

# Efficient Distributed Third-Party Data Authentication for Tree Hierarchies \*

Hao Yuan and Mikhail J. Atallah  
Purdue University  
{yuan3, mja}@cs.purdue.edu

## Abstract

*In the third-party model for distribution of data, the trusted data creator or owner provides an untrusted distributor  $D$  with integrity verification (IV) items that are stored at  $D$  in addition to the  $n$  data items. When a user  $U$  has a subset (of size  $n'$ ) of those  $n$  data items and needs to verify their integrity,  $U$  sends a query to  $D$  to request a number of IV items that can be used to verify its data's integrity. The model forbids  $U$  from receiving any information (in the information-theoretic sense) about the  $n - n'$  data items that the user is not authorized to access, and assumes that  $D$  has no signature authority (it stores only pre-signed IVs).*

*Most of the published work in this area uses the Merkle tree or variants thereof, and typically requires  $D$  to store a linear or close to linear (in  $n$ ) number  $s(n)$  of IV items that are pre-signed by the trusted authority. Moreover, most of the existing schemes impose on  $D$  a non-constant amount of computation work  $t(n)$  (typically logarithmic in  $n$ ) in order to provide  $U$  with the IV items that enable  $U$  to verify the integrity of its data; we call  $h(n)$  the number of such IV items that the user needs. The  $h(n)$  values found in the literature are non-constant, i.e., they depend on  $n$ .*

*The main contribution of this paper is to achieve linear or almost linear  $s(n)$ , constant  $h(n)$  and constant or logarithmic  $t(n)$  when the  $n$  data items are organized in a tree hierarchy  $T$ , and the user's subset of  $n'$  items form a subtree  $T'$ . The cases of  $T'$  considered are when  $T'$  is (i) rooted at a node  $v$  and of depth  $k$  below  $v$ ; and (ii) reachable in  $k$  hops from  $v$  going both up and down in  $T$ .*

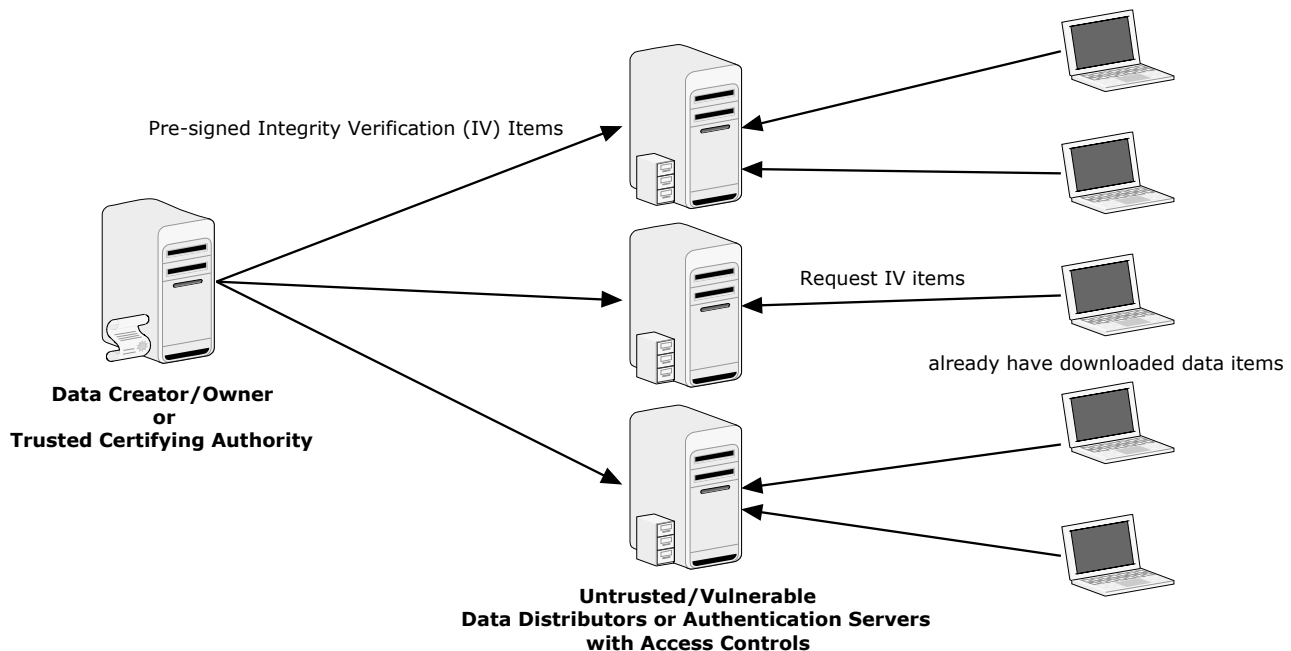
## 1. Introduction

When important decisions are made based on data (e.g., business, health-care, scientific, etc), it becomes especially important to provide a user  $U$  the means of verifying the integrity of the data to which he or she is entitled; such access rights for  $U$  occur because of  $U$ 's role in his or her organization, government security clearance level, or perhaps because of payment, subscription, etc. The integrity of the data is usually verified by the user  $U$  through comparison of a user-computed cryptographic hash of that data, to an “expected” value which is the hash of the original (untampered) version of that data: A mismatch indicates tampering. The user's belief in this “expected value” usually comes (directly or indirectly) from a digital signature done by a trusted entity (called Trent in this paper) that could be the creator of that data, or a certifying authority.

The distribution is often done through third parties who are “moderately untrusted” in the sense that the trusted data creator or owner relies on such a third party  $D$  for distribution but does not wish to provide  $D$  with the authority to sign on its behalf. We note that this may not be due to a lack of trust in  $D$ , but merely a recognition

---

\* A preliminary version of this work appeared in Proceedings of The 28th International Conference on Distributed Computing Systems (ICDCS 2008).



**Figure 1. Third-Parties Data Authentication Model.**

of the fact that typical systems such as  $D$ 's are more vulnerable (to break-ins, spyware, insider misbehavior, or simply accidents) than Trent's own systems. The third-party distribution model offers many economic advantages but also a challenge: How does the third-party distributor  $D$ , who does not have the authority to sign (only Trent does) prove to a user  $U$  the integrity of the user's data?

The obvious answer is to store at  $D$  a number of integrity verification ( $IV$ ) items that  $D$  can later on provide to any  $U$  who is legitimately requesting a subset of the data which  $U$  is authorized to access; these  $IV$ s, which are cryptographic hashes or signatures (by Trent) thereof, must enable the user  $U$  to verify the integrity of its own subset of the data. Under no circumstance should  $U$  be allowed access to more than his or her allowed subset of data, even though this can make the problem of verification easier: For example, a user  $U$  who is entitled to a subset of  $n/2$  items of the  $n$  data items stored at  $D$  cannot be supplied with a single  $IV$  for all  $n$  items, because  $U$  cannot make use of such an  $IV$  without seeing all  $n$  items. Moreover, supplying  $U$  with a cryptographic hash of data, which it is not authorized to access, is a security hazard as it enables enumerative brute-force attacks; our scheme is required not to send any information (in the information-theoretic sense) to  $U$  about data items that are not in  $U$ 's subset.

Minimizing the storage and computational burden on  $D$  is obviously of great importance, since  $D$  is a focal point of a potentially large number of integrity verification requests from users. The performance metrics are the followings.

- $s(n)$ , the storage space that is used up by the integrity verification information stored at  $D$  (i.e., the number of pre-signed  $IV$ s);
- $h(n)$ , the number of  $IV$ s that  $D$  needs to send to  $U$  (i.e., the communication complexity); Here, we assume that the size of a cryptographic hash or signature to be constant;
- $t(n)$ , the time for  $D$  to online compute those  $h(n)$   $IV$  items.

Of particular interest is the situation where those  $n'$  items have “logical proximity” to each other, as measured by the particular organization of the data – in GIS or image data the organization is as a grid, and proximity is measured by the distance within the grid; in hierarchically organized data proximity is measured by the distance along the edges of the tree hierarchy, with two cases possible depending on whether one ignores hierarchical edge directions or not: In the former case the relevant notion of “within a distance  $k$  from vertex  $v$ ” is of a *radius* from vertex  $v$  where one starts at  $v$  and can move down or up the tree (but no more than  $k$  hops), we call it  **$k$ -radius subtree authentication problem**; in the latter case it is the subtree of depth  $k$  rooted at  $v$ , we call this  **$k$ -depth subtree authentication problem**.

Here is a sample application for the  $k$ -radius subtree authentication problem. A road map is organized in a tree network shape, and each node of the tree may be associated with some for-pay information, e.g., the information can be some sort of data that help make important business decision. Someone who intends to start a local business may want to buy or subscribe to those data within a distance  $k$  of its local store (located at  $v$ ).

As a sample application for the  $k$ -depth subtree authentication problem, consider an information provider (e.g., financial data vendor) selling to users subscription products, and the products are organized in a tree hierarchy. Each node of the tree may represent a product of a specific-level of detailed information, and a child node of that node may represent a deeper level of the information. The user may want to subscribe to a subset of the products that he or she is interested in, but only to a limited depth in the tree, because the more detailed information the user subscribes, the more he or she needs to pay for the service provider.

We consider both of these cases for a tree hierarchy, and provide for each a solution that achieves

- $O(n)$  storage space for the  $k$ -depth subtree authentication problem;
- $O(n \log n)$  storage space for the  $k$ -radius subtree authentication problem;
- constant communication burden on  $D$  per user request, i.e.,  $h(n) = O(1)$ ;
- constant computation cost per user request for the  $k$ -depth subtree authentication problem, i.e.,  $t(n) = O(1)$ .
- logarithmic computation cost per user request for the  $k$ -radius subtree authentication problem, i.e.,  $t(n) = O(\log n)$ .

Our scheme does not hand the user any cryptographic hash of any items that are outside the user’s authorized range: it only hands the user an  $IV$  item that pertains to its authorized subtree. Therefore, our scheme does not leak any information in the information-theoretic sense.

This paper is organized as follows. Related work is covered in Section 2. Section 3 reviews pairing-based cryptography and defines notations. Section 4 and 5 give efficient authenticated data structures for  $k$ -depth subtree authentication and  $k$ -radius subtree authentication problem respectively. Section 6 concludes.

## 2. Related Work

The landmark work in this area is, of course, by Merkle [21] who suggested a digital signature system based on a secure conventional encryption function over a tree of document fragments. This technique has since been used extensively (more on this below), but with a logarithmic number of  $IV$ s to verify even a small number of data items. The work that uses the Merkle tree idea include the work of Devanbu *et al.* [8, 9] for authentication of relational and XML data; by Kocher [18] who described the use of a static Merkle hash tree that is distributed to third parties; by Naor *et al* [23] who described a dynamic version that supports logarithmic time insertion and deletion in a tree through both server and distributor maintaining identical copies of a 2-3 tree. Goodrich and Tamassia [11] provided a scheme that uses skip lists and commutative hash functions (one-way accumulators), and later extended their work to other data structures including range trees [12]. For small devices which have

limited computation power, low verification overhead and latency schemes were given in [19, 26, 27], including one-time signature schemes for multicasting or broadcasting (these schemes are secure for a short period of time and still require signatures). Perrig [25] proposed TESLA which is based on symmetric cryptography for secure multicast, but requires static data ordering. Devanbu and Martel *et al.* [8, 20] also suggested a data authentication and non-repudiation scheme in the untrusted third party based on a public key cryptosystem and a hash tree.

An approach with a similar integrity-verification goal, but a different way of achieving it, is to use a trusted server to both avoid the signer being a bottleneck and also reduce a user’s verification overhead. In [16], Horne *et al.* provided an escrow service infrastructure to verify P2P data, in which the escrow server is responsible for data verification and for payments to peers (in contrast to our present paper and the related literature on authenticated data structures, where the distributor is untrusted).

The third-party model for distribution of data is also studied in the form of outsourced databases, where queries are sent to untrusted servers and the integrities of the query results need to be verified [22, 24].

For k-radius subtree authentication problem and the k-depth subtree authentication problem, to the best of our knowledge, we do not know any previous results on them. A naive solution may pre-compute signatures for all possible k-radius or k-depth subtrees of the tree hierarchy, which will require  $O(n^2)$  storage space.

### 3. Preliminaries

#### 3.1. Paring-based Cryptography

Our efficient authentication scheme relies on paring-based cryptography. In [5], Boneh et al. constructed an aggregate signature scheme based on the work of [6] using bilinear maps. Their aggregate signature scheme can achieve the following : given  $n$  signatures on  $n$  distinct messages (data items) from  $n$  distinct users, any public user can aggregate the signatures into a single and short signature. Such a signature aggregation scheme is the fundamental factor for us to achieve constant time communication complexity: the third-party data distributor sends back only a short signature for a subtree authentication query. This, however, is not surprising with a linear computational burden – the real challenge is achieving it with a constant or logarithmic computational burden.

##### 3.1.1 Bilinear Maps

We briefly review bilinear maps here <sup>1</sup>. Let  $G$  and  $G_T$  be two (multiplicative) cyclic groups of prime order  $p$ , and  $g$  be a generator of  $G$ . A mapping  $e : G^2 \rightarrow G_T$  is a bilinear map if it satisfy the following notation

- Bilinearity:  $e(u^a, v^b) = e(u, v)^{ab}$  for all  $u, v \in G$  and  $a, b \in Z_p^*$
- Non-degeneracy: for any generator  $g$  of  $G$ , we have  $e(g, g) \neq 1$  (i.e.,  $e(g, g)$  is also a generator of  $G_T$ ).

In the cryptographic setting, we assume that there is an efficient (polynomial-time) algorithm to compute  $e(x, y)$  for any  $x, y \in G$ . Also, our scheme relies on the following DDH fact and CDH assumption.

- Decisional Diffie-Hellman (DDH) is easy in  $G$ : Given  $g, g^a, h, h^b \in G$  where  $a, b \in Z_p^*$  are unknown, there is a polynomial-time algorithm to answer whether  $a = b$ . (We can check to see if  $e(g, h^b) = e(g^a, h)$ ).
- Computational Diffie-Hellman (CDH) is difficult in  $G$ : Given  $g, g^a, h \in G$  where  $a \in Z_p^*$  is unknown, for any probabilistic polynomial-time algorithm (an adversary), the probability for the adversary to generate the correct  $h^a$  is negligible.

---

<sup>1</sup>Our notation is a simplified version from [5]: we set  $G = G_1 = G_2$ ,  $g = g_1 = g_2$  and use identity mapping  $\phi = I$ .

### 3.1.2 Aggregate Signature Scheme

We apply Boneh et al.'s aggregate signature scheme [5] based on the following setting: there is only one trusted authority (or one pair of public and private keys) which can sign messages.

**Setup:** The trusted authority has a pair of keys: a private key  $x \in Z_p^*$  which is chosen randomly; a public key  $v = g^x \in G$ . (The generator  $g$  is also public.)

**Signing:** For a single message  $M \in \{0, 1\}^*$ , we first compute  $h \leftarrow H(M)$ , where  $H : \{0, 1\}^* \rightarrow G$  is a hash function. Then we generate the signature  $\sigma \leftarrow h^x \in G$ .

**Verification:** Given the generator  $g$ , the public key  $v$ , a message  $M$  and a signature  $\sigma$ , the verifier first compute  $h \leftarrow H(M)$  and then check whether  $e(\sigma, g) = e(h, v)$ . ( If the signature is genuine, then we will have  $e(\sigma, g) = e(h, v) = e(h, g)^x$ .)

**Aggregation:** Given the signatures  $\sigma_1, \sigma_2 \dots \sigma_n \in G$  for  $n$  distinct messages<sup>2</sup>  $M_1, M_2, \dots, M_n \in \{0, 1\}^*$ , we can aggregate the signatures by setting  $\sigma = \prod_{i=1}^n \sigma_i$ , where  $\sigma$  is the aggregate signature.

**Aggregation Verification:** Given the aggregate signature of  $n$  distinct messages  $M_1, M_2, \dots, M_n \in \{0, 1\}^*$ , the verifier first computes  $h_i \leftarrow H(M_i)$  for  $1 \leq i \leq n$ , and then it accepts the signature if and only if  $e(\sigma, g) = \prod_{i=1}^n e(h_i, v)$  holds.

In Section 4 and Section 5, we will use the above aggregate signature scheme to generate a single signature for  $U$ 's query subtree, i.e., the scheme help compute a constant-size  $IV$  item.

## 3.2 Notations

If  $T$  is a tree, then we use  $V(T)$  to represent the node set of the tree and  $E(T)$  to represent the edge set. We say  $v \in T$  if  $v$  is a node of  $V(T)$ , and  $(u, v) \in T$  if the edge  $(u, v)$  belongs to  $E(T)$ . The distance between two nodes  $u$  and  $v$  (the number of edges on the path connecting  $u$  and  $v$ ) is denoted by  $d(u, v)$ . When  $u = v$ , we define  $d(u, v) = 0$ .

In the case of rooted tree  $T$ , we use  $d(T; v)$  to represent the depth of  $v$  in  $T$ , i.e.,  $d(T; v) = d(\text{root}(T), v)$  where  $\text{root}(T)$  denote the root of  $T$ . In some contexts, we use  $d(v)$  instead of  $d(T; v)$  if there is no ambiguous understanding of the implicit  $T$ .

Usually each node  $v$  is associated with a message  $M_v$ , we use  $S(v)$  to represent the signature for  $M_v$ . For a tree  $T$ , we use  $S(T)$  to represent the aggregate signature for all the messages associated with the nodes of  $T$ , i.e.,

$$S(T) = \prod_{v \in T} S(v).$$

For an empty tree, we define  $S(\emptyset) = 1$ , where 1 is the identity element in the group  $G$ .

Unless explicitly specified, throughout the rest of this paper, we use  $x$  and  $v$  to represent a node in a tree rather than making them the secret key and public key as in the pairing-based cryptography reviewed in subsection 3.1.

## 3.3 Related Data Structures on Trees

Our authenticated data structures for subtree authentication require the use of several classic data structures on trees as described below.

### 3.3.1 Distance of Two Nodes

**Lemma 1** *There is a linear space data structure, built in linear time, which enables us to calculate  $d(u, v)$  for any  $u$  and  $v$  in constant time.*

---

<sup>2</sup>The messages must be distinct in order to ensure the security of the aggregate signature scheme.

To achieve this, we specify a node  $r$  as the root (if it's the unrooted case), and then do a depth-first search to pre-compute the depth  $d(z)$  for each node  $z$ . For an online distance query for  $u$  and  $v$ , we first compute the nearest common ancestor of  $u$  and  $v$  in constant time by the method from [15, 2](using linear space, also built in linear time). Assume that the nearest common ancestor is  $w$ , then we have  $d(u, v) = d(u) + d(v) - 2d(w)$ , which can be computed in constant time.

### 3.3.2 Nearest Neighbor on a Path

**Lemma 2** *There is a linear space data structure, built in linear time, which enables us to answer the following query in constant time: given two distinct nodes  $u$  and  $v$ , we want to find out  $NEP(u, v)$ , which is the nearest (and only) edge of  $u$  on the path from  $u$  to  $v$ , i.e.,  $NEP(u, v) = (u, x)$  if the path from  $u$  to  $v$  is like  $ux \dots v$ .*

The constant-time computation of  $NEP(u, v)$  is done by the following way: (we make the tree rooted as in subsection 3.3.1)

- First, find the nearest common ancestor of  $u$  and  $v$  in constant time [15, 2], denote it by  $w$ .
- If  $w \neq u$ , then the desired  $x$  is actually the parent node of  $u$ , and we have identified the edge  $(u, x)$  (assuming that we store the downward edge for each node from its parent).
- If  $w = u$ , then it implies that  $u$  is an ancestor of  $v$ . In this case we can locate  $x$  by finding the ancestor of  $v$  on the depth of  $d(u) + 1$  in constant time using the method of “level ancestor query” [4, 3]. Also, the desired edge  $(u, x)$  is the edge which goes down from  $x$ 's parent (i.e.,  $u$ ) to  $x$ .

All the preprocessing algorithms used (nearest common ancestor[15, 2], level ancestor[4, 3], etc) are linear-time and linear-space.

## 4. $k$ -Depth Subtree Authentication

In this section, we will provide an efficient scheme for  $k$ -depth subtree authentication. The basic idea is to let Trent give the data distributor pre-signed signatures for every node of the tree, and then the distributor pre-aggregates some signatures to help perform fast online signature aggregation for the query subtrees. This basic idea also applies to our solution for  $k$ -radius subtree authentication that will be covered in Section 5.

### 4.1 Problem Definition

Assume that there is a rooted tree  $T = (V, E)$  where  $|V| = n$ . A message (i.e., a data item)  $M_v$  is associated with each node  $v \in V$ , and we assume that all those  $n$  messages are distinct<sup>3</sup>. Let  $d(v)$  denote the distance of node  $v$  from the root. Assume that the root is  $v_r$ , then we have  $d(v_r) = 0$ .

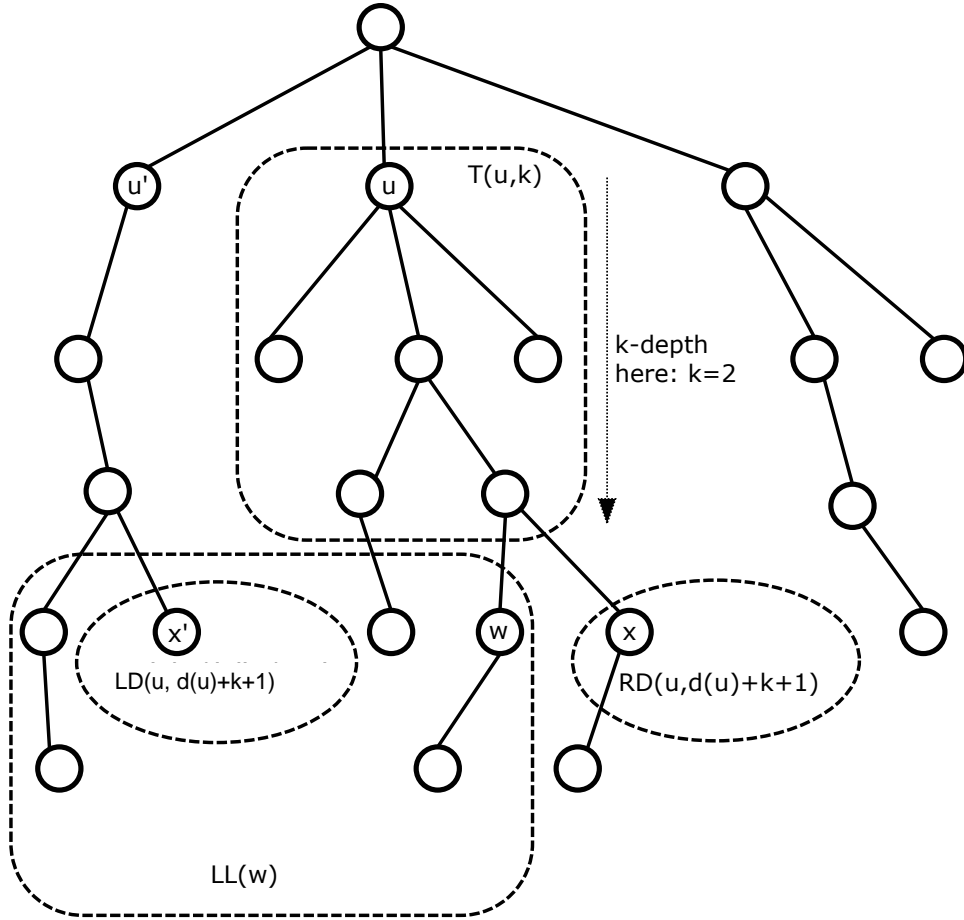
The third-party data distributor has the signature (signed once by the trusted authority) for each message:  $S(v)$  for all  $v \in V$  are given. These  $S(v)$ 's are the pre-signed  $IV$  items stored at the third-party data distributor. The task of  $k$ -depth subtree authentication is to design an efficient data structures for the following problem: give a node  $u \in V$  and a depth  $k \in \mathbb{N}$ , we want to efficiently aggregate the signatures for all the messages which are associated to

$$T(u, k) = \{v \in V \mid v \text{ is } u \text{ or a descendant of } u, \\ \text{and } d(u) \leq d(v) \leq d(u) + k.\}$$

---

<sup>3</sup>The distinctness can be achieved by appending location information (or any other unique keys) to the messages.

We will use  $T(u)$  to represent  $T(u, +\infty)$  for short, i.e.,  $T(u)$  is the full subtree rooted at  $u$ . See Figure 2 for examples. The aggregated signature  $S(T(u, k))$  is the *IV* item to send back to a user in response to the query parameters  $u$  and  $k$ .



**Figure 2. An example for  $k$ -depth subtree authentication problem.**

A naive solution for the third-party data distributor to aggregate the signatures for  $T(u, k)$  is to directly multiply all associated signatures:  $S(T(u, k)) \leftarrow \prod_{v \in T(u, k)} S(v)$ . But this approach is very time-consuming because the computational costs are  $O(|T(u, k)|)$ , which can be as bad as  $\Theta(n)$ .

#### 4.2 Linear Space Data Structures

In this subsection, we provide a linear space data structure to compute  $S(T(u, k))$  in constant time. The key idea is based on the following fact

$$T(u, k) = T(u) \setminus \bigcup_{v \in \mathcal{D}(u, k)} T(v), \tag{1}$$

where  $\mathcal{D}(u, k)$  is the set of all descendants of  $u$  on depth level  $d(u) + k + 1$ . Section 4.2.1 and 4.2.2 will develop the data structures to help aggregate  $\bigcup_{v \in \mathcal{D}(u, k)} T(v)$  efficiently.

### 4.2.1 Lower-Left Forest Aggregation

For each  $v \in V$ , we pre-aggregate the signature for  $T(v)$ . This can be done by a bottom-up aggregation in linear time for all  $v \in V$ . And then for every node, we order its child nodes from left to right, then we can define the “lower left” forest of a node  $w$  by the following way:

$$LL(w) = \bigcup_{v \in LSD(w)} T(v),$$

where  $LSD(w)$  is the set of all nodes which has the same depth as  $w$  but not to the right of  $w$  (see Figure 2 for example). We pre-aggregate the signatures of  $LL(w)$  for all  $w \in V$ . This can also be done in linear time by a bottom-up aggregation (or a postorder traversal of the tree) as follows:

- For each depth level  $l$ , we maintain  $LastVisit(l)$ , the last visited node on depth  $l$  during the bottom-up traversal. Initially,  $LastVisit(l) = \emptyset$ . After every visit of  $v$ , we set  $LastVisit(d(v)) \leftarrow v$ .
- We also mark down  $LS(v) \leftarrow LastVisit(d(v))$  during the visit of  $v$ , which represents the rightmost node among all the nodes that are: of depth  $d(v)$  and to the left of  $v$ . ( In the example tree of Figure 2, we have  $u' = LS(u)$  and  $w = LS(x)$ . )
- During the visit of  $v$ , we know that  $LS(v)$  is visited and assume that  $S(LL(LS(v)))$  has been computed, then based on the fact that  $LL(v) = LL(LS(v)) \cup T(v)$ , we have

$$S(LL(v)) = S(LL(LS(v))) \cdot S(T(v)).$$

Please note that we assume  $S(\emptyset) = 1$ .

### 4.2.2 Level Descendant Problem

For each node  $v$ , we set  $PostOrder(v)$  to be the order number (or say “time”) for the postorder traversal to visit (or say “leave”) the node  $v$ . For example,  $PostOrder(v_r) = n$  since the root is the last (and  $n^{\text{th}}$ ) visited node. Similarly, we set  $PreOrder(v)$  to be the order number when we do a preorder traversal of the tree.

For a node  $w$ , we define  $RD(w, l)$  to be the node that has the largest postorder number in the set

$$\{v \mid v \text{ is at depth level } l \text{ and } PostOrder(v) \leq PostOrder(w)\}.$$

Similarly, for a node  $w$ , we define  $LD(w, l)$  to be the node that has the largest preorder number in the set

$$\{v \mid v \text{ is at depth level } l \text{ and } PreOrder(v) < PreOrder(w)\}.$$

Please note that we need to handle the special cases, when  $LD(w, l)$  or  $RD(w, l)$  does not exist. The special cases are trivial and easy to handle. Also, please note that  $LD(w, l)$  or  $RD(w, l)$  does not need to be a descendant of  $w$ .

To efficiently compute  $RD(w, l)$  and  $LD(w, l)$ , we can reduce the problem to the Level Ancestor Problem [4, 3], which can be solved in constant time with linear-time/space preprocessing. The reduction use linear time and space in the preprocess stage, and constant time in the query stage. Here, we will give the reduction for computing  $RD(w, l)$ , and the reduction for  $LD(w, l)$  can be done in a similar way. See also the work of Ben-Amram [1] for another linear data structure for the level descendant problem.

We compute an auxiliary graph for the given tree as follow: for every node  $v$ , we link an arc from  $v$  to  $RD(v, d(v) + 1)$ . This can be done in linear time: during the postorder traversal of the tree, when we visit  $v$ ,

we link it to  $\text{LastVisit}(d(v) + 1)$ . See Section 4.2.1 for the definition of  $\text{LastVisit}(\cdot)$ . If  $\text{LastVisit}(d(v) + 1)$  is  $\emptyset$ , we treat it specially:  $\text{RD}(v, l)$  does not exist for  $l > d(v)$ .

The resulting graph is indeed a forest, because each node has only one out-going arc to a node at a lower level. The roots of the trees in the forest are  $\{v \mid \text{RD}(v, d(v) + 1) \text{ does not exist}\}$ . For cleaner presentations and without loss of generality, we only discuss the case when the auxiliary graph is a tree (denoted by  $T'$ ). Denote the root of  $T'$  by  $T'_r$ .

A useful property of the RD function is that: for any node  $v$  and  $d(v) \leq l_1 \leq l_2$ , we have

$$\text{RD}(v, l_2) = \text{RD}(\text{RD}(v, l_1), l_2),$$

when  $\text{RD}(v, l_2)$  exists. This implies an iterative method to compute  $\text{RD}(v, l)$  for any  $l \geq d(v)$ , as follows:

- First we compute  $v_1 = \text{RD}(v, d(v) + 1)$
- then  $v_2 = \text{RD}(v_1, d(v_1) + 1)$ , is  $\text{RD}(v, d(v) + 2)$
- we can get  $v_i = \text{RD}(v_{i-1}, d(v_{i-1}) + 1)$  for  $i \geq 2$ , and the calculated  $v_i$  is indeed  $\text{RD}(v, d(v) + i)$ .

The node  $\text{RD}(v, d(v) + 1)$  in tree  $T$  is the parent node of  $v$  in the auxiliary tree  $T'$ . So the above iterative method can be directly reduced to the level ancestor problem in  $T'$  as follows: to compute  $\text{RD}(w, l)$ , we need to find the ancestor of  $w$  in  $T'$  such that the ancestor is  $l - d(w)$  levels above  $w$  in  $T'$ . Here,  $d(w)$  is still calculated in  $T$ .

### 4.2.3 KDS Algorithm

The data structures above can help us online aggregate the signatures for  $T(u, k)$  efficiently. We process an online authentication request for parameters  $u$  and  $k$  by Algorithm 1. It is based on Equation (1) and

$$\bigcup_{v \in \mathcal{D}(u, k)} T(v) = \text{LL}(\text{RD}(u, d(u) + k + 1)) \setminus \text{LL}(\text{LD}(u, d(u) + k + 1)).$$

**Theorem 1** *There is a linear space authenticated data structure for the  $k$ -depth subtree authentication problem to online aggregate the signatures in  $O(1)$  time. The time complexity for pre-computing the data structures is linear. The size of the resulting signature is  $O(1)$ .*

## 5. $k$ -Radius Subtree Authentication

In this section, we will give an efficient authentication scheme for  $k$ -radius subtrees.

### 5.1 Problem Definition

Assume that there is an unrooted tree  $T = (V, E)$ . We associate a message  $M_v$  to each node  $v \in V$ , and assume that the messages are distinct. Given  $u \in V$  and  $k \in \mathbb{N}$ , a  $k$ -radius subtree of  $T$  consists of all the nodes that are within a distance of  $k$  from  $u$ . We denote such a subtree by  $U_T(u, k)$ , i.e.,

$$U_T(u, k) = \{v \in T \mid d(u, v) \leq k\},$$

where  $d(u, v)$  denotes the distance between  $u$  and  $v$  in the tree. The problem of  $k$ -radius subtree authentication is to aggregate the signatures for  $U_T(u, k)$  when  $u$  and  $k$  are given online, i.e., the third-party data distributor can send back the short aggregate signature as the  $IV$  item in response to the query parameters  $u$  and  $k$ .

---

**Algorithm 1** An algorithm for  $k$ -depth subtree authentication

---

**Function:**  $\text{KDS}(u, k)$ **Input:**  $u, k$ **Output:** return  $S(T(u, k))$ 

- 1: set  $\sigma_1 \leftarrow S(T(u))$
  - 2: let  $x \leftarrow \text{RD}(u, d(u) + k + 1)$
  - 3: **if**  $x$  does not exist **then**
  - 4:   return  $\sigma_1$  // in this case,  $T(u, k) = T(u)$
  - 5: **end if**
  - 6: set  $\sigma_2 \leftarrow S(\text{LL}(x))$
  - 7: let  $x' = \text{LD}(u, d(u) + k + 1)$
  - 8: **if**  $x'$  does not exist **then**
  - 9:    $\sigma_3 \leftarrow 1$ .
  - 10: **else**
  - 11:    $\sigma_3 \leftarrow S(\text{LL}(x'))$
  - 12: **end if**
  - 13: return  $\sigma_1(\sigma_2\sigma_3^{-1})^{-1} = \sigma_1\sigma_2^{-1}\sigma_3$
- 

## 5.2 Centroid Decomposition

In this subsection, we will briefly describe how to preprocess a tree  $T$  by the centroid decomposition method [7, 13]. The result of tree centroid decomposition can help to transform a  $k$ -radius subtree into a small set of subtrees whose signatures are pre-aggregated.

A node  $x$  in a tree  $T$  is called a centroid of  $T$  if the removal of  $x$  will make all the rest connected components have size no greater than  $|T|/2$ . A tree may have at most two centroids, and if there are two then one must be a neighbor of the other [14, 17]. Throughout this paper, we specify the centroid of a tree to be the one whose numbering is lexicographically smaller (i.e., we number the nodes from 1 to  $n$ ). There exists a linear time algorithm to compute the centroid of a tree due to the work of Goldman [10]. We use  $\text{CT}(T)$  to denote the centroid of  $T$  found by the linear time algorithm.

Algorithm 2 is the well-known recursive tree centroid decomposition method [7, 13]: (see Figure 3 for an example of the tree centroid decomposition)

---

**Algorithm 2** Tree Centroid Decomposition

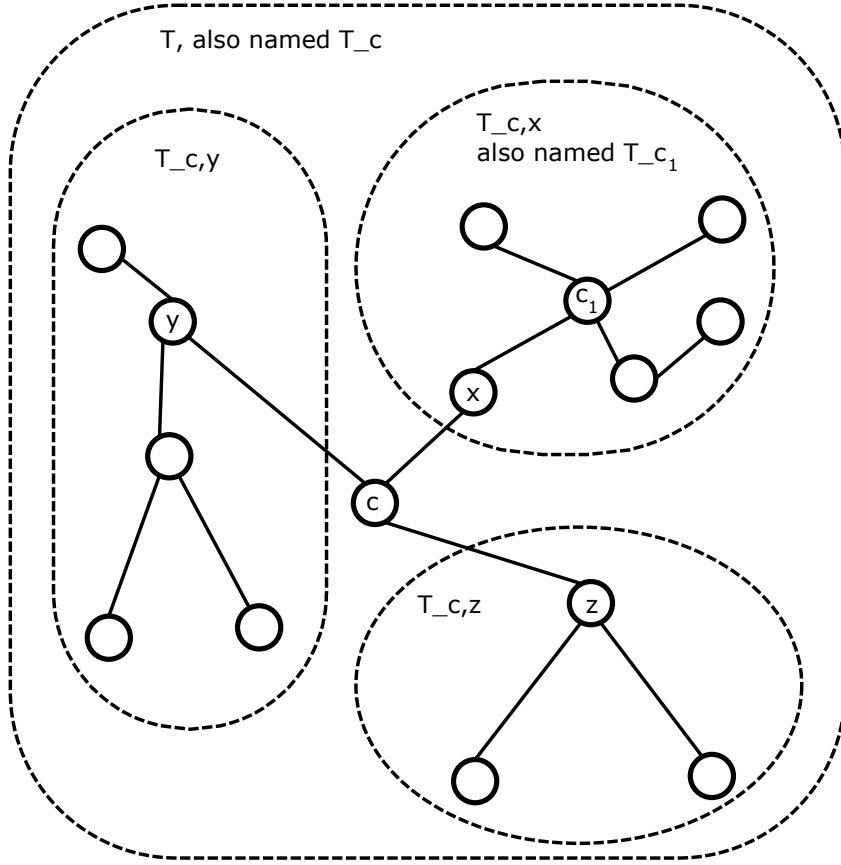
---

**Function:**  $\text{CentroidDecomposition}(T)$ 

- 1: Find the centroid of  $T$ . Denote the set of the rest connected components by  $\text{Rest}(T) = \{T' \mid T' \text{ is a connected component after the removal of } \text{CT}(T)\}$ .
  - 2: Let  $c = \text{CT}(T)$  be the computed centroid, then for each neighbor  $x$  of  $c$  in  $T$ , we use  $T_{c,x}$  to denote the rest connected component that contains  $x$ . Also, we denote the current tree  $T$  by  $T_c$ .
  - 3: Recursively call  $\text{CentroidDecomposition}(T')$  for each  $T' \in \text{Rest}(T)$ .
- 

The time complexity for the above recursive tree centroid decomposition scheme is  $O(n \log n)$ , since no node will participate in the centroid computations for more than  $O(\log n)$  times.

The stack space for the recursion is bounded by  $O(n + n/2 + n/4 + n/8 + \dots) = O(n)$ . To space-efficiently store the tree centroid decomposition, we do not store the whole description of  $T_{c,x}$  for every such subtree (otherwise



**Figure 3. An example for the tree centroid decomposition. In this example, node  $c$  is a centroid of the whole tree, and  $c_1$  is the centroid of the subtree  $T_{c,x}$ .**

the storage space can be as bad as  $\Omega(n \log n)$ ). Instead, we only store the centroid node  $CT(T_{c,x})$  for each  $T_{c,x}$ , and associate that centroid node with the edge  $(c, x)$ . Also, we do not store the description of  $T_c$ . So the space we use for storing the centroid decomposition is linear.

### 5.3 $O(\log n)$ -Time Query Answering

In this subsection, we will give an  $O(n \log n)$ -space authenticated data structure for online aggregating the signatures for any  $k$ -radius subtree in  $O(\log n)$  time.

Define canonical subtrees to be all the subtrees considered during the recursive call of Algorithm 2 if we start the recursion at  $T$ , i.e., the canonical subtrees are  $\{T_c \mid c \in V\}$ . Please note that, each node  $c \in V$  must be a centroid of some canonical subtree; an extreme case is when node  $c$  is the centroid of a subtree which only consists of a single node ( $c$  itself). Therefore, there are exactly  $n$  such canonical subtrees, and one can see that each “rested connected component”  $T_{c,x}$  is just  $T_{CT(T_{c,x})}$ . Based on the fact that no node will be in more than

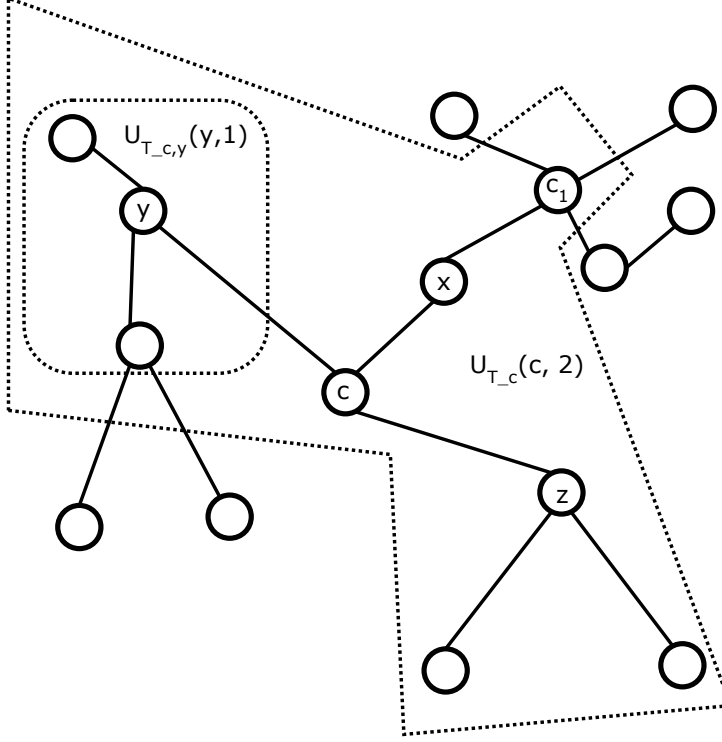


Figure 4. An example for the notations of  $U_{T_c}(c, j)$  and  $U_{T_{c,x}}(x, j)$ .

$O(\log n)$  canonical subtrees, we have

$$\sum_{c \in V} |T_c| = O(n \log n); \quad (2)$$

$$\sum_{c \in V} \sum_{(c,x) \in T_c} |T_{c,x}| = O(n \log n). \quad (3)$$

We pre-aggregate the signatures for the following subtrees (see Figure 4 for examples):

$$U_{T_c}(c, j) \quad \text{for all } 0 \leq j \leq \max_{v \in T_c} d(c, v);$$

$$U_{T_{c,x}}(x, j) \quad \text{for all } 0 \leq j \leq \max_{v \in T_{c,x}} d(x, v).$$

To do that, we can simply build the authenticated data structures from section 4 for all  $T_c$  (rooted at  $c$ ) and  $T_{c,x}$  (rooted at  $x$ ), and the total space complexity is  $O(n \log n)$  because of Equation (2) and (3). After the data structures are built, Algorithm 3 can be used to calculate  $S(U_T(u, k))$  in  $O(\log n)$  time: just call  $\text{KRS}(c_T; u, k)$  where  $c_T$  is a pre-computed centroid for the tree  $T$ .

The analysis of Algorithm 3 is as follows

- The interface  $\text{KRS}(c; u, k)$  means: for input parameters  $c \in V$ ,  $u \in T_c$  and  $k \in \mathbb{N}$ , the function needs to return the aggregate signature for the  $k$ -radius subtree centering at  $u$  and restricted within  $T_c$ .
- Line 1-3: If  $u$  is the centroid  $c$  of  $T_c$ , then we can see that  $U_{T_c}(u, k)$  is actually  $U_{T_c}(c, k)$ , and it's already pre-computed.

---

**Algorithm 3** An algorithm for  $k$ -radius subtree authentication

---

**Function:**  $\text{KRS}(c; u, k)$ **Output:** return  $S(U_{T_c}(u, k))$ 

```
1: if  $u = c$  then
2:   return  $S(U_{T_c}(c, k))$ 
3: end if
4: find the node  $x$  and the related edge  $(c, x)$ 
   so that  $T_{c,x}$  contains  $u$ ;
5: if  $c \in U_{T_c}(u, k)$  then
6:    $\sigma_1 \leftarrow S(U_{T_c}(c, k - d(c, u)))$ 
7:    $\sigma_2 \leftarrow S(U_{T_{c,x}}(x, k - d(c, u) - 1))$ 
8:    $\sigma_3 \leftarrow \text{KRS}(\text{CT}(T_{c,x}); u, k)$ 
9:   return  $\sigma_1 \sigma_2^{-1} \sigma_3$ 
10: else
11:   return  $\text{KRS}(\text{CT}(T_{c,x}); u, k)$ 
12: end if
```

---

- Line 4: This is done by setting  $(c, x) \leftarrow \text{NEP}(c, u)$  in constant time (see subsection 3.3.2).
- Line 5: The testing of  $c \in U_{T_c}(u, k)$  can be done by checking whether  $d(c, u) \leq k$  in constant time (see subsection 3.3.1).
- Line 6-9: In the case that  $c \in U_{T_c}(u, k)$ , we try to split the query subtree into two parts: one part is in  $T_{c,x}$ , and the other part is the rest. For the part in  $T_{c,x}$ , an aggregate signature is computed by calling  $\text{KRS}(\text{CT}(T_{c,x}); u, k)$  as in Line 8, and remember that  $\text{CT}(T_{c,x})$  is already pre-computed (associated with  $(c, x)$ ) as described in the previous subsection. For the rest part that is cut by  $(c, x)$  (and outside of  $T_{c,x}$ ), we can see that it is actually

$$\begin{aligned} & U_{T_c}(u, k) \setminus T_{c,x} \\ &= U_{T_c}(c, k - d(c, u)) \setminus T_{c,x} \\ &= U_{T_c}(c, k - d(c, u)) \setminus U_{T_{c,x}}(x, k - d(c, u) - 1), \end{aligned}$$

where

$$U_{T_{c,x}}(x, k - d(c, u) - 1) \subseteq U_{T_c}(c, k - d(c, u)).$$

Therefore, we have Line 9 to compute the aggregate signature.

- Line 11: In the case that  $c \notin U_{T_c}(u, k)$ , then we can conclude that  $U_{T_c}(u, k)$  is completely within the branching subtree  $T_{c,x}$ , so we recursive call  $\text{KRS}(\text{CT}(T_{c,x}); u, k)$ .

Since all the operations within a call is constant, and the depth of the recursive calls are at most  $O(\log n)$ , we have the following result.

**Theorem 2** *There is an  $O(n \log n)$ -space authenticated data structure for the  $k$ -radius subtree authentication problem to online aggregate the signatures in  $O(\log n)$  time. The time complexity for pre-computing the data structures is  $O(n \log n)$ . The size of the resulting signature is  $O(1)$ .*

## 6. Conclusions and Future Work

We gave an efficient solution to the problem of data authentication in a distributed setting in which the trusted data owner uses untrusted servers to provide data distribution and/or authentication services to users. Our solution is for the case where the  $n$  data items are organized as a tree hierarchy  $T$ , and where a user's access set is defined as a subtree of  $T$  using two natural notions of proximity to a node of the hierarchy. For the specific problems considered, the specific communication bounds that we achieve do not leave room for improvement and the computation bounds are almost optimal (constant for  $k$ -depth subtree authentication;  $O(\log(n))$  for  $k$ -radius subtree authentication). Still, there is much room for future work, including: (i) making the scheme dynamic (it currently assumes a static hierarchy); and (ii) extending the scheme to non-tree hierarchies.

## Acknowledgements.

Portions of this work were supported by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

## References

- [1] A. M. Ben-Amram. The Euler path to static level-ancestors. *Unpublished manuscript*. arXiv:0909.1030v1 [cs.DS] Available at <http://arxiv.org/abs/0909.1030>.
- [2] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, 2000.
- [3] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004.
- [4] O. Berkman and U. Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994.
- [5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Advances in Cryptology – Eurocrypt’03*, pages 416–432, 2003.
- [6] D. Boneh, I. Mironov, and V. Shoup. A secure signature scheme from bilinear maps. In *Topics in Cryptology – CT-RSA 2003*, pages 98–110.
- [7] B. Chazelle. A theorem on polygon cutting with applications. In *FOCS ’82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 339–349, Washington, DC, USA, 1982. IEEE Computer Society.
- [8] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11(3):291–314, 2003.
- [9] P. T. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of XML documents. In *ACM Conference on Computer and Communications Security*, pages 136–145, 2001.
- [10] A. Goldman. Optimal center location in a simple network. *Transportation Science*, 5:212–221, 1971.
- [11] M. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. *Technical Report, Johns Hopkins Information Security Institute*, 2000.
- [12] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proceedings of RSA Conference – Cryptographers’ Track*, volume 2612, pages 295–313. Springer, LNCS, 2003.
- [13] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *SCG ’86: Proceedings of the second annual symposium on Computational geometry*, pages 1–13, New York, NY, USA, 1986. ACM.
- [14] S. Hakimi. Optimum locations of switching center and the absolute center and medians of a graph. *Operations Research*, 12:450–459, 1964.
- [15] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
- [16] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of ACM Electronic Commerce (EC)*, October 2001.
- [17] D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

- [18] P. C. Kocher. On certificate revocation and validation. In *Proceedings of International Conference on Financial Cryptography*, volume 1465. LNCS, Springer-Verlag, 1998.
- [19] L. Lamport. Constructing digital signatures from a one-way function. In *Technical report, SRI-CSL-98*, pages 28–37, SRI International Computer Science Laboratory, October 1979.
- [20] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authentic data publication. *Algorithmica*, 39(1), 2001.
- [21] R. C. Merkle. A certified digital signature. In *Advances in Cryptography - Annual International Cryptography Conference*, volume 435, pages 218–238, 1989.
- [22] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Proceedings of ISOC Symposium on Network and Distributed Systems Security (NDSS'04)*, 2004.
- [23] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.
- [24] M. Narasimha and G. Tsudik. DSAC: integrity for outsourced databases with signature aggregation and chaining. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 235–236, 2005.
- [25] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings ACM Conference on Computer and Communications Security (CCS)*, pages 28–37, Philadelphia, PA, November 2001.
- [26] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. *Proceedings Network and Distributed System Security Symposium, (NDSS)*, February 2001.
- [27] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *Proceedings ACM Conference on Computer and Communications Security (CCS)*, pages 93–100, Philadelphia, PA, November 2001.