or $G_\delta(V_\delta, E_\delta)$, respectively. A node x represents an atomic unit of the content denoted by c_x. Throughout this paper, unless otherwise stated, a tree is a rooted directed tree $T(V, E)$; a graph is a general directed graph. The notation $x \prec y$ denotes that x is a left sibling of y. Such an ordering may exist in a tree or in a directed acyclic graph (DAG). A subtree of $T(V, E)$ is denoted by $T_\delta(V_\delta, E_\delta)$,$(T_\delta(V_\delta, E_\delta) \subseteq T(V, E))$. Similarly, a subgraph of $G(V, E)$ is denoted by $G_\delta(V_\delta, E_\delta)$.

Other Common Notations: $\Upsilon(V, E)$ refers to either a tree $T(V, E)$ or a graph $G(V, E)$. The notation $\widetilde{T}(\widetilde{V}, E)$ (or simply, $\widetilde{T}$) refers to the structure of the tree $T(V, E)$, i.e., if the content c_x of each node x in T is removed, then the resultant tree structure is referred to by $\widetilde{T}(\widetilde{V}, E)$; edges in $\widetilde{T}$ remain as they are in T. Similarly, $\widetilde{G}(\widetilde{V}, E)$ is defined. We use m to denote the number of nodes in T. The parent of a node v is denoted by $\hat{p}_v$. The concatenation operator is denoted by $\|$, and the bitwise XOR operation is denoted by $\oplus$.

In what follows, whenever we say "generate a random" we mean a cryptographically secure random number. The notation $\mathcal{H}(v)$ denotes a cryptographically secure one-way hash of a value v.

Distribution Model. The *trusted owner* Alice signs a data object organized as a tree/graph Υ; the signature is referred to as $\Psi_{\Upsilon(V,E)}$. After signing Υ, *Alice* may delegate the job of publishing Υ or processing queries over Υ to a *third-party distributor* $\mathcal{D}$. The distributors are moderately untrusted in the sense that they do not have signature authority on behalf of *Alice*. The data object that a user receives as a result of its query or access request on a tree or a graph is

a subtree or a subgraph, respectively. $\mathcal{D}$ computes a verification object $\mathcal{VO}_{\Upsilon_\delta(V_\delta,E_\delta)}$ (or $\mathcal{VO}_{\Upsilon_\delta}$) for $\Upsilon_\delta(V_\delta, E_\delta)$, and sends both Υ_δ and $\mathcal{VO}_{\Upsilon_\delta}$ to the user *Bob*.

Threats. We assume a probabilistic polynomial adversary [8] throughout the paper. There are two types of attacks. (1) *Data tampering attack* by an adversary over the communication channel or at the distributor: the adversary tampers the content, the structural order and/or the type of structural relation (edge) between two or more nodes of a tree/subtree; (2) *Inference attack*: a user, who has access to Υ_δ, which is a portion of the data object Υ, attempts to infer information from the signature $\Psi_{\Upsilon(V,E)}$ and the verification object $\mathcal{VO}_{\Upsilon_\delta}$ that it receives from $\mathcal{D}$.

Security Definitions. As part of the authentication process, a user *Bob* can verify the authenticity, i.e., integrity and origin of $\Upsilon_\delta(V_\delta, E_\delta)$ using the signature Ψ_Υ and the verification object $\mathcal{VO}_{\Upsilon_\delta}$. Definition A.1 gives the formal definition of leakage-free signatures of trees and graphs. We define two experiments and two respective notions of security for signatures of trees/graphs: $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ and $\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)$. The notion of integrity of a subtree/subgraph captured in these definitions is: a subtree or subgraph is not compromised as long as neither the content of any node nor any structural relationship and ordering among siblings has been compromised. Leakage-free notion is about "not leaking any extraneous information in a tree/graph": Extraneous information in a tree/graph $\Upsilon(V, E)$ with respect to its subtree $\Upsilon_\delta(V_\delta, E_\delta)$ comprises of the nodes and edges that are in Υ but not in Υ_δ.The auxiliary information used by the MHT in order to authenticate a subtree also comprises of extraneous information [15].

In the appendix, we define two experiments and two security definitions of the signature scheme in an adverserial model based on a probabilistic polynomial-time adversary $\mathcal{A}$: (1) one for the *existentially unforgeable under adaptive chosen-message attack* property (Definition A.2), and (2) another for the *indistinguishable under the adaptive chosen-signature attack* (leakage-free) property (Definition A.3) of the signature and the verification object. The "adaptive chosen-signature attack" is analogous to "adaptive chosen-ciphertext attack". In this form of attack, a probabilistic polynomial-time adversary $\mathcal{A}$ learns one or more messages (trees or graphs) and the respective signatures as well as the verification objects (instead of ciphertexts).

3. TREES

In this section, we propose a leakage-free signature scheme Γ for trees. It relies on the 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 θ_a, θ_b, and θ_c are assigned to a, b, and c, respectively. Given θ_a, and θ_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 θ_x to each node x in the tree, and then computes the signature Ψ_T of the tree using these secure names. Using the secure names of the nodes in a tree, an "integrity verifier" for each node is computed, which in turn is used to define the signature of the tree $\Psi_{T(V,E)}$. A user that receives a subtree T_δ also receives the signature of the tree, and a verification object ($\mathcal{VO}$) in order authenticate the integrity and the origin of this subtree. $\mathcal{VO}$ is computed using the integrity verifiers of those nodes that are not in the subtree T_δ. The user verifies the signature of the tree using the $\mathcal{VO}$ and the received subtree together. The structural relationships between the nodes and the order between the siblings in T_δ are verified using their secure names.

In the following sections, we have shown how to compute the signature of a tree, and distribute, and how to authenticate a subtree. To ease the exposition, we first introduce a preliminary approach for naming the nodes, which is easier to understand and is secure but is not efficient - it has an exponential complexity. We then present the efficient solution that is both secure and efficient for trees with large branching factor.

3.1 Preliminary Scheme

3.1.1 Secure Names

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 $\theta_{v_i} \| \theta_{v_{i+1}}$ *and* the hash of $\theta_{v_{i+1}} \| \theta_{v_i}$ are 1 and 0, respectively. We call this as the *ordering property* of secure names.

Scheme-1: Algorithm to Compute Secure Names for $T(V, E)$:

1. For the root node *root* of T, assign a random to $\theta_{\hat{p}_{root}}$.
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 $\theta_{v_{\pi(1)}}$ to be any random.
5. For $i = 2, \ldots, k$, compute $\theta_{v_{\pi(i)}}$ as follows.
   (a) Choose a random r.
   (b) For $j = 1, \ldots, i-1$, do the following:
      i. $\lambda_\prec \leftarrow \mathcal{H}(\theta_{v_{\pi(j)}} \parallel r)$.
      ii. $\lambda_\succ \leftarrow \mathcal{H}(r \parallel \theta_{v_{\pi(j)}})$.
      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 $\theta_{v_{\pi(i)}} \leftarrow r$.
      v. Else go back to Sub-step 3(a).

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

3.1.2 Complexity

Scheme-1 takes $O(n)$ time, where n is the number of nodes in the tree. The probability that a particular choice of r is found suitable for $\theta_{v_{\pi(i)}}$ is 4^{-i+1}, and the average number of r values generated for the selection of such an θ_{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

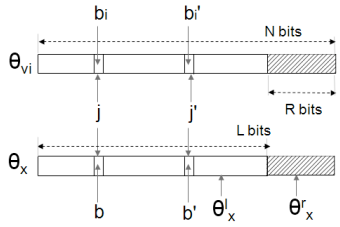

Figure 2: Secure names θ_{V_i} and θ_x of siblings V_i and x in the context of the efficient naming scheme.

compute all secure names is $(n-\ell)*(4^k-1)/3$. Note that, 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.

3.2 Efficient Scheme

The main drawback of the Scheme-1 is the fact that the worst-case time to compute an θ_x, when x is the $(j+1)$'th leftmost child of its parent, is exponential in j (Step 3 in Section 3.1.1). 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.

3.2.1 Secure Names

The idea is, as before, to compute the θ_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 θ_x is computed.

In this approach, we split the N-bit long secure name θ_x of a node x into two disjoint parts: θ^l_x and θ^r_x of sizes L and R refer to the left and right parts of θ_x, respectively (Figure 2). If x is the leftmost child (i.e., the first child) of its parent then θ_x is selected randomly. If x is $(m+1)$'th leftmost child of its parent, then θ_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 θ^l_w and two bits (b_x and b'_x) in θ^l_x 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 θ^r_w and θ^r_x 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 θ^r_w and θ^r_x in the reverse order. This is the *ordering property* of secure names computed in this fashion. Alongwith N, $(L-2*k)$, and R are sufficiently large as security parameters.

Algorithm to Compute Secure Names (Scheme-2):

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.
2. Assign random values to the R bits of θ^r_x, and zero values to the L bits of θ^l_x.
3. Associate with each bit of θ^l_x a status that is initially set to *not-used*.
4. Let $v_1, \ldots, v_k$ be the siblings to the left of x, where v_i is the ith leftmost one. Note that each of the θ_{v_i}'s of these k siblings of x have already been computed (due to the above mentioned left-to-right order of computing the new names).
5. For $i = 1$ to k do the following.
   (a) $\lambda_\prec \leftarrow \mathcal{H}(\theta^r_{v_i} \parallel \theta^r_x)$,
   (b) $\lambda_\succ \leftarrow \mathcal{H}(\theta^r_x \parallel \theta^r_{v_i})$.
   (c) $j \leftarrow 1 + (\lambda_\prec \mod L)$.
   (d) Let b_i (resp., b) denote the jth leftmost bit of $\theta^l_{v_i}$ (resp., θ^l_x).
   (e) If the status of b is *not-used* then continue with the next step, else go back to step (3).
   (f) $j' \leftarrow 1 + (\lambda_\succ \mod L)$.
   (g) Let b'_i (resp., b') denote the (j')'th leftmost bit of $\theta^l_{v_i}$ (resp., θ^l_x).
   (h) If $(j \neq j')$ and the status of b' is *not-used* then proceed to the next step, else go back to step (3).
   (i) Set b and b' such that $b_i \oplus b < b'_i \oplus b'$.
   (j) Change the status of b and b' from *not-used* to *used*.

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 θ_a, and θ_b, are assigned to a, and b, respectively. θ_a is an N-bit random and each bit of θ^l_a is marked as *not-used*. θ_b is computed as follows. θ^r_b is an R-bit random and θ^l_b is initialized to 0. Each bit of θ^l_b is marked as *not-used*. j is computed as $(1 + \mathcal{H}(\theta^r_a \parallel \theta^r_b) \mod L)$. Since j'th leftmost bits of θ^l_a and θ^l_b referred to as b_i and b, respectively, are marked as *not-used*, j' is computed as $(1 + \mathcal{H}(\theta^r_b \parallel \theta^r_a) \mod L)$. If $(j \neq j')$ and the (j')'th leftmost bits of θ^l_a and θ^l_b 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 θ^l_a, and (0,1) or (1,0) to (b, b') in θ^l_b. Such an assignment assures that $b_i \oplus b (=0) < b'_i \oplus b' (=1)$. The j'th and (j')'th bits of θ_a and θ_b are marked as *used*. θ_c depends on both θ_a and θ_b. This process is repeated for each set of siblings.

3.2.2 Complexity

Note that 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 (θ^l_x) 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.

3.3 Leakage-free Signatures of Trees (`Sign`)

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 θ_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.

3.3.1 Integrity Verifiers

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 inserted nodes, no contents is used in IV. Using the IVs, we define a signature $\Psi_{T(V,E)}$ (also referred to as Ψ_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. $\Psi_{T(V,E)}$ is an aggregate signature, computed over the IVs of its nodes. We define two types of signatures for trees: one based on the condensed-RSA signatures [16] and the other based on bilinear maps [4].

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: $\xi_x \leftarrow \mathcal{H}(\theta_{\hat{p}_x} \| \theta_x \| c_x)$.*

3.3.2 Signatures using Condensed-RSA

In this section, we define the signature of a tree based on Condensed-RSA (CRSA, in short) signature scheme [16]. First, a review of this scheme.

Review of Condensed-RSA Signatures

The public key and private keys of an RSA scheme are $(\bar{n}, \bar{e})$, and $\bar{d}$, respectively [16], where $\bar{n}$ is the product of two large random primes $\bar{p}$ and $\bar{q}$. $\bar{e}, \bar{d} \in Z_n^*$, such that $\bar{e}\bar{d} \equiv 1 \ mod \ (\bar{p}-1)(\bar{q}-1)$. A message M is signed using RSA signature as follows: Signature of M $\Psi_M \leftarrow \mathcal{H}(M)^{\bar{d}} \ mod \ \bar{n}$. A RSA signature Ψ_M is verified as follows: ${\Psi_M}^{\bar{e}} \stackrel{?}{=} \mathcal{H}(M) \ mod \ \bar{n}$. Let $M_1, M_2, \ldots, M_m$ refer to m messages. The condensed signature of these m messages is computed as follows: first compute the RSA signature Ψ_i for each message i, $1 \leq i \leq m$; then compute the product of these signatures $\Psi_{1,m} \leftarrow \prod_{i=1}^{m} \Psi_i \ mod \ \bar{n}$. The Condensed-RSA signature $\Psi_{1,m}$ is verified as follows: $(\Psi_{1,m})^{\bar{e}} \stackrel{?}{=} \prod_{i=1}^{m} \mathcal{H}(M_i) \ (mod \ \bar{n})$. The definitions are simplified versions of RSA with no padding. Condensed-RSA is unforgeable against an adaptive chosen-message attack under the assumption that RSA is a one-way function [16][2].

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 Ψ_x of each node x be defined as follows $\Psi_x \leftarrow \xi_x^{\bar{d}} \ mod \ \bar{n}$, 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 Ψ_T, is defined as*

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

3.3.3 Signatures using Aggregate Signatures

In this section, we define the structural signature of a tree based on aggregate signatures (BGLS, in short)[4]. First, a review of the same.

Review of Aggregate Signatures

Let $\mathbb{G}_1 = \langle \mathtt{P} \rangle$ be an additively-written group of prime order p, and let $\mathbb{G}_2$ be a multiplicatively written group of the same prime order p. A mapping $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map if (i) $\mathbb{E}(aX, bY) = \mathbb{E}(X,Y)^{ab}$ for all $X, Y \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p^*$; and (ii) $\mathbb{G}_2 = \langle \mathbb{E}(\mathtt{P}, \mathtt{P}) \rangle$. The mapping $\mathbb{E}$ is efficiently computable, but given only $\mathtt{P}$, $a\mathtt{P}$, and X (but not a) it is computationally infeasible to compute aX (i.e., the Computational Diffie-Hellman problem is difficult in $\mathbb{G}_1$). 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 $\underline{\mathtt{sk}} \in Z_p^*$, $\mathtt{Q} = \underline{\mathtt{sk}}\mathtt{P} \in \mathbb{G}_1$ is public (i.e., $\underline{\mathtt{pk}}$), and the signature for a message m is $\underline{\mathtt{sk}}M$ with $M = \mathcal{H}(m) \in \mathbb{G}_1$, where $\mathcal{H}$ is a cryptographic one-way hash function. In the aggregate signatures, given the public $\mathtt{P}$ and $\mathtt{Q}$, and given k message-signature pairs $M_i, \Psi_i = \underline{\mathtt{sk}}M_i$, $1 \leq i \leq k$, the signature is verified by checking that the following equality holds: $\mathbb{E}(\mathtt{Q}, \sum_{\mathtt{i}=1}^{\mathtt{k}} \mathtt{M_i}) = \mathbb{E}(\mathtt{P}, \sum_{\mathtt{i}=1}^{\mathtt{k}} \Psi_\mathtt{i})$.

DEFINITION 3.3 (SIGNATURE OF TREES USING BGLS). *Let $T(V,E)$ be a tree. Let $\mathcal{H}$ denote a random oracle. Let the salt be $\omega_T \leftarrow \mathcal{H}$(a distinct random). The signature Ψ_x of each node x is defined as $\Psi_x \leftarrow \underline{\mathtt{sk}}\xi_x$. The signature of T, denoted by Ψ_T, is defined as*

$$\Psi_T \leftarrow \mathbb{E}(\mathtt{P}, \underline{\mathtt{sk}}(\omega_\mathtt{T} + \sum_{\mathtt{x} \in \mathtt{V}} \xi_\mathtt{x})). \quad (2)$$

Algorithm for Signing a Tree $T(V,E)$:

1. For each node $x \in V$, compute its secure name θ_x, and compute its IV : $\xi_x \leftarrow \mathcal{H}(\theta_{\hat{p}_x} \| \theta_x \| c_x)$.
2. Assign a salt ω_T to T.
3. If CRSA is used, compute the "signature of the tree" $\Psi_{T(V,E)}$ as follows:
   (a) For each $x \in V$, $\Psi_x \leftarrow (\xi_x)^{\bar{d}} \ mod \ \bar{n}$.
   (b) Compute the signature Ψ_T by evaluating Eq. 1, where $\Omega_T \leftarrow {\omega_T}^{\bar{d}} \ mod \ \bar{n}$.
4. If BGLS is used, compute the signature of each node, and compute "signature of the tree" Ψ_T by evaluating Eq. 2.

3.4 Distribution (Dist)

The distributor $\mathcal{D}$ sends the following items to Bob, who has access to $T_\delta(V_\delta, E_\delta)$, a subtree of tree $T(V,E)$: $(T_\delta(V_\delta, E_\delta), \mathcal{VO}_{T_\delta}, \Psi_T)$, 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 Ψ_T the signature of the $T(V,E)$. The following steps show how to compute $\mathcal{VO}_{T_\delta}$. $\mathcal{D}$ computes two collective integrity verifiers Ψ_{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.

1. $\mathcal{VO}_{T_\delta} \leftarrow \langle \Psi_{T_\delta}, \Delta_{T_\delta}, \Theta_{T_\delta} \rangle$, computed as follows.
2. Θ_{T_δ} is the set of all secure names of the nodes and their respective parents in T_δ: $\Theta_{T_\delta} \leftarrow \{(\theta_x, \theta_{\hat{p}_x}) | \ x \in V_\delta\}$.
3. Compute the collective integrity verifier Δ_{T_δ} as follows.
   (a) CRSA: $\Delta_{T_\delta} \leftarrow \omega_T \prod_{x \in (V - V_\delta)} \xi_x \ mod \ \bar{n}$;
   $\Psi_{T_\delta} \leftarrow \prod_{x \in V_\delta} \Psi_x \ mod \ \bar{n}$.
   (b) BGLS: $\Delta_{T_\delta} \leftarrow \omega_T + \sum_{x \in (V - V_\delta)} \xi_x$;
   $\Psi_{T_\delta} \leftarrow \mathbb{E}(\mathtt{P}, \sum_{\mathtt{x} \in \mathtt{V}_\delta} \Psi_\mathtt{x})$.

Δ_{G_δ} 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. Ψ_{T_δ} is used to verify the signature of al 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. θ_x is the secure name of x.

3.5 Authentication (Vrfy)

Bob receives the subtree $T_\delta(V_\delta, E_\delta)$, the signature of the tree Ψ_T, and the verification object $\mathcal{VO}_{T_\delta}$. 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 Δ_{T_δ}. If the contents of the nodes are valid, the structural integrity is verified with the help of secure names: the parent-child relationship, and the order among the siblings.

3.5.1 Authentication of Contents

Authentication of contents and structural positions of the subtree received includes (1) verification of integrity and, (2) verification of the source of the subtree.

1. For each node y in the set of received nodes V_δ, compute $\xi_y \leftarrow \mathcal{H}(\theta_{\hat{p}_y} \| \theta_y \| c_y)$.
2. CRSA: Verify (a) and (b):
   (a) $(\Psi_{T_\delta})^{\bar{e}} \stackrel{?}{=} \prod_{y \in V_\delta} \xi_y \ (mod\ \bar{n})$ *and*,
   (b) $(\Psi_T)^{\bar{e}} \stackrel{?}{=} \Delta_{T_\delta} \prod_{y \in V_\delta} \xi_y \ (mod\ \bar{n})$.
3. BGLS: Verify (a) and (b):
   (a) $(\Psi_{T_\delta}) \stackrel{?}{=} \mathbb{E}(\mathtt{Q}, \sum_{y \in V_\delta} \xi_y))$, *and*,
   (b) $(\Psi_T) \stackrel{?}{=} \mathbb{E}(\mathtt{Q}, (\Delta_{T_\delta} + \sum_{y \in V_\delta} \xi_y))$.
4. 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, Δ_{T_δ} and/or Ψ_T have been tampered with.

3.5.2 Verification of Structural Relations

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 steps are as follows:

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 $(\theta_x \neq \theta_{\hat{p}_y})$, 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 3.1): $y \prec z \Leftrightarrow (lsb(\mathcal{H}(\theta_y \| \theta_z)) = 1) \wedge (lsb(\mathcal{H}(\theta_z \| \theta_y)) = 0)$.
   (b) For scheme-2 (Section 3.2):
      i. $j \leftarrow 1 + (\mathcal{H}(\theta_y^r \| \theta_z^r) \bmod L)$
      ii. $j' \leftarrow 1 + (\mathcal{H}(\theta_z^r \| \theta_y^r) \bmod L)$
      iii. b_y and b_z are the j'th, and b'_y and b'_z are the (j')'th bits in θ_y and θ_z respectively.
      iv. $y \prec z \Leftrightarrow b_y \oplus b_z < b'_y \oplus b'_z$.

4. GRAPHS

Our proposed authentication scheme for graphs is a general one. It can be used for trees, DAGs, as well as graphs with cycles. However, for trees, it is recommended to use the scheme specifically developed for trees (in Section 3), especially when the fact that the data is organized as trees need not be kept as a secret from users. The scheme for trees computes only one signature and is more efficient than that for the graphs (described below). Moreover, as we will see later (Section 6), the scheme for graphs can be used for signing forests of trees and graphs as well as authenticate such forests in a leakage-free manner.

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 (this DAG can be turned into a cyclic graph, by adding a back-edge such as one from f to 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 $(\alpha(x) - \alpha_\delta(x))$: whether it is empty or non-empty, what is its size, etc? For example, c has 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.

To that end, we define a notion of "signature of the immediate ancestors" of a node. Recall that in the case of trees, the integrity verifier of a node involves the secure name of its parent (Definition 3.1). In case of graphs, the signature of the immediate ancestors of a node x play the role of the "secure name of the parent of a node". Such a signature is computed as an aggregate signature so that it facilitates the authentication of a (possibly empty) subset $\alpha_\delta(x)$ ($\subseteq \alpha(x)$) of immediate ancestors without leaking any information about the remaining immediate ancestors ($\alpha(x)$ - $\alpha_\delta(x)$).

The signature of the immediate ancestors of a node x denoted by Ψ_x^α is computed over the set of all immediate ancestors $\alpha(x)$ of x. Such a signature of a node c can be used by a user to authenticate its immediate ancestors without leaking. For example, if a user has access to c and h but not f (such as in G_δ), then the following can be used to authenticate the fact that h indeed is an immediate ancestor of c, without leaking any information about f: Ψ_c^α, c, h, $e(c,h)$ and a item Δ_c^α (part of $\mathcal{VO}_{G_\delta}$).

The computation of such a signature Ψ_x^α is dependent on the contents of all its immediate ancestors but not on the structural order between them, if any. In order to compute Ψ_x^α, we define an integrity verifier ξ_x^α of a node with respect to its set of immediate ancestors $\alpha(x)$. The signature is salted and the salt is referred to by ω_x^α (in order to hide the facts that x (such as h) does not have any immediate ancestor or a user has access to all the immediate ancestors). ξ_x^α is a hash of a distinct random η_x and the content c_x of x. Ψ_x^α is either a CRSA-based or BGLS-based aggregate signature computed using the integrity verifiers ξ_y^α of all of its immediate ancestors ($y \in \alpha(x)$). The following sections describes how to compute the signature of immediate ancestors and the secure name of a node in a subgraph.

4.1 Secure Names and Immediate Ancestors

Let us define the integrity verifiers and the signatures of immediate ancestors first.

DEFINITION 4.1 (*IV* OF IMM. ANCESTORS). *Let x be a node in graph $G(V,E)$. Let η_x be a random assigned to x, and let c_x be the contents of x. The integrity verifier ξ_x^α of x in the context of its immediate ancestors of x is defined as: $\xi_x^\alpha \leftarrow \mathcal{H}(\eta_x \| c_x)$.*

DEFINITION 4.2 (CRSA-SIGNATURE OF IMM. ANCESTORS). *Let x be a node in graph $G(V,E)$. Let ω_x^α be a salt defined as: $\omega_x^\alpha \leftarrow \mathcal{H}$(distinct random). Let $\Omega_x^\alpha \leftarrow (\omega_x^\alpha)^{\bar{d}} \ mod\ \bar{n}$. Let $\Psi_x^\alpha \leftarrow (\xi_y^\alpha)^{\bar{d}} \ mod\ \bar{n}$. The signature $\Psi_{G(V,E)}^\alpha$ of $G(V,E)$ based on CRSA is defined as:*

$$\Psi_x^\alpha \leftarrow \Omega_x^\alpha \prod_{y \in \alpha(x)} \Psi_y^\alpha \ mod\ \bar{n}. \quad (3)$$

In the computation of CRSA-based signature, we reduce the number of modular exponentiations by applying the optimization described in the context of trees in Section 3.3.

DEFINITION 4.3 (BGLS-SIGNATURE OF IMM. ANCESTORS). *Let x be a node in graph $G(V,E)$. Let ω_x^α be a random. The signature $\Psi^\alpha_{G(V,E)}$ of $G(V,E)$ based on BGLS is defined as:*

$$\Psi_x^\alpha \leftarrow \mathbb{E}(\mathtt{P}, \underline{\mathtt{sk}}(\omega_{\mathtt{x}}^\alpha + \sum_{\mathtt{y}\in\alpha(\mathtt{x})} \xi_{\mathtt{x}}^\alpha)). \quad (4)$$

Algorithm for Computation of Secure Names for Graphs:

1. Let $G(V,E)$ be a graph and $x \in V$.
2. Let $\alpha(x)$ denote the set of immediate ancestors of x in graph $G(V,E)$.
3. Let Ψ_x^α denote the "signature of the immediate ancestors" of x (to be computed in the following steps).
4. For each node x,
   (a) Assign a distinct random η_x.
   (b) Assign a salt ω_x^α to x with respect to the ancestors of x.
   (c) Compute the integrity verifier $\xi_x^\alpha \leftarrow \mathcal{H}(\eta_x \| c_x)$.
   (d) If CRSA is used, compute the "signature of the immediate ancestors" Ψ_x^α as follows:
      i. $\Omega_x^\alpha \leftarrow (\omega_x^\alpha)^{\bar{d}}$ *mod* $\bar{n}$.
      ii. For each $y \in \alpha(x)$, $\Psi_y^\alpha \leftarrow (\xi_y^\alpha)^{\bar{d}}$ *mod* $\bar{n}$.
      iii. Compute the signature Ψ_x^α by applying Eq. 3.
   (e) If BGLS is used, compute the "signature of the immediate ancestors" Ψ_x^α by applying Eq. 4.
5. If G does not have any ordering between the siblings, then: for each node x, assign a random θ_x (or as an optimization, use η_x as θ_x).
6. Else (i.e., G is an ordered DAG - it has structural order among sibling nodes), compute the secure names θ_x for each node x as follows.
   (a) If Scheme-1 (Section 3.1) is used to compute θ_x, then use Scheme-1 as it is except the way $\lambda_\prec$ and $\lambda_\succ$ are computed, which is given below.
      i. Let the notations r, $v_{\pi(j)}$, x, $\lambda_\prec$, $\lambda_\succ$ have the same meaning as they do in the context of Scheme-1.
      ii. Compute $\lambda_\prec$ as follows (instead of as in Step 5$(b)(i)$ in Scheme-1): $\lambda_\prec \leftarrow \mathcal{H}(\theta_{v_{\pi(j)}} \parallel r \parallel \Psi_x^\alpha)$
      iii. Compute $\lambda_\succ$ as follows (instead of as in Step 5$(b)(ii)$ in Scheme-1): $\lambda_\succ \leftarrow \mathcal{H}(r \parallel \theta_{v_{\pi(j)}} \parallel \Psi_x^\alpha)$.
   (b) If Scheme-2 (Section 3.2) is used to establish the structural order among siblings, then use Scheme-2 as it is except the way $\lambda_\prec$ and $\lambda_\succ$ are computed, which is given below.
      i. Let the notations v_i, x, $\theta^r_{v_i}$, and θ^r_x have the same meaning as they do in the context of Scheme-2.
      ii. Compute $\lambda_\prec$ as follows (instead of as in Step 6(a) in Scheme-2): $\lambda_\prec \leftarrow \mathcal{H}(\theta^r_{v_i} \parallel \theta^r_x \parallel \Psi_x^\alpha)$
      iii. Compute $\lambda_\succ$ as follows (instead of as in Step 6(b) in Scheme-2): $\lambda_\succ \leftarrow \mathcal{H}(\theta^r_x \parallel \theta^r_{v_i} \parallel \Psi_x^\alpha)$.

4.2 Leakage-free Signatures of Graphs (Sign)

Now let us discuss how to compute the signature $\Psi_{G(V,E)}$ (also referred to as Ψ_G) of the actual graph $G(V,E)$. An integrity verifier ξ_x for each node x in the graph is computed using θ_x, contents c_x (as in the case of trees) as well as Ψ_x^α, and η_x. The signature Ψ_G of the graph is computed using the integrity verifiers of its nodes and are of two types: CRSA-based or BGLS-based.

DEFINITION 4.4 (INTEGRITY VERIFIER). *Let x be a node in tree $G(V,E)$, and c_x be the content of node x. Its integrity verifier (IV) denoted by ξ_x, is defined as: $\xi_x \leftarrow \mathcal{H}(\theta_x \| \eta_x \| \Psi_x^\alpha \| c_x)$.*

DEFINITION 4.5 (SIGNATURE OF GRAPHS USING CRSA). *Let $\mathcal{H}$ denote a random oracl, and ω_G be a random. The RSA signature of each node x in a graph $G(V,E)$ is $\Psi_x \leftarrow \xi_x^{\bar{d}}$ mod $\bar{n}$. Let the salt ω_G be a random. The signature of G, denoted by Ψ_G, is defined as*

$$\Psi_G \leftarrow (\omega_G \prod_{x\in V} \xi_x)^{\bar{d}} \ mod\ \bar{n}. \quad (5)$$

DEFINITION 4.6 (SIGNATURE OF GRAPHS USING BGLS). *Let $\mathcal{H}$ denote a random oracle, and ω_G be a random. The signature Ψ_x of each node $x$is defined as:$\Psi_x \leftarrow \underline{\mathtt{sk}}\xi_x$. The signature of G, denoted by $\Psi_{T(V,E)}$, is defined as*

$$\Psi_G \leftarrow \mathbb{E}(\mathtt{P}, \underline{\mathtt{sk}}(\omega_{\mathtt{G}} + \sum_{\mathtt{x}\ \in\ \mathtt{V}} \xi_{\mathtt{x}})). \quad (6)$$

Algorithm for Signing a Graph $G(V,E)$:

1. For each node $x \in V$,
   (a) Compute the signature of its immediate ancestors Ψ_x^α (either CRSA or BGLS).
   (b) For each node x, compute its secure name θ_x.
   (c) For each node x, compute its integrity verifier $\xi_x \leftarrow \mathcal{H}(\theta_x \| \eta_x \| \Psi_x^\alpha \| c_x)$.
2. Assign a salt ω_G to G: $\omega_G \leftarrow \mathcal{H}$(a distinct random).
3. If CRSA is used, compute the "signature of the graph" Ψ_G:
   (a) For each $x \in V$, $\Psi_x \leftarrow (\xi_x)^{\bar{d}}$ *mod* $\bar{n}$.
   (b) Evaluate Eq. 5, where $\Omega_G \leftarrow {\omega_G}^{\bar{d}}$ *mod* $\bar{n}$.
4. If BGLS is used, compute Ψ_G by evaluating Eq. 6.

The signature scheme (including the scheme for computing secure names for signatures of immediate ancestors) has a complexity of $O(|V| + |E|)$.

4.3 Distribution of Graphs (Dist)

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

Computation of the verification object $\mathcal{VO}_{G_\delta(V_\delta,E_\delta)}$:

1. For each node $x \in V_\delta$, compute Δ_x^α as follows.
   (a) Let $\alpha_\delta(x)$ be the set of all immediate ancestors of x that are in G_δ (Note that $\alpha_\delta(x) \subseteq \alpha(x)$).
   (b) CRSA: $\Delta_x^\alpha \leftarrow \omega_x^\alpha \prod_{w\in(\alpha(x)-\alpha_\delta(x))} \xi_w^\alpha$ *mod* $\bar{n}$.
   (c) BGLS: $\Delta_x^\alpha \leftarrow \omega_x^\alpha + \sum_{w\in(\alpha(x)-\alpha_\delta(x))} \xi_w^\alpha$.
2. Compute Ψ_{G_δ} and Δ_{G_δ} as follows.
   (a) CRSA: (a)$\Psi_{G_\delta} \leftarrow \prod_{y\in V_\delta} \Psi_y$ *mod* $\bar{n}$.
   (b) $\Delta_{G_\delta} \leftarrow \omega_G \prod_{y\in V-V_\delta} \xi_y$ *mod* $\bar{n}$.
   (b) BGLS: (a)$\Psi_{G_\delta} \leftarrow \mathbb{E}(\mathtt{P}, \sum_{\mathtt{y}\in\mathtt{V}_\delta} \Psi_{\mathtt{y}})$.
   (b) $\Delta_{G_\delta} \leftarrow \omega_G + \sum_{y\in V-V_\delta} \xi_y$.
3. $\Theta_{G_\delta} \leftarrow \{(\theta_x, \eta_x, \Psi_x^\alpha, \Delta_x^\alpha) \mid x \in V_\delta\})$.
4. $\mathcal{VO}_{G_\delta} \leftarrow \langle \Psi_{G_\delta}, \Delta_{G_\delta}, \Theta_{G_\delta} \rangle$.

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. Ψ_{G_δ} is used to verify the signature of al 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. θ_x is the secure name of x, η_x is the random used to compute the integrity verifier for immediate ancestors, Ψ_x^α is the signature of the immediate ancestors of x. Δ_x^α is used to verify the signature of the immediate ancestors of x.

Example: $\mathcal{D}$ has to send G_δ in our example to Bob. Ψ_G is a CRSA-signature. For the nodes a, b, c and d, $\mathcal{D}$ computes the Δ_a^α, Δ_b^α, Δ_c^α, and Δ_d^α, respectively. Δ_c^α is the modular multiplication of the salt ω_c^α and the "integrity verifier for immediate ancestors" of h, because h is not an immediate ancestor of c in G_δ. For b, Δ_b^α is just the $\omega_a^\alpha \pmod{\bar{n}}$ because all the immediate ancestors of b (i.e., a and d) are in G_δ. Next, $\mathcal{D}$ 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 Ψ_{G_δ}, Δ_{G_δ} and a set consisting of an element for each node in G_δ. Such an element for a consists of the secure name θ_a, η_a, signature of its immediate ancestors Ψ_a^α, and ξ_a^α. $\mathcal{D}$ then sends the signature of the graph Ψ_G and $\mathcal{VO}_{G_\delta}$ alongwith G_δ, to the user.

4.4 Authentication (`Vrfy`)

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

4.4.1 Authentication of the contents of subgraph G_δ

By contents we mean the contents of each node x as well as the following items: θ_x, η_x, Ψ_x^α, Δ_x^α. They are used for computing the integrity verifier of x (according to Definition 4.4). 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 Ψ_G. If the signature verifies, the contents are also verified. Authentication of contents of $G_\delta(V_\delta, E_\delta)$ has a complexity of $O(|V_\delta| + |E_\delta|)$.

Verification of Contents in a Subgraph:

1. For each node $x \in V_\delta$ in a subgraph $G_\delta(V_\delta, E_\delta)$, compute its integrity verifier: $\xi_x \leftarrow \mathcal{H}(\theta_x \| \eta_x \| \Psi_x^\alpha \| c_x)$.
2. CRSA: Compute (a) $((\Psi_{G_\delta})^{\bar{e}} \stackrel{?}{=} \prod_{x \in V_\delta} \xi_x \ (mod\ \bar{n}))$ *and*, (b) $((\Psi_G)^{\bar{e}} \stackrel{?}{=} \Delta_{G_\delta} \prod_{x \in V_\delta} \xi_x \ (mod\ \bar{n}))$.
3. BGLS: (a) $(\Psi_{G_\delta} \stackrel{?}{=} \mathbb{E}(\mathtt{Q}, \sum_{x \in V'} \xi_x))$ *and*, (b) $(\Psi_G \stackrel{?}{=} \mathbb{E}(\mathtt{Q}, \Delta_{G_\delta} + \sum_{x \in V'} \xi_x))$.
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 Ψ_G have been tampered with.

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 Ψ_G of the graph. If the verification turns out to be valid, the contents are authenticated.

4.4.2 Authentication of the structural relationships

In order to verify whether Bob has the correct set of immediate ancestors $\alpha_\delta(x)$ of each node x, Bob verifies the signature of the set of immediate ancestors Ψ_x^α. The verification process first computes the integrity verifier ξ_y^α of each node $y \in \alpha_\delta(x)$ using η_x and c_x, which are part of Θ_{G_δ}. In order to verify the signature Ψ_x^α with respect to the received set of immediate ancestors of x, the integrity verifiers of each such y (for x) are combined in the same manner as in Section 4.4.1 with the respective collective integrity verifier for Δ_x^α. If the signature is verified to be valid, x has the correct set of immediate ancestors in the subgraph.

In order to verify the integrity of the ordering among siblings (in ordered DAGs), Step 4(a) or 4(b) in Section 3.5.2 can be used for Scheme-1 or Scheme-2, respectively. Verification of immediate ancestors and the structural order between siblings in $G_\delta(V_\delta, E_\delta)$ has a complexity of $O(|V_\delta| + |E_\delta|)$.

Verification of Immediate Ancestors in a Subgraph:

1. For each node $x \in V_\delta$ in a subgraph $G_\delta(V_\delta, E_\delta)$,
   (a) Let $\alpha_\delta(x)$ be the set of immediate ancestors of x in G_δ. Verify the signature of the graph.
   (b) CRSA: If $(\Psi_x^\alpha)^{\bar{e}} \equiv \Delta_x^\alpha \prod_{y \in \alpha_\delta(x)} \xi_y^\alpha \ (mod\ \bar{n}))$, then $\alpha_\delta(x)$ is authenticated.
   (c) BGLS: If $(\Psi_x^\alpha == \mathbb{E}(\mathtt{Q}, \Delta_x^\alpha + \sum_{y \in \alpha_\delta(x)} \xi_y^\alpha))$, then $\alpha_\delta(x)$ is authenticated.

Example: Bob computes the "integrity verifiers for immediate ancestors" of a, b, c and d in G_δ in our example. Consider CRSA signatures. For c, Bob computes a modular multiplication of the integrity verifier ξ_d^α and the Δ_c^α received as part of Θ_{G_δ}, because d is the immediate ancestor of c in G_δ. Then Bob applies the signature verification process of CRSA on the result of this multiplication and the received signature Ψ_c^α of immediate ancestors of c. If the verification turns out to be valid, d is authenticated to be the immediate ancestor of c.

Optimization: Batch Verification of Immediate Ancestors in a Subgraph: The above scheme for verification of immediate ancestors of each x in G_δ can be carried out in constant ($O(1)$) time. We can use CRSA or BGLS aggregation towards this in the same manner as we batch verify the signature of the subgraph and the graph. The distributor computes an aggregate of Ψ_x^α for all x in V_δ (which is denoted by $\Psi_{G_\delta}^\alpha$), and the aggregate of Δ_x^α (which is denoted by $\Delta_{G_\delta}^\alpha$). The user *Bob* batch verifies the signature $\Psi_{G_\delta}^\alpha$ by $\Delta_{G_\delta}^\alpha$ and the received nodes.

5. SECURITY ANALYSIS

This section analyzes the soundness of the structural authentication scheme in terms of its authenticity and confidentiality guarantees with respect to information leakage defined earlier. The proofs are given in the Appendix.

5.1 Trees

LEMMA 5.1. *Under the Random Oracle Model, the signature scheme* Γ = (`Gen`, `Sign`, `Dist`, `Vrfy`) *for trees using either CRSA or BGLS is existentially unforgeable under the adaptive chosen-message attack, i.e.,* $\mathtt{Pr}(\mathtt{Sig\text{-}Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + \mathtt{negl}(n)$.

LEMMA 5.2. *Under the Random Oracle Model, the signature scheme* $\Gamma \equiv$ *(*`Gen`, `Sign`, `Dist`, `Vrfy`*) for trees using either CRSA or BGLS is indistinguishable under the adaptive chosen-signature attack, i.e.,* $\texttt{Pr}(\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + \texttt{negl}(n)$.

The proposed signature schemes for trees and graphs are also secure against the leakage of "absence" of extraneous information (indistinguishability). For example, the fact that a leaf node in a subtree is also a leaf node in the tree, cannot be inferred with non-negligible probablity by an adversary $\mathcal{A}$. With respect to an appropriate definition of signature indistinguishability for such a notion, it can be (formally) shown that the proposed signatures for trees and graphs are secure. Definition A.3 deals with the "presence" of extraneous information.

5.2 Graphs

LEMMA 5.3. *Under the Random Oracle Model, the signature scheme* $\Gamma =$ *(*`Gen`, `Sign`, `Dist`, `Vrfy`*) for graphs using either CRSA or BGLS is existentially unforgeable under the adaptive chosen-message attack, i.e.,* $\texttt{Pr}(\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + \texttt{negl}(n)$.

LEMMA 5.4. *Under the Random Oracle Model, the signature scheme* $\Gamma =$ *(*`Gen`, `Sign`, `Dist`, `Vrfy`*) for graphs using either CRSA or BGLS is indistinguishable under the adaptive chosen-signature attack, i.e.,* $\texttt{Pr}(\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + \texttt{negl}(n)$.

6. DISCUSSION

Forests of Trees and Graphs. Our scheme for graphs can be used to sign and authenticate forests, i.e., dis connected graphs (including trees) in a leakage-free manner. There are two approaches: (1) assign a dummy node and connect all the disconnected components to this dummy node. Since our scheme is provably leakage-free, signing such a transformed graph is also provabyly leakage-free. (2) treat each non-dis-connected (or weakly connected) component of the forest as a super-node, assign a salt to each such super-node, compute the signature of each super-node using the scheme for graphs, and then compute the signature of the forest as in the same manner as the signature of the immediate ancestors in a graph is computed, where each immediate ancestor is essentially one of the super-nodes (Section 4.1).

Dynamic Trees and Graphs. In order to inrementally 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 are local in nature, and do not get propagated up in a tree.

Answer Freshness and Prevention of Replay Attacks: The proposed authentication schemes prevent replay attacks and guarantee answer freshness by incorporating timestamps in the signatures as an extra element.

7. RELATED WORK

Integrity assurance of tree-structured data is primarily carried by the Merkle hash technique [15]. This scheme requires knowledge of certain extraneous information in order to verify the integrity of a subtree. It has been used in many scenarios such as: integrity assurance of XML [6], selective dissemination of XML data [3], integrity assurance of DAGs [14], verifying completion of query results [18], and authentication of text search results [17]. 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. For example, even a small amount of leakage of healthcare information may prove to be quite disastrous for a healthcare provider or the patient. Use of one-way accumulators [9] cannot also prevent leakage due to Merkle hash techniques and techniques derived from it.

There is little work concerning the problem of leakage-free integrity verification of trees and graphs. Kundu and Bertino [12] have investigated this problem and have proposed a notion of structural signatures for trees. However, the structural signature scheme for graphs proposed by them [13] is more expensive than the one proposed here 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). Moreover, the scheme proposed in this paper can be applied to implement privacy-preserving sets.

8. CONCLUSION AND FUTURE WORKS

In this paper, we solve the problem of how to authenticate trees and graphs without leaking. Such a problem has important applications such as in third party data distribution environments, cloud computing, and in privacy-preserving data mining. The proposed schemes are important in authenticating integrity of data as well as in protecting confidentiality of data and privacy of associated users.

We proposed two leakage-free authentication schemes: one for trees and another for graphs. The scheme for graphs is a general one - it can be used to sign as well as authenticate (at the user side) any form data organization structures - trees, DAGs, graphs with cycles, forests of trees and graphs. As part of the signature schemes, we also proposed to an efficient secure naming schemes; secure names are used to establish the sibling order between the nodes in case of ordered trees and ordered DAGs. The scheme for trees compute only one signature (based on CRSA or BGLS), whereas the scheme for graphs compute $O(m)$ number of signatures, where m is the number of nodes. Our schemes are highly scalable. Complexity analysis as well as performance results show that not only the scheme for trees but also the scheme for graphs incur linear cost. We also proved the security of the authentication schemes. To that end, we defined a notion of indistinguishability for signatures. How dynamic modifications of trees and graphs are authenticated are described. We also show how forests (e.g., a set of databases) can be authenticated without leaking.

The proposed authentication schemes have applications such as in healthcare databases, and in authentication of query results of biological and scientific databases. In future, we plan to apply this scheme to some of those domains, as well as in leakage-free assurance of data authenticity in cloud computing.

9. REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California, Berkeley, 2009.

[2] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT*, 1998.

[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 *EUROCRYPT*, 2003.

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

[6] 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.

[7] P. T. Devanbu, M. Gertz, Ch. U. Martel, and S. G. Stubblebine. Authentic data publication over the internet. *J. Comput. Secur.*, 11(3), 2003.

[8] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Special issue of Journal of Computer and Systems Sciences*, 28(2):270–299, April 1984.

[9] M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *ISC*, 2002.

[10] H. Hacigumus, S. Mehrotra, and B. Iyer. Providing database as a service. In *ICDE*, 2002.

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

[12] A. Kundu and E. Bertino. Structural signatures for tree data structures. In *VLDB*, 2008.

[13] A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *EDBT*, 2010.

[14] 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.

[15] R. C. Merkle. A certified digital signature. In *CRYPTO*, 1989.

[16] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *ACM Trans. of Storage*, 2(2):107–138, 2006.

[17] HweeHwa Pang and Kyriakos Mouratidis. Authenticating the query results of text search engines. In *VLDB*, 2008.

[18] Hweehwa Pang and Kian-Lee Tan. Verifying completeness of relational query answers from online servers. *ACM TISSEC*, 11(2):1–50, 2008.

APPENDIX

A. SIGNATURE SECURITY

DEFINITION A.1 (LEAKAGE-FREE SIGNATURE SCHEME Γ). *Let $\Upsilon(V,E)$ refer to either a tree or a graph, and $\Upsilon_\delta(V_\delta,E_\delta) \subseteq \Upsilon(V,E)$. A leakage-free signature scheme Γ for a tuple of three probabilistic polynomial algorithms and one deterministic algorithm (*`Vrfy`*) $\Gamma \equiv$ (*`Gen`*, *`Sign`*, *`Dist`*, *`Vrfy`*) satisfying the following requirements:*

1. *A key generation algorithm* `Gen` *takes as input a security parameter 1^n and outputs a pair of keys* (`pk`, `sk`)*, where* `pk` *and* `sk` *are the public and private keys, respectively. We assume for convention that each of these keys has length n, and that n can be determined from* `pk` *and* `sk`*.*
2. *The signing algorithm* `Sign` *takes as input a private key* `sk` *and tree/graph Υ, where the content c_x of each node $x \in V$ is such that $c_x \in \{0,1\}^*$. It (i.e.,* $\texttt{Sign}_{\underline{\texttt{sk}}}(\Upsilon(V,E))$*) outputs a signature Ψ_Υ and a* verification object $\mathcal{VO}_\Upsilon$*.*
3. *The distribution algorithm* `Dist` *takes as input $\Upsilon(V,E)$, its signature Ψ_Υ, the verification object $\mathcal{VO}_\Upsilon$, and subtree/subgraph (depending on whether Υ is a reference to a tree or a graph) $\Upsilon_\delta(V_\delta,E_\delta) \subseteq \Upsilon(V,E)$.* $\texttt{Dist}(\Upsilon,\Psi_\Upsilon,\mathcal{VO}_\Upsilon,\Upsilon_\delta)$ *outputs a verification object $\mathcal{VO}_{\Upsilon_\delta}$ for Υ_δ.*
4. *The deterministic verification algorithm* `Vrfy` *takes as input a public key* `pk`*, a subtree/subgraph $\Upsilon_\delta(V_\delta,E_\delta)$, whose integrity needs to be verified, a signature Ψ_Υ, and a verification object $\mathcal{VO}_{\Upsilon_\delta}$.* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\Upsilon_\delta(V_\delta,E_\delta),\mathcal{VO}_{\Upsilon_\delta},\Psi_\Upsilon)$ *outputs a bit b, with $b=1$ meaning* `valid` *(i.e., Υ_δ has not been tampered with) and $b=0$ meaning* `invalid` *(i.e., Υ_δ has been tampered with).*

Existentially unforgeable under adaptive chosen-message attack:

DEFINITION A.2 (SIGNATURE: INTEGRITY). *Consider the signature-forging experiment* $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$*:*

1. $\texttt{Gen}(1^n)$ *is run to obtain keys* (`pk`, `sk`)*.*
2. *Probabilistic polynomial-time (or simply, poly-time) adversary $\mathcal{A}$ is given* `pk`*, and oracle access to* $\texttt{Sign}_{\underline{\texttt{sk}}}(\cdot)$*, and* $\texttt{Dist}(\cdot)$*, which in turn have oracle access to $\mathcal{H}(\cdot)$. $\mathcal{A}$ then outputs the signature Ψ_Υ for Υ. $\mathcal{A}$ also outputs one or more distinct pairs $(\Upsilon_\delta,\mathcal{VO}_{\Upsilon_\delta})$, where $\mathcal{VO}_{\Upsilon_\delta}$ is a verification object for $\Upsilon_\delta \subseteq \Upsilon$. Let $\mathcal{Q}$ be the set consisting of Υ's (and $\alpha(x)$'s for a node x in a graph Υ), whose signatures were requested by $\mathcal{A}$ from* `Sign` *during its execution.*
3. *The output of the experiment is defined to be* 1 *if either (a), (b) or (c) holds, otherwise the output is 0:*
   - *(a) $\Upsilon \notin \mathcal{Q}$, $\Upsilon_\delta \subseteq \Upsilon$:* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\Upsilon_\delta,\mathcal{VO}_{\Upsilon_\delta},\Psi_\Upsilon)=1$*.*
   - *(b) $\Upsilon_\delta \subseteq \Upsilon$, $\Upsilon'_\delta \neq \Upsilon_\delta$:* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\Upsilon_\delta,\mathcal{VO}_{\Upsilon'_\delta},\Psi_\Upsilon)=1$*. (This is equivalent to the following:* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\Upsilon'_\delta,\mathcal{VO}_{\Upsilon_\delta},\Psi_\Upsilon)=1$*.)*
   - *(c) $\Upsilon'_\delta \not\subseteq \Upsilon$:* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\Upsilon'_\delta,\mathcal{VO}_{\Upsilon'_\delta},\Psi_\Upsilon)=1$*.*

The signature scheme $\Gamma \equiv$ (`Gen`*, *`Sign`*, *`Dist`*, *`Vrfy`*) is existentially unforgeable under adaptive chosen-message attack if*
$\Pr(\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + \texttt{negl}(n)$.[1]

[1] $\texttt{negl}(n)$ denotes a negligible function defined as follows: if for every polynomial $p(.)$, an integer N exists such that for all integers $n > N$ it holds that $\texttt{negl}(n) < \frac{1}{p(n)}$ ([11]:Definition 3.4).

Indistinguishable under the adaptive chosen-signature attack (leakage-free):

DEFINITION A.3 (SIGNATURE: INDISTINGUISHABILITY). *Consider the signature-indistinguishability experiment* $\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)$*:*

1. $\texttt{Gen}(1^n)$ *is run to obtain keys* $(\underline{\texttt{pk}}, \underline{\texttt{sk}})$.
2. *Probabilistic polynomial-time adversary* $\mathcal{A}$ *is given* $\underline{\texttt{pk}}$ *and oracle access to* $\texttt{Sign}_{\underline{\texttt{sk}}}(\cdot)$, $\texttt{Dist}(\cdot)$ *and* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\cdot)$*, which in turn have oracle access to the* $\mathcal{H}(\cdot)$*. A outputs an object* $\Upsilon_0(V_0, E_0)$*, which is a tree or a graph. Draw a random a from {0,1}. If a is 0,* $\Upsilon_1(V_1, E_1)$ *refers to* $\Upsilon_0(V_0, E_0)$*, else create* $\Upsilon_1(V_1, E_1)$ *such that* $\Upsilon_0 \subset \Upsilon_1$*, and if* Υ_0 *is a tree, then* Υ_1 *is also a tree. A signature* Ψ_{Υ_1} *and* $\mathcal{VO}_{\Upsilon_1}$ *are computed by* $\texttt{Sign}_{\underline{\texttt{sk}}}(\Upsilon_1(V_1, E_1))$. $\texttt{Dist}(\Upsilon_1, \Psi_{\Upsilon_1}, \mathcal{VO}_{\Upsilon_1}, \Upsilon_0)$ *computes* $\mathcal{VO}_{\Upsilon_0}$ *for* $\Upsilon_0(V_0, E_0)$*, and the* challenge $(\Upsilon_0(V_0, E_0), \mathcal{VO}_{\Upsilon_0}, \Psi_{\Upsilon_1})$ *is given to* $\mathcal{A}$.
3. *The adversary* $\mathcal{A}$ *continues to have oracle access to* $\texttt{Sign}_{\underline{\texttt{sk}}}(\cdot)$, $\texttt{Dist}(\cdot)$*, and* $\texttt{Vrfy}_{\underline{\texttt{pk}}}(\cdot)$*, which in turn have oracle access to the* $\mathcal{H}(\cdot)$*. Eventually,* $\mathcal{A}$ *outputs a structure (i.e., with no content in any node)* $\widetilde{\Upsilon}'(\widetilde{V}', E')$.
4. *The output of the experiment is 1 if* $\widetilde{\Upsilon_0}(\widetilde{V}_0, E_0) \subset \widetilde{\Upsilon}'(\widetilde{V}', E') \subseteq \widetilde{\Upsilon_1}(\widetilde{V}_1, E_1)$ *holds, and 0 otherwise.*

The signature scheme $\Gamma \equiv (\texttt{Gen}, \texttt{Sign}, \texttt{Dist}, \texttt{Vrfy})$ *is indistinguishable under the adaptive chosen-signature attack (leakage-free) if* $\texttt{Pr}(\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)=1) \leq \frac{1}{2} + min(\texttt{negl}(n), \texttt{negl}(m))$*, where the number of nodes in* Υ_1 *be m.*

B. PROOFS FOR TREES

PROOF (SKETCH) OF LEMMA 5.1. Let us first consider the authentication of contents of the nodes. In Γ with CRSA, the authentication of the contents in the nodes of $T_\delta \subseteq T$, $\texttt{Vrfy}$ computes $\Delta_{T_\delta} \prod_{y \in V_\delta} \xi_y$. Substituting $\omega_T \prod_{x \in (V - V_\delta)} \xi_x$ for Δ_{T_δ}, we get $\Delta_{T_\delta} \prod_{y \in V_\delta} \xi_y = (\omega_T \prod_{x \in (V - V_\delta)} \xi_x) \prod_{y \in V_\delta} \xi_y$ $= \omega_T \prod_{x \in V} \xi_x = (\Psi_{T(V,E)})^{\bar{e}}$ $(mod\ \bar{n})$ (by Definition 3.2).

Similarly, when BGLS is used, the signature works because, $\texttt{Vrfy}$ computes: $\mathbb{E}(\mathtt{Q}, (\Delta_{T_\delta} + \sum_{y \in V_\delta} \xi_y))$ $= \mathbb{E}(\mathtt{Q}, (\omega_T + \sum_{x \in (V - V_\delta)} \xi_x + \sum_{y \in V_\delta} \xi_y)) = \mathbb{E}(\mathtt{Q}, (\omega_T + \sum_{x \in V} \xi_x))$ $= \Psi_{T(V,E)}$ (by Definition 3.3).

Now consider the conditions in which the experiment $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ would output 1 with a non-negligible probability. Forger $\mathcal{F}$ is a polynomial time algorithm that invokes adversary $\mathcal{A}$ in order to forge the signature of a tree:

1. Forger generates a $T(V, E)$, and sends it to $\mathcal{A}$.
2. If $T(V, E)$ has not been signed by $\texttt{Sign}_{\cdot}(,)$ $\mathcal{A}$ computes Ψ_T and sends $(T(V, E), \mathcal{VO}_T, \Psi_T)$ back to $\mathcal{F}$.

Suppose that the condition Step 3(a) in this experiment holds. for $T \notin \mathcal{Q}$, *and* for $T_\delta(V_\delta, E_\delta) \subseteq T(V, E)$, $\texttt{Vrfy}_{\underline{\texttt{pk}}}(T_\delta(V_\delta, E_\delta), \mathcal{VO}_{T_\delta}, \Psi_T) = 1$. Note that Δ_T (in $\mathcal{VO}_T$) is same as ω_T. If $\mathcal{F}$ is capable of coming up with a signature Ψ_T for T in polynomial time, then Condensed-RSA can been forged in polynomial time, which in turn implies that the batch RSA signature can be broken. However, Condensed-RSA is known to be existentially unforgeable under adaptive chosen-message attack [16] due to the fact that batch-RSA is also existentially unforgeable under adaptive chosen-message attack [2]. Therefore, the probability for the experiment $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ to return 1 is negligible: $\texttt{Pr}(\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \texttt{negl}(n)$. Similarly, if BGLS is used, and $\mathcal{F}$ is capable of forging Ψ_T for some tree, then the BGLS aggregate signature scheme is broken by the polynomial-time algorithm $\mathcal{F}$. However, BGLS is existentially unforgeable under adaptive chosen-message attack [4]. Therefore, $\texttt{Pr}(\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \texttt{negl}(n)$.

Next, consider that the condition Step 3(b) in the experiment holds: for $T_\delta(V_\delta, E_\delta) \subseteq T(V, E)$, $T'_\delta \neq T_\delta$, $\texttt{Vrfy}_{\underline{\texttt{pk}}}(T_\delta(V_\delta, E_\delta), \mathcal{VO}_{T'_\delta}, \Psi_T) = 1$. The signature Ψ_T is returned by $\texttt{Sign}_{\underline{\texttt{sk}}}(T(V, E))$. The forger $\mathcal{F}$ implements a polynomial time algorithm that invokes $\mathcal{A}$ in order to authenticate a genuine subtree $T_\delta(V_\delta, E_\delta)$ with the verification object $\mathcal{VO}_{T'_\delta}$ of some other subtree T'_δ, i.e., either $\texttt{Dist}(T(V, E), \Psi_T, \mathcal{VO}_T, T_\delta)$ was not invoked by $\mathcal{A}$ or the verification object that it returned is not same as $\mathcal{VO}_{T'_\delta}$. $\texttt{Vrfy}_{\underline{\texttt{pk}}}((T_\delta(V_\delta, E_\delta), \mathcal{VO}_{T'_\delta}, \Psi_T))$ computes $\Delta_{T'_\delta} \prod_{y \in V_\delta} \xi_y$. Substituting $\Delta_{T'_\delta} = \omega_{T'} \prod_{x \in (V' - V'_\delta)} \xi_x$, where T' may or may not be the same tree as T, we get $\Delta_{T'_\delta} \prod_{y \in V_\delta} \xi_y =$ $(\omega_{T'} \prod_{x \in (V' - V'_\delta)} \xi_x)(\prod_{y \in V_\delta} \xi_y)$. In case, T' is same as T, then the above translates to $(\omega_T \prod_{x \in (V - V'_\delta)} \xi_x)(\prod_{y \in V_\delta} \xi_y)$. Since the forgery by $\mathcal{F}$ succeeds, this computation in fact leads to Ψ_T: $(\omega_T \prod_{x \in (V - V'_\delta)} \xi_x)$ $(\prod_{y \in V_\delta} \xi_y) = \Psi_T$. Note that $V'_\delta \neq V_\delta$, since $T'_\delta \neq T_\delta$. Therefore, a forgery on Condensed-RSA has been successful. In case, T' is a different tree than T, the fact that $(\omega_{T'} \prod_{x \in (V' - V'_\delta)} \xi_x)(\prod_{y \in V_\delta} \xi_y)$ translates to Ψ_T implies that a forgery on Condensed-RSA has been successful. Therefore, for the condition (b) in the context of CRSA, $\texttt{Pr}(\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \texttt{negl}(n)$. Similar arguments can be applied for BGLS to show that the probability for $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1$ is negligible.

Now consider that the condition Step 3(c) in $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ holds: for $T'_\delta(V'_\delta, E'_\delta) \not\subseteq T(V, E)$, $\texttt{Vrfy}_{\underline{\texttt{pk}}}(T'_\delta(V'_\delta, E'_\delta), \mathcal{VO}_{T'_\delta}, \Psi_T)$ $= 1$. $\texttt{Vrfy}_{\underline{\texttt{pk}}}(T'_\delta(V'_\delta, E'_\delta), \mathcal{VO}_{T'_\delta}, \Psi_T)$ computes $\Delta_{T'_\delta} \prod_{y \in V'_\delta} \xi_y$. Substituting $\omega_{T'} \prod_{x \in (V' - V'_\delta)} \xi_x$ for $\Delta_{T'_\delta}$, where T' is not the same tree as T, we get $\Delta_{T'_\delta} \prod_{y \in V_\delta} \xi_y$ $= (\omega_{T'} \prod_{x \in (V' - V'_\delta)} \xi_x)(\prod_{y \in V'_\delta} \xi_y) = \Psi_{T'(V',E')}$. Since the forgery by $\mathcal{F}$ succeeds, the signature $\Psi_{T'(V',E')}$ of tree $T'(V', E')$ is in fact same as the signature Ψ_T of a different tree $T(V, E)$. This in turn means we have broken the CRSA scheme. Similar arguments for BGLS can show that if a forger succeeds, then the BGLS scheme is broken. Therefore, the probability for $\texttt{Sig-Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ to be successful is negligible ($\texttt{negl}(n)$).

The $\texttt{Vrfy}$ procedure verifies the structural integrity after the contents of all the nodes in T_δ as well as the secure names (in $\mathcal{VO}_{T_\delta}$) are authenticated. Suppose that a forger $\mathcal{F}$ successfully modifies the relationship that x is the parent of y to "y is the parent of x". In order to achieve this, $\mathcal{F}$ cannot modify the secure name of x, and y, which otherwise would lead to the failure of authentication of contents. This is because, signature of T is dependent on the integrity verifier of x and y, which in turn are dependent on the secure names θ_x, $\theta_{\hat{p}_x}$, and θ_y. Another way in which $\mathcal{F}$ can be successful is as follows: Let ${\xi_x}' = \mathcal{H}(\theta_x \| \theta_y \| c_x)$ (y is parent of x), and ${\xi_y}'$ $= \mathcal{H}(\theta_y \| \theta_{\hat{p}_x} \| c_x)$ (the old parent of x $\hat{p}_x$ is the parent of y, after $\mathcal{F}$ modifies the relationship). Therefore $\mathcal{F}$ ensures that ${\xi_x}' = \mathcal{H}(\theta_x \| \theta_{\hat{p}_x} \| c_x)$, which is same as the integrity verifier ξ_x; and ${\xi_y}' = \mathcal{H}(\theta_y \| \theta_x \| c_x)$, which is same as the integrity verifier ξ_y. This leads to a collision for $\mathcal{H}$, which is not feasible under the Random Oracle Model.

If $\mathcal{F}$ succeeds in modifying the order between two siblings x and y, then $\mathcal{F}$ needs to modify θ_x or θ_y, which however is a hard problem. □

PROOF (SKETCH) OF LEMMA 5.2. Inferrer $\mathcal{F}$ is a polynomial time algorithm that invokes adversary $\mathcal{A}$ in order to infer extraneous information from the signature and verification objects of a tree using the experiment $\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)$:

1. $\mathcal{F}$ generates a $T_0(V_0, E_0)$, and sends it to $\mathcal{A}$.
2. $\mathcal{A}$ invokes $\mathtt{Sign}'_{\underline{\mathtt{sk}}}(T_0)$, which outputs Ψ_{T_1} and $\mathcal{VO}_{T_1}$;
3. $\mathcal{A}$ then invokes $\mathtt{Dist}(T_1, \Psi_{T_1}, \mathcal{VO}_{T_1}, T_0)$, which returns $(T_0, \mathcal{VO}_{T_0}, \Psi_{T_1})$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends a tree structure $\widetilde{T}'(\widetilde{V}', E')$ to $\mathcal{F}$.

In order to satisfy the condition in Step 5 of the experiment $\mathtt{Sig\text{-}Priv}^{csa}_{\mathcal{A},\Gamma}(n)$, $\mathcal{A}$ outputs a tree structure $\widetilde{T}'(\widetilde{V}', E')$ such that $T_0 \subset \widetilde{T}' \subseteq T_1$. $\widetilde{T}'(\widetilde{V}', E')$ is minimal as follows: a node w is in both V_1 and $\widetilde{V}'$ but not in V_0, an edge $e(x, w)$ is in both E_1 and $\widetilde{E}'$, but not in E_0, and x is in V_1, $\widetilde{V}'$, as well as in V_0. The following cases describe the possible positions of x in $\widetilde{T}'$ with respect to T_0:

1. x is a leaf in T_0, and w is child of x in $\widetilde{T}'$.
2. x is the root of T_0, and w is the parent of x in $\widetilde{T}'$.
3. x is an intermediate node in T_0, and w is the leftmost child of x in $\widetilde{T}'$.
4. x is an intermediate node in T_0, and w is the rightmost child of x in $\widetilde{T}'$.
5. x is an intermediate node in T_0 and y and z are children of x in T_0, and w is a child of x in $\widetilde{T}'$ such that $y \prec w \prec z$.

In all these cases, computation of Δ_{T_0} (in $\mathcal{VO}_{T_0}$=$\langle \Delta_{T_0}, \Theta_{T_0} \rangle$) (by $\mathtt{Dist}(\cdot)$) involves θ_w, because w is in T_1, but not in T_0. That is, $\Delta_{T_0} = \omega_{T_1} \prod_{z \in (V_1 - V_0)} \xi_z \ (mod\ \bar{n})$. The fact that $\mathcal{A}$ claims that w is in T_1 but not in T_0 (by incorporating w in $\widetilde{T}'$) implies that it has determined in polynomial time that other than ω_{T_1}, there is at least one more value ξ_w (which is pseudorandom due to Definition 3.1) involved in the computation of Δ_{T_0}, which however is hard. For Cases 1 to 4, it is not possible to infer the existence of w from Θ_{T_0}, which is the following set: $\{(\theta_x, \theta_{\hat{p}_x}) | x \in V_0\}$. For Case 5, the adversary $\mathcal{A}$ uses the knowledge of the secure names of the nodes in T_0 in order to show that w exists in $\widetilde{T}'$ as described in Case 5 (and in turn T_1, because $\widetilde{T}' \subseteq \widetilde{T}_1$). However, as we show in the next paragraphs, θ_y of each node y with a rank i among its siblings (the leftmost and rightmost siblings have ranks 1 and k, resp.), reveals something about i with only a negligible probability. Therefore, $\mathtt{Sig\text{-}Priv}^{csa}_{\mathcal{A},\Gamma}(n) = 1$ occurs with negligible probability $\mathtt{negl}(n)$, assuming that the size of the secure names is n-bits.

Scheme-1: Let all secure names be in the interval $[1 : U]$. Let y be a node with rank i among its siblings and be referred to as v_i (to remain consistent with the terminology used in the scheme). To prove that a secure name θ_{v_i} reveals nothing about i, it suffices to prove that the proposed process for secure-name computation is such that the probability of an θ_{v_i} being equal to any $u \in [1 : U]$ is independent of i. We write the event $\{\theta_{v_i} = u\}$ as the union of k disjoint events $E_1, \ldots, E_k$ where $E_j = \{\pi(i) = j, \theta_{v_i} = u\}$. We thus have: $\mathtt{Pr}(\theta_{v_i} = u) = \sum_{j=1}^{k} \mathtt{Pr}(\pi(i) = j, \theta_{v_i} = u) = \sum_{j=1}^{k} (1/k)(1/U)(1/4^{j-1})$
where we used the facts that: (i) $\mathtt{Pr}(\pi(i) = j) = 1/k$; (ii) θ_{v_i} is u iff Sub-step 3(a) selects u out of the U choices (with probability $1/U$) *and* that choice is not discarded in Sub-step 3(b), i.e., the choice is admissible relative to the $j-1$ other already assigned secure names (probability of non-discard is 4^{-j+1}). Because $\sum_{j=1}^{k} 1/4^{j-1} = (4/3)(1 - 4^{-k})$ we obtain: $\mathtt{Pr}(\theta_{v_i} = u) = (4/3kU)(1 - 4^{-k})$, which is independent of i, as required.

The secure names do not leak information on k or m (the number of nodes in the tree) either, because each secure name is drawn uniformly from a subset of $[1 : U]$ that is both large (of size approximately $4U/3k$) and uniform over all such subsets of $[1 : U]$, hence indistinguishable from a random choice over $[1 : U]$.

Scheme-2: As earlier, let all secure names be in the interval $[1 : U]$. To prove that a secure name θ_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 θ_x being either 0 or 1 is $\frac{1}{2}$, and it is true for all the bits in θ_x. We give a proof by induction.
Basis: Case I: x is the left-most child of its parent: θ_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 [11]. 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 θ_{v_k} is a random number, then θ_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 θ_x are already "used". By Step 2 in the scheme, two bits at positions j and j' that are still unused in θ_{v_k} are chosen. The $r(x)$ bits are randoms as well as the two bits at j and j' leftmost positions in θ_x are also randoms. The remaining bits are "not used" and chosen randomly. Thus θ_x is also a random. □

B.1 Proofs for Graphs

Proof (Sketch) of Lemma 5.3. Like in the case of the trees, the forger $\mathcal{F}$ who invokes the adversary $\mathcal{A}$ from a polynomial time algorithm can attack the signature scheme Γ by (1) forging the signature of the graph, and /or (2) forging a signature of the immediate ancestors for a node x. The proof that $\mathcal{A}$ can successfully carry out (1) has only a negligible probability, is analogous to the proof of unforgeability in case of trees (Lemma 5.1). In order to prove the unforgeability of Γ, the only thing we need to prove, which we provide here, is about (2): the probability of forging a signature of the immediate ancestors for a node x is negligible.

Let $\mathcal{F}$ forge the signature Ψ^{α}_x for node x in $G(V, E)$. Suppose that the condition (a) (Step 3(a)) in $\mathtt{Sig\text{-}Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ holds. For $\alpha(x) \notin \mathcal{Q}$, *and* for $\alpha_\delta(x) \subseteq \alpha(x)$,
$\mathtt{Vrfy}_{\underline{\mathtt{pk}}}(\alpha_\delta(x), \Delta^{\alpha}_x, \Psi^{\alpha}_x) = 1$. Note that Δ^{α}_x is same as ω^{α}_x. If $\mathcal{F}$ is capable of coming up with a signature Ψ^{α}_x for $\alpha(x)$ in polynomial time, then Condensed-RSA can been forged in polynomial time, which however is hard. Therefore, the probability for the experiment $\mathtt{Sig\text{-}Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ to return 1 is negligible: $\mathtt{Pr}(\mathtt{Sig\text{-}Forge}^{cma}_{\mathcal{A},\Gamma}(n)=1) \leq \mathtt{negl}(n)$. Similarly, (and in the line of the proofs for (b) and (c) in Lemma 5.1) it can be shown that the probability for $\mathtt{Sig\text{-}Forge}^{cma}_{\mathcal{A},\Gamma}(n)$ to output 1 for each of (b) and (c) is negligible. Since the

Ψ_x^α for any x does not involve the order between x and its siblings, if such an order exists, the signature of immediate ancestors is unforgeable. Thus the lemma is proven. □

Proof (Sketch) of Lemma 5.4. The claim that $\texttt{Pr}(\texttt{Sig-Priv}^{csa}_{\mathcal{A},\Gamma}(n)=1) \leq \texttt{negl}(n)$ for graphs $G(V, E)$ can be shown to be true as is done in the case of trees. □