# Efficient Content Authentication in Peer-to-peer Networks[⋆]

**– Extended Abstract[⋆⋆]–**

Roberto Tamassia[1] and Nikos Triandopoulos[2]

[1] Department of Computer Science, Brown University
[2] Institute for Security Technology Studies, Dartmouth College

**Abstract.** We study a new model for data authentication over peer-to-peer (p2p) storage networks, where data items are stored, queried and authenticated in a totally decentralized fashion. The model captures the security requirements of emerging distributed computing applications. We present an efficient construction of a *distributed Merkle tree* (DMT), which realizes an authentication tree over a p2p network, thus extending a fundamental cryptographic technique to distributed environments. We show how our DMT can be used to design an *authenticated distributed hash table* that is secure against replay attacks and consistent with the update history. Our scheme is built on top of a broad class of existing p2p overlay networks and achieves generality by using only the basic functionality of object location. We use this scheme to design the first efficient *distributed authenticated dictionary*.

## 1 Introduction

Peer-to-peer (p2p) networks provide the basis for the design of fully decentralized systems, where data and computing resources are shared among participating peers. Properties of p2p networks include scalability, self-stabilization, data availability, load balancing, and efficient searching. As p2p networks become more mature and established and new applications emerge for them, the need increases for their security.

In this paper, we study data authentication in p2p networks, where a data set originated at a trusted source is shared and dispersed over the nodes of a p2p network and is queried and retrieved by users through the API exported by the network. We focus our study on the basic put-get functionality that is realized by any distributed data structure built over overlay p2p networks, including the

[⋆] Research supported in part by IAM Technology, Inc., the National Science Foundation under grants IIS–0324846 and CCF–0311510, and the Institute for Information Infrastructure Protection (I3P) under an award from the Science and Technology Directorate at the Department of Homeland Security. Points of view or opinions in this document are those of the authors and do not represent the official position or policies of the U.S. Department of Homeland Security.

[⋆⋆] To appear at the 2007 International Conference on Applied Cryptography and Network Security (ACNS '07).

important class of *distributed hash tables* (DHTs) (e.g., [31, 36]). In particular, we are interested in guarding users against malfunctioning or malicious network nodes that incorrectly execute get and put operations.

Current authentication techniques for data stored in p2p networks are static and centralized. Also, several authentication methods based on signatures on a per-object basis are vulnerable to *replay attacks*, where an old, out-of-date or invalid, data object is returned as the answer to a get operation.

In this paper, we introduce a new model for *distributed data authentication* in p2p networks and present an efficient realization of this model for securely performing dictionary operations on a p2p network. Our authentication structure and protocol are resilient against replay attacks and extend the functionality of p2p networks by supporting authenticated versions of operations put, get and remove, thus providing a transparent security layer to higher-level p2p applications.

By using only the basic operation of object location, our technique achieves generality and can be applied to a broad class of p2p architectures (e.g., existing DHT implementations). Our authentication scheme is based on the design of an efficient *distributed Merkle tree (DMT)*—the first distributed version of Merkle's authentication tree [23]. Thus, our construction can serve as a general-purpose authentication structure for decentralized computing architectures with minimal trust assumptions.

## 1.1 Motivation

Data storage and retrieval are essential tasks in p2p systems, where large data collections (e.g., documents, media files, database records) are shared over a network among participating peers. An important security problem in p2p system is data authentication in the presence of faulty or malicious network nodes. For instance, adversarial network nodes may wish to degrade the performance of a p2p storage system by providing false responses to queries.

We wish to ensure the integrity of shared data and to provide cryptographically sound techniques that allow a user to verify that retrieved data from the system is authentic and unaltered. Moreover, in a dynamic setting, where data evolve over time through updates, we want to also ensure that data items retrieved by queries have the most up-to-date versions. We consider the standard query model in p2p storage systems, where data items are stored as key-value pairs of the type $(k, x)$ (keys are unique identifiers and values are associated with keys) and managed through operation put$(k, x)$ (which inserts a new pair in the system) and query get$(k)$ (which returns the value associated with key $k$). This is the core functionality exported by distributed data structures.

Assuming an established PKI, a straightforward approach to authentication is to individually sign each data item stored in the p2p system: when the data source wishes to add $(k, x)$, it computes the signature $\sigma$ of pair $(k, x)$ using its private key and inserts $(k, (\sigma, x))$ into the data structure. A query for key $k$ now returns the pair $(\sigma, x)$, where $\sigma$ allows the user to verify whether $x$ is the valid answer. However, this "sign-all" approach is vulnerable to *replay attacks* for old values because it does not provide any mechanism for invalidating signatures on currently invalid pairs, such as pairs that have expired, were removed from the p2p system, or whose values have been modified. Therefore, in response to operation get($k$), a malicious network node can return an invalid (old or out-of-date) value that is still verifiable. Note that most p2p systems do not support explicit item deletion; e.g., DHTs only keep a soft state, where data items expire after a time interval and are removed from the system (thus, to maintain these items, the source has to reinsert them). Nevertheless, even when some form of item deletion is supported, replay attacks are still possible: invalid signed pairs can simply be cached and never deleted. In general, we need a mechanism ensuring that only recent signatures are used to validate answers to queries and that deletions are correctly handled by the system.

Replay attacks can be prevented by introducing time-stamps in the signed values and a validity period for the signature, called *time quantum* [26]. However, this extension of the "sign-all" approach incurs a significant computational overhead: after each time quantum, each of all the valid pairs that currently reside in the system need to be retrieved, resigned by data source and then reinserted in the system. Thus, it is preferable to maintain at all times a *global authentication state* of the system that includes only the currently valid data items and essentially authenticates that data is properly updated. This is achieved by *signature amortization*, the state-of-the-art technique for dynamic data authentication, where a data source signs only one digest (short cryptographic description) of the entire collection of (valid) stored data items owned by this source. The canonical method for amortizing one signature over a large data set is Merkle's authentication tree [23]; however, there is currently no distributed implementation of this scheme. Existing p2p storage systems and DHT implementations that support an authentication service for the stored data are all using "sign-all" techniques. Thus, a replay attack is feasible for malicious network nodes; and if, instead, signature refreshing is used to solve the problem, this leads to inefficient and impractical authentication schemes.

### 1.2 Related Work

**Merkle tree.** The Merkle tree [23] is a simple and widely-used cryptographic construction for efficiently certifying set membership. The idea is to use a balanced tree and a cryptographic collision-resistant hash function (e.g., SHA-1) to produce a short cryptographic description of a large data set. Elements of the set are stored at the tree leaves and internal nodes store the result of applying the hash function to the concatenation of the values stored at the children nodes. The root value is signed and, when verified, the collision-resistant property propagates authentication from the root to the leaves. Certifying that an element is in the set is performed by using a *verification path* to recompute the authentic root value. This path consists (of the hash values) of the siblings of the nodes in the path from the leaf associated with the element to the root. Updates in the Merkle tree are handled with complexity proportional to the height of the tree (e.g., [26]). An extension to the symmetric-key setting is given in [12], where it is shown that verification along a path can be performed in parallel. No distributed implementation for Merkle trees currently exists.

**Authenticated data structures.** Authenticated data structures (ADSs) (see, e.g., [37, 11, 21, 26]) use a three-party model where data created by a trusted data source is replicated at several untrusted responders that answer query from users on behalf of the source. Signature amortization is used, similarly to the Merkle tree, but specially designed according to the supported query type. A significant amount of work has been done on developing efficient authenticated data structures for various type of queries (e.g., [6, 11, 21, 5, 1, 38]). The related model of *outsourced database* (ODB) systems studies the special case where SQL queries (essentially, range queries over indexes) are issued over databases published at remote sites (e.g., [18, 25, 24, 28]). Both models involve servers that keep a copy of the entire data set. Thus, they do not capture the architecture of p2p networks, where data is shared and distributed on a per-item basis.

**Distributed hash tables.** Distributed hash tables (DHTs) are a popular class of p2p storage networks that support the dictionary functionality (e.g., [36, 16, 33, 34, 20, 40, 29, 31]). Using distributed routing techniques over an overlay network, a DHT can locate the value associated with a query key with $O(\log n)$ expected communication steps, where $n$ is the number of participating nodes, each maintaining $O(\log n)$ routing information. Based on DHTs, several practical distributed storage systems over p2p networks have been developed that support efficient retrieval (e.g., [7, 31, 4, 14, 32]). Other distributed data structures with similar efficiency provide more elaborate functionalities over p2p networks; for instance, skip-graphs [2] and their extensions (e.g., [13, 10]) support searching over ordered keys.

**Trees over p2p overlay networks.** The development of DHTs was followed by the design of various search and aggregation trees built over DHTs or other type of distributed trees (e.g., [15, 19, 30]). However, these trees can neither be used to implement a distributed Merkle tree nor meet the requirements for efficient data authentication over p2p networks, since these constructions are static or correspond to special-purpose search trees inappropriate to efficiently realize an authentication tree—which is sensitive to node losses or structural changes because of the use of the cryptographic hash function.

**Security in p2p systems.** Security issues related to p2p systems are discussed in [35, 39], where the authentication problem is treated simply using per-object signatures. Although with respect to routing and searching, numerous DHTs have been shown to tolerate significant network-node failures—random (e.g., [36, 16, 33, 31]) or malicious (e.g., [3, 8, 27])—data authentication has not been systematically studied in p2p networks. Existing p2p storage systems (e.g., [4, 7, 29, 31, 32]) support an elementary authentication service for retrieved data which is of the "sign-all" type, where stored contents are individually signed by their source. Often, authentication involves the so-called *self-certified data* [9], where large data items (e.g., a file system) get partitioned into blocks, which are stored as separate objects in the system and are bound together using collision-resistant hashing in some tree-like hierarchy, and where the root-block is signed.

Although this technique resembles a Merkle tree, it only implements signature amortization among a large item and not among all data items owned by a source, which are still separately signed. Additionally, this authentication structure is static (no updates are supported) and, generally, unbalanced (e.g., parts of file systems can be flat and other can be extremely skewed). Overall, currently used authentication solutions are vulnerable to replay attacks (even if item removal is supported, as, e.g., in [32]) and lack efficiency for supporting signature refreshing and updates. Finally, privacy and anonymity issues or other security issues (e.g., the Sybil attack) related to p2p systems have been studied.

## 1.3 Paper Outline and Our Contributions

In Section 2, we introduce a new model for distributed data authentication over p2p networks, where management and retrieval of shared data resources are totally decentralized, yet *cryptographically verifiable* by the interested parties. In this model, data objects that are originated at a trusted source become available to users through an untrusted p2p storage network and authentication protocols guarantee the correct functionality of the underlying storage system. We present an efficient realization of this model for the basic dictionary operations performed on a dynamic set of data objects and describe data authentication protocols that allow users to verify the integrity of the data objects retrieved by the network and allow the source to verify the integrity of updates executed by the network.

The main idea behind our security solution is conceptually simple: we use an authentication tree to produce a cryptographic commitment of the stored data set, against which any update or query operation is checked for correctness. Implementing this idea in a dynamic distributed environment entails certain challenges. First, we design a balanced tree-like authentication structure that can be dispersed among network nodes and, at the same time, provide efficient retrieval of verification paths and allow efficient structural adjustments after updates. Additionally, we ensure that the commitment is updated correctly after any changes on the data set, even in the presence of malicious network nodes.

In Section 3, we present our main result, the first efficient scheme for implementing a fully dynamic *distributed Merkle tree* (DMT), using only the object-location functionality exported by any p2p system or DHT. Our scheme has certain properties that allow its efficient distribution over a p2p network and is designed to support locality for answer verification and facilitate the use of caching, thus achieving efficiency and resilience against malicious nodes. Moreover, balance is maintained at low cost over the course of updates on the tree. We analyze its performance and compare it with naive implementations. Our scheme, designed for both bottom-up and top-down access, constitutes a new, general-purpose, dynamic distributed tree.

In Section 4 and based on our DMT construction, we realize an efficient *authenticated distributed hash table* (ADHT), which extends DHTs in various ways. In particular, we show how our DMT can be extended to an authentication structure that provides authenticated and efficient versions of operations get, put, and also operation remove, supporting *authenticated deletions*, the first of this type. We compare ADHT with the "sign-all" solution. Our ADHT provides efficient distributed storage, secure against replay attacks and consistent with the update history. Our ADHT can in turn support a more general data authentication scheme for dictionary operations. In particular, we present the first efficient *distributed authenticated dictionary*. In a totally distributed setting over a p2p network with $n$ nodes and using only the basic object-location operation, we show how to authenticate membership queries in a fully dynamic set of $m$ data elements in $O(\log n \log m)$ time using $O(m \log m)$ storage, with similar complexities for supporting updates.

In Table 1, we summarize the comparison of our work with existing methods for distributed data authentication. Our scheme is the first to provide secure and efficient data authentication in totally decentralized environments over p2p networks. We conclude and discuss future work in Section 5.

**Table 1.** Qualitative comparison of our method with existing authentication models.

| | decentralized | replay-safe | efficient |
|---|---|---|---|
| "Sign-all" method in p2p networks | ● | | ● |
| "Sign-all" method in p2p networks & timestamps | ● | ● | |
| ADS & ODB data authentication models | | ● | ● |
| **Our results** | ● | ● | ● |

This extended abstract omits many details of this work, which will appear in the full version of the paper.

## 2 Authentication Model

We introduce a new model for distributed data authentication, where data items are stored, queried and authenticated in a totally decentralized fashion. This model captures fundamental security requirements that arise in p2p distributed storage systems. The model consists of:

– a trusted *data source* $S$, originator of a dynamic data set $D$;
– an untrusted distributed *p2p network* $\mathcal{N}$ that exports a specific functionality for storing, accessing and retrieving shared data resources; the nodes of $\mathcal{N}$ distributively store set $D$ and answer queries about $D$ on behalf of source $S$; $D$ evolves over time through updates submitted by $S$ to network $\mathcal{N}$; and
– *users* who issue queries about $D$ by accessing any node of network $\mathcal{N}$.

In our model, network nodes are untrusted and can exhibit adversarial behavior in updates submitted by the source or queries posed by a user. A network node responsible for an update of $D$ may maliciously fail to perform the update and cause the p2p system to become inconsistent with the sequence of updates issued by $S$ (e.g., $\mathcal{N}$ attempts an "universe-split" attack, where two different states of set $D$ are represented in the system). Also, a network node responsible for a query on $D$ may maliciously falsify the returned answer (e.g., $\mathcal{N}$ attempts to compromise data integrity or launch a replay attack).

To guard the source and the users against attacks, our goal is to design an *authentication scheme*, a set of authentication structures and corresponding protocols that augment the functionality of network $\mathcal{N}$ with verification features. In particular, the scheme allows the source to check that an update has been correctly performed by $\mathcal{N}$, according to the history of previously performed updates. The scheme also allows a user to verify that the answer to a query is authentic, as if it was coming directly from $S$.

An authentication scheme should be *secure*, informally meaning that the verification protocols reliably characterize the behavior of the network. Security, implies that when the p2p system is not under attack, then any update or query operation always passes the verification test (*completeness*); and, for any polynomial-time (on some security parameter) adversary that controls $\mathcal{N}$ and observes a chosen history of polynomial size of updates on $D$, succeeding in falsifying the verification test for an update or query is an event of negligible (on the security parameter) probability (*soundness*). Note that the above notion of security includes safety against replay attacks.

An authentication scheme should also be *decentralized*. Protocols and data-management procedures that are associated with data authentication should be distributed, designed as much as possible in accordance with the underlying p2p architecture. Or else, a security solution in a decentralized setting would be centralized, which would automatically diminish the advantages of the system.

An authentication scheme should finally be *efficient*, imposing low computational, communication and storage overhead to the parties participating in the protocols and network $\mathcal{N}$. Cost parameters that should be minimized are: the *storage cost*, the amount of authentication information stored at $S$ and $\mathcal{N}$ or needed by a user to verify an answer; the *update* and *query costs*, the computational and communication costs incurred due to authentication at $\mathcal{N}$ after updates and queries on data set $D$; and the *verification cost*, the computational cost incurred by $S$ or a user to verify the correctness of an update or query.

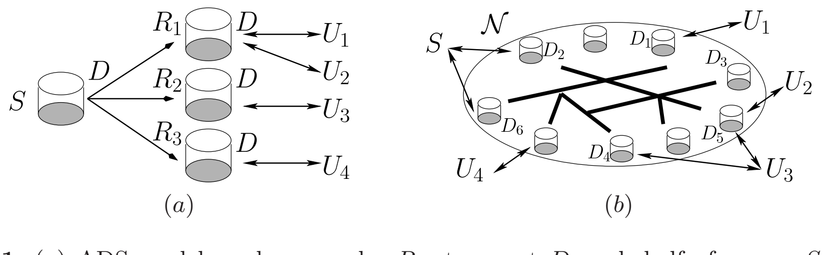

**Fig. 1.** $(a)$ ADS model: each responder $R_i$ stores set $D$ on behalf of source $S$ and answers users' queries. $(b)$ Distributed authentication: $D$ is dispersed as $D_1, D_2, \ldots$ over p2p network $\mathcal{N}$; updates and queries are performed by contacting any node of $\mathcal{N}$.

Our model drastically differs from those of authenticated data structures (ADS) (see Figure 1) and outsourced database systems (ODB) in that it is inherently decentralized. Data and authentication information are distributed over a p2p storage network at the data-item level and they are accessed by the source and the users through the interface of the network by contacting any of its nodes. In contrast, ADS model involves data replication at remote responder-servers, thus data distribution occurs only at the data-set level, and ODB model involves data outsourcing to a designated server, thus adopting centralized data management. Thus, we extend the client-server model of data authentication to a distributed authentication model that operates over any p2p network.

In developing an authentication scheme, it is desirable that as few as possible assumptions are made about network $\mathcal{N}$. Ideally, the underlying network should be any structured p2p network, for instance any DHT. By designing an authentication scheme using popular and well-studied distributed data structures, we leverage a broad class of existing p2p architectures, thus providing p2p systems with a transparent security layer at the application level. In this paper, we follow this principle and achieve generality by building our authentication scheme over the primitive search operation locate, which returns the network node corresponding to a given abstract object identifier. Since our constructions do not depend on the details of the p2p system implementation, we gain simplicity, extensibility and usability.

In what follows, we denote with $\mathcal{N}$ a DHT over which we wish to build an authentication scheme. Note that the scheme inherits the following properties shared by most DHT implementations: (1) a DHT with $n$ network nodes uses $O(\log n)$ storage per node and performs a locate operation (also, put and get) in $O(\log n)$ network hops (node-to-node communication steps) with high probability; (2) node additions, deletions, and failures are handled dynamically through a distributed algorithm that incrementally updates the routing information; (3) some form of redundancy is used, which replicates data objects to a constant number of neighboring nodes so that node failures are tolerated also with respect to data recovery; and (4) caching techniques are used to improve data retrieval.

Finally, we consider all other types of misbehavior by network nodes (e.g., against routing or the DHT functionality) to be denial-of-service attacks, a distinct or orthogonal problem to data authentication. Of course, these attacks can also limit the functionality of the authentication scheme; but they will not compromise its security. Actually, since our authentication scheme is agnostic of the implementation of the underlying DHT, we can strengthen the resilience against malicious nodes in our model by using a specific DHT that tolerates certain DoS attacks (e.g, a DHT that authenticates routing information).

Our authentication scheme implements the signature amortization technique. In particular, we assume that the users of the system know and trust the public key of the source $S$. Using Merkle's authentication tree, $S$ maintains at all times a digest of the data set. Queries are authenticated in the standard way: along with the answer, $\mathcal{N}$ returns the digest signed by $S$ and a proof linking the answer to the digest. Updates are authenticated by using the digest to check that the current state of the data set is consistent with the update history.

## 3 An Efficient Distributed Merkle Tree

In this section, we present a distributed Merkle tree (DMT), an efficient implementation of an authentication tree built over a p2p network realizing a DHT. This is the core construction in our authentication scheme and a result of independent interest: since numerous security protocols and cryptographic constructions are based on Merkle's tree, a DMT yields distributed versions of such protocols and constructions. We first discuss our design goals.

Built on a data set $D$, the distributed authentication tree should be dynamic, allowing for efficient hash updates after changes in $D$. It should also be balanced, providing verification paths (membership proofs) of size that is logarithmic in $|D|$, and its height balance should be efficiently maintainable after updates. In order to optimize its verification properties and further improve its usability, we are particularly interested in facilitating the construction (location) of the verification paths of the DMT; that is, the verification path of any data item should be retrieved by the network as fast as possible. All the above should be implemented in a distributed way, using only the locate operation, which is provided by the network and takes $O(\log n)$ time for a network of $n$ nodes. Accordingly, cost parameters we wish to minimize are: the *path location cost*, the cost for constructing a verification path (proof), the *update cost*, the cost for maintaining the authentication structure after updates on $D$, and the *storage cost*. Both the location and update costs each consists of (1) processing cost, i.e., computational cost for the participating nodes in the system, and (2) communication cost, i.e., cost of locate operations or direct communications between nodes.

To gain some intuition behind our construction, assume that set $\{x_1, \dots, x_n\}$ is distributed over the network (each element stored at a unique network node) and consider a Merkle tree computed over this set. All hash values in the tree need be distributed in the network. The first problem to consider is how the hash values are indexed, i.e., with which keys they are stored in the system. The hash value is a value that is unknown to network nodes, thus the value itself cannot be used as a key. Additionally, for immediate location of verification paths, network node storing element $x_i$ should also store information about the verification path that corresponds to $x_i$. We briefly describe two less efficient approaches for realizing a DMT. One solution is to replicate the tree structure to all involved network nodes and use unique identifiers for storing hash values in the DHT. The cost to construct a verification path is $O(\log n)$ locate operations, however, the cost to maintain the tree after structural updates is high, $O(n \log n)$, and also the $O(n^2)$ total storage is prohibitive. Alternatively, a network node can store the entire verification path (hashes) of the element it stores. The path location cost is $O(1)$, however any update on the data set now incurs $O(n \log n)$ cost; the total storage cost is $O(n \log n)$. Our approach is to distribute information about routes in the network that construct verification paths, essentially combining the above ideas: using unique identifiers, hash values are stored in the network and the network node storing an element also stores the identifiers of the corresponding verification path. For efficiency in the dynamic case, we use a weight-balanced ($BB[\alpha]$) hash tree. Details about the construction and its analysis follow.

### 3.1 Our Construction

We consider the more general case, where an authentication structure over $m$ data items $\{x_1, x_2, \ldots, x_m\}$, owned by the same source, is stored in a p2p network of size $n \geq m$ that realizes a distributed hash table. We design our distributed Merkle tree using the primitive locate operation over the network. Without loss of generality, we assume that objects are stored at distinct network nodes. Our results generalize to the cases where more than one data items are stored at nodes and also where more than one sources produce these items.

Our scheme is described as follows (see Figure 2). For convenience, tree nodes are denoted by lower-case letters and network nodes by capital letters.

Let $T$ be a balanced binary tree built over the elements of (dynamic) data set $\{x_1, x_2, \ldots, x_m\}$, with one-to-one correspondence between leaves and elements, and let $h$ be a cryptographic hash function. Each tree node $u$ in $T$ has a unique identified $id_u$ (drawn efficiently from some space). Conventionally, the identifier of a leaf is set to be the corresponding element. Tree $T$ is used as a hashing structure in the standard way: each non-leaf tree node $u$ with children nodes $v_1$ and $v_2$ in $T$ is associated with (or stores, conceptually) hash value $L(u)$, which equals to $h(L(v_1) \| L(v_2))$, i.e., the hash of the hash values that $v_1, v_2$ are associated with, and each leaf $w_i$ stores the hash value $L(w_i) = h(x_i)$ of the corresponding element $x_i$. We augment the hashing structure as follows: we require that each internal tree node also stores the hash values of its children.

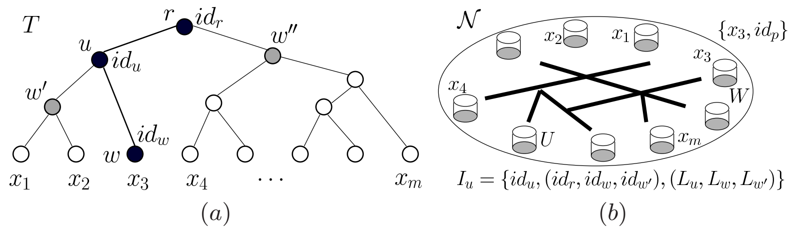

**Fig. 2.** $(a)$ Hash tree $T$ over elements $x_1, \ldots, x_m$: node $u$, with identifier $id_u$ storing hash $L_u$, is mapped to network node $U$ and leaf $w$, storing $x_3$ with verification path $p$, to network node $W$. $(b)$ Distribution of $T$ over the network: $U$ stores information $I_u$ related to $u$; $W$ stores $\{x_3, id_p\}$ and structural/balancing information of $p$.

Next, we use the tree identifiers for distributing tree $T$ into the nodes of the underlying p2p network. Each non-leaf tree node $u$ is mapped to a network node $U$ through a function $f$ and according to $id_u$, i.e., $U = f(id_u)$. Node $U$ stores some information $I_u$ related to $u$: the (three) hash values associated with node $u$, the tree identifiers of the parent tree node and the children of $u$ and local structural information about node $u$. Moreover, a leaf node $w_i$, corresponding to element $x_i$, is also mapped to a network node $W_i = f(x_i)$ through function $f$.[3]

[3] We impose no restrictions on $f(\cdot)$; in general, it is a known function used by the underlying p2p network for mapping objects to network nodes (many DHTs use the SHA-1 function), implemented by the locate operation supported by the network.

Along with $x_i$, node $W_i$ stores some information $I_{x_i}$ related to the (verification) path $p$ in $T$ from $w_i$ to the root $r$ of $T$; in particular, this information includes:

– the ids of the tree nodes of path $p$, denoted as $id_p$;
– structural and balancing information of tree nodes in $p$; for each node $u$ in $p$ with children $v_1$ and $v_2$, $W_i$ stores: (1) whether $v_1$ or $v_2$ belongs in $p$; (2) the balancing information of node $u$, which is a pair $(b_1, b_2)$, expressing balancing information related to the subtrees defined by $v_1$ and $v_2$ respectively.

The above scheme distributes tree nodes and verification paths over a p2p network and correctly implements a DMT. Our construction is designed mainly for bottom-up tree traversal, which is appropriate for most security-related and cryptographic applications, although it can be easily extended to support also top-down traversal, similar to search tree (see Section 4). Accordingly, our tree is accessed very efficiently: given a data element $x_i$, the corresponding verification path is distributively retrieved using $O(\log m)$ locate operations for mapping identifiers in $id_p$ to network nodes. Using route distribution, that is, maintaining the invariance that each network node knows the route for its verification path, we can actually achieve extra efficiency, as we discuss at the end of the section.

Finally, we choose our tree $T$ to be a weight-balanced tree and, in particular, a $BB[\alpha]$ tree [22] (weight-balanced trees with important balancing properties). This choice is related to efficiency in maintaining the structure of the hash tree after updates in the data set (element insertions or deletions). We consider two types of updates along a verification path: hash updates (due to rehashing) and structural updates (due to re-balancing changes, i.e., rotations). Route distribution supports efficient hash updates, since $O(\log m)$ locate operations suffice in updating the hash values of a path $id_p$. Similarly, structural updates can be executed by successively contacting the network nodes corresponding to $id_p$ and performing any necessary local update at node $U$ using information $I_u$, however, they incur an additional cost. Any rotation at level $k$ of $T$ triggers an extra update cost for publishing the new routes to the $O(2^k)$ in total involved network nodes. Using a $BB[\alpha]$ tree, where $\alpha$ is a balancing parameter bounding the ratio of the weights (i.e., size of corresponding subtree) of neighboring tree nodes, we achieve that on average $O(1)$ rotations occur after an update and they occur more often at nodes closer to leaves than at nodes higher in $T$. This gives on average a very good performance in an amortized sense, since expensive reconstructions happen rarely. The following lemma formalizes the above argument.

**Lemma 1.** *For any series of $m$ update operations on an initially empty set, the DMT based on a $BB[\alpha]$ hash tree $T$, with $\alpha \in (\frac{1}{4}, 1 - \frac{\sqrt{2}}{2})$, has $O(\log n \log m)$ amortized structural update cost. Moreover, during these operations, structural updates at level $k$ of $T$ with cost $O(2^k)$ occur with frequency $O(\frac{1}{2^k})$.*

*Proof.* (Sketch.) The proof is based on the update technique in our scheme and the properties of $BB[\alpha]$ trees (e.g., see analysis in [22]). Consider any update in the data set that results in a structural update in the tree $T$, in particular, along the verification path $p_i$ of element $x_i$, and assume that the corresponding network node $W_i$ storing $x_i$ has been located. Node $W_i$ can initiate a structural update along $p_i$ by examining $p_i$ in bottom-up fashion and contacting the appropriate network nodes. If $p_i$ is structurally updated to $p'_i$ and a rotation occurs at node $u$ in $T$, let $T_u$ be the subtree in $T$ defined by $u$. Each network node corresponding to leaf node $x_i$ in $T_u$ or a neighboring node $v$ of $u$ in $T_u$ must respectively update its local information $I_{x_i}$ and $I_v$. In total, $O(|T_u|)$ network nodes must be notified about the updates in $T$. This is possible by accessing $T_u$ in a top-down fashion (starting at $u$), contacting the corresponding network nodes and communicating the necessary updates, a process completed after $O(|T_u|)$ locate operations using $O(|T_u| \times \log n)$ communication. Overall, the structural update cost is $O(|T_u|)$ locate operations. By the properties of $BB[\alpha]$ trees with parameter $\alpha$ in the appropriate range, we have that the total cost for updating all verification paths in the DHT, for a sequence of $t$ update operations (insertions or deletions) on an initially empty set, is $O(t \log t)$. Thus, for the same series of update operations, the total structural update cost is $O(\log n \times t \times \log t)$ (time and communication). For $t = O(m)$, we get that the amortized overall structural update cost is $O(\log n \log m)$ over a sequence of operations of size linear on $m$. Using the additional property shown in [22], namely that costly rotations at levels close to the root occur rarely with frequency inversely proportional to the corresponding subtree size, the proof is completed. □

The following theorem summarizes the efficiency of our DMT and our main result. A p2p network with $n$ nodes is called *efficient* if location operations take time $O(\log n)$.

**Theorem 1.** *There exists a scheme for implementing a distributed Merkle tree $T$ on a data set of size $m$ over a peer-to-peer network $\mathcal{N}$ with $n$ nodes ($m \leq n$) with the following properties:*

1. *Tree $T$ uses space $O(m \log m)$, distributed over $O(m)$ network nodes, and incurs $O(\log m)$ storage overhead per network node.*
2. *A verification path of $T$ has size $O(\log m)$ and can be accessed with $O(\log m)$ locate operations on $\mathcal{N}$; thus, if $\mathcal{N}$ is efficient, the expected computational and communication cost for accessing a verification path is $O(\log n \log m)$.*
3. *A hash update on tree $T$ involves $O(\log m)$ location operations; thus, for an efficient network $\mathcal{N}$, the expected computational and communication cost of a hash update is $O(\log n \log m)$.*
4. *A structural update on tree $T$ involves $O(m \log m)$ location operations, amortized over a series of $O(m)$ structural updates on an initially empty tree; thus, for an efficient network $\mathcal{N}$, the expected amortized computational and communication cost of a structural update is $O(\log n \log m)$.*

*Proof.* (Sketch.) Tree $T$ is balanced, thus verification paths have $O(\log m)$ size. The storage complexity is $O(m \log m)$, since internal tree nodes require $O(1)$ storage and leaves $O(\log m)$ storage. Through route distribution, constructing any verification path requires only $O(\log m)$ locate operations, performed by the initiating node according to $p_i$. Similarly, hash updates are performed by bottom-up traversal of nodes in $p_i$. Structural updates are performed as in Lemma 1. □

Note that above, storage is optimal for route distribution and in accordance with storage requirements for an efficient network, where each network node stores $O(\log n)$ routing information; thus, our DMT does not asymptotically increase the storage requirements of the underlying network.

**Improvement through caching.** We discuss a simple extension that under a reasonable assumption, can improve the costs for path location and update. Assuming that network-node failures occur less often than queries and updates on the DMT, we can improve the efficiency of our scheme as follows. The goal is to transform the multiplicative $O(\log n)$ factor (introduced due to locate operations for retrieving or updating hash values) into an additive term in the complexity of our scheme. This is achieved by caching network node identifiers: the idea is to have each network node corresponding to a leaf of $T$ to cache in its memory the identifiers of the $O(\log m)$ network nodes that store the hash values of its corresponding verification path. That is, once such a network node is first contacted, its identifier is remembered. Since network nodes can fail or go down, it is possible that cached nodes are no longer nodes of the network. In this case, we have a cache miss which will trigger a location operation. Although we can still use some techniques to avoid this overhead (e.g., by caching neighboring nodes storing the same information due to redundancy), we observe that when the rate of network node failures is sufficiently small then we can actually amortize the $O(\log n)$ factor due to occasional location operations (cache misses) in the cost for operating on the tree. In particular, if network nodes fail independently with probability $O(\frac{1}{\log m})$ during the time interval of a tree traversal, then the expected number of network node failures that occur during a path location or update is $O(1)$. Thus, using caching the expected complexity for path location and updates on the tree is $O(\log n + \log m)$.

Table 2 summarizes the comparison between the various schemes for implementing a DMT. We see that our scheme provides an very efficient solution that, using caching and under reasonable assumptions, can be asymptotically optimal in an amortized sense. Finally, we note that, when $m > n$, we can appropriate extend our scheme by having each network node maintaining an additional data structure (for locating the stored elements). Our scheme supports authentication of data collections of one data source; we can support multiple data sources simply by using multiple instantiations of our DMT.

**Table 2.** Efficiency comparison of various schemes for realizing a DMT for a data set of size $m$ over a p2p network of size $n$. Expected complexity is denoted with $^*$ and amortized expected complexity is denoted with $^{**}$.

| | storage | path location | hash update | structural update |
|---|---|---|---|---|
| tree replication | $O(m^2)$ | $O(\log n \ \log m)^*$ | $O(\log n \log m)^*$ | $O(m \log n)^*$ |
| path replication | $O(m \log m)$ | $O(1)$ | $O(m \log n)^*$ | $O(m \log n)^*$ |
| **route distribution** | $O(m \log m)$ | $O(\log n \log m)^*$ | $O(\log n \log m)^*$ | $O(\log n \log m)^{**}$ |
| **w/ caching** | $O(m \log m)$ | $O(\log n)^*$ | $O(\log n)^*$ | $O(\log n)^{**}$ |

## 4 An Efficient Authenticated Distributed Hash Table

In this section, we design an authentication scheme for membership operations on dynamic sets in our authentication model. First, we use our DMT construction to realize authentication protocols for verifying the basic operations of any DHT and we design an efficient *authenticated distributed hash table* (*ADHT*).

Consider a source $S$ that produces $m$ data objects as key-value pairs, which are stored in a DHT that supports the basic put-get operations. We design protocols for augmenting this functionality to a new p2p storage system that provides better information assurance, supporting the following authenticated operations:

- auth_put: a key-value pair is inserted in the system by source $S$ in an authenticated way, so that $S$ verifies the correctness of the insertion;
- auth_get: the value of an existing in the system key is retrieved by a user in an authenticated way, so that the user verifies the authenticity of the value;
- auth_remove: an existing key-value pair is removed from the system by source $S$ in an authenticated way, so that $S$ verifies the correctness of the removal.

**Implementation.** To realize the above functionality of an ADHT, we use the technique of signature amortization over the data set that exists in the system. The main idea is use the DMT of the previous section as a distributed authentication structure for implementing the following invariant: at all times, source $S$ maintains (by storing locally) a cryptographic digest of the currently correct (up-to-date) data set. Query verification is achieved by having $S$ signing the digest along with a fresh timestamp and storing this information in the system. Update verification is achieved by having the source computing the new digest that correctly corresponds to the new data set after the update, using authentication information that is first verified against the current digest.

The digest is defined as the root hash value of the tree $T$ that is built over the data set and that corresponds to the DMT maintained in the p2p system. We augment the construction of $T$ as follows. First, we add an additional level of hashing in $T$: the leaf node corresponding to pair $(k, x)$ now stores hash value $h(h(k)\|h(x))$. Moreover, each non-leaf tree node $u$ with children nodes $v_1$ and $v_2$ in $T$ stores a hash value $L(u)$ that also encodes the structural and balancing information of $u$, that is, $L(u) = h(L(v_1)\|L(v_2)\|h(s_u))$, where $s_u$ encodes whether $u$ is a left or right child and its balancing information.[4] Additionally, we augment $T$ to also serve as a *search tree*[5]: key-value pairs are sorted according to their keys (we assume they are drawn from a totally order set) and are stored at the leaves matching their left-to-right ordering; also, each non-leaf node $u$ stores a corresponding search key, e.g., the maximum key stored at the leaves of $T_u$. We assume that, using bootstrapping techniques, both $S$ and the users have access (through direct connection) to an active node of the p2p network and that there exists a publicly known function for creating identifiers for the new nodes in $T$.

We next describe the exact protocols for the authenticated operations.

[4] For $BB[\alpha]$ trees, this is the ratio $|T_{v_1}|/|T_u|$, assuming that $v_1$ is the left child of $u$.
[5] Not needed, if the DHT supports searches for both exact and near matches [2, 13, 10].

**Queries.** Queries are performed by any user by first contacting a network node and issuing a get request on a key. For queries, the verification path of a leaf is simply the corresponding leaf-to-root path. A path retrieval query is executed by the system over the DMT and what is returned to the user is: (1) the corresponding value, (2) the verification path (collection of hash values and relative information for computing the root hash) and (3) the signed digest. The user accepts the answer if and only if $(i)$ hashing over the value and the verification path results in a hash value that equals the root hash (digest) and $(ii)$ the signature on the digest is valid and contains a fresh timestamp.
**Updates.** Updates are performed by the source $S$ by first contacting a network node and issuing an update request. For updates, the verification path of a leaf is augmented to also include the sibling nodes of the nodes in leaf-to-root path. The system then reports to $S$ the verification path $p_\ell$ of the leaf $\ell$ of the DMT $T$ that is related to the update, i.e., the sibling leaf of the new leaf (put operation) or the leaf to be deleted (remove operation). For put operations, $\ell$ can be located through top-down traversal of $T$; we assume that $S$ stores, and updates when needed, the identifier of the root. Path $p_\ell$ contains all the necessary information for computing the digest, given the information stored at $\ell$ (note that, in the case of a put operation, it contains the information stored at the sibling of $\ell$). In particular, $p_\ell$ contains all the structural and balancing information that is needed for any hash or structural update on it. This information serves as a *consistency proof* for the source $S$ and is used to compute the new digest in three steps. First, the verification path is checked to be authentic by hashing along the path and recomputing the current digest; this also verifies that the structural and balancing information is also authentic, consistent with the current digest stored by $S$. In the check fails, the protocol rejects: the system failed to correctly execute the previous update. Otherwise, $S$ locally executes the tree update by operating on path $p_\ell$; this is feasible because both hash updates and structural tree adjustments only happen along this path in a bottom-up fashion. Finally, $S$ hashes over the new tree path and computes and stores the new digest. Once the new digest is computed, $S$ timestamps and signs it and returns the signed copy for storage in the system. Then a regular hash-tree update is performed by the system to execute the put or remove operation. By this interaction, $S$ needs only to keep $O(1)$ authentication information, the current signed digest. Asymptotically, no additional computational or communication cost is introduced by this extra interaction between the system and the source.
**Security.** The security of our protocols can be proved with standard reductions to the security of the cryptographic primitives that are used in our authentication scheme, under standard hardness assumptions. By using a family of collision-resistant hash functions and a signature scheme secure against adaptive chosen-message attacks, we have that the authentication scheme in ADHT is secure: any successful attack by the network against the security of our scheme corresponds to either a forged signature or a hash collision.

Before stating the main result of this section, we recall that a p2p network with $n$ nodes is called efficient if location operations take time $O(\log n)$.

**Theorem 2.** *There exists an authenticated distributed hash table over an efficient peer-to-peer network with $n$ nodes that supports authenticated operations* auth_put*,* auth_get *and* auth_remove *on a data set of size $m \leq n$ and has the following properties:*

1. *The distributed authentication scheme is secure.*
2. *The storage at the source is $O(1)$; the storage at the network is $O(m \log m)$.*
3. *The query cost is $O(\log m)$, that is, $O(\log m)$ locate operations; or, equivalently, the expected time and communication complexity to answer a query is $O(\log n \log m)$.*
4. *The amortized update cost is $O(\log m)$, that is, $O(\log m)$ locate operations; or, equivalently, the amortized expected time and communication complexity of an update is $O(\log n \log m)$.*

Table 3 summarizes the comparison of our ADHT and its extension ADHT-c using caching with the existing data authentication schemes for dynamic content in p2p storage networks. Existing authentication methods support data integrity by separately signing all data items stored by the same source, but they are either vulnerable to replay attacks ("Sign-all"), since old items can still be incorrectly verified, or induce a significant update cost when timestamping is used to eliminate replay attacks ("Sign-all"-t), since all data items should be resigned after any update or refreshed at regular time intervals. Using a logarithmic space overhead per network node, ADHT provides a secure, efficient and distributed new authentication scheme for verifying basic operations over p2p storage systems and offers a transparent security layer that is independent of the exact implementation of the system.

**Table 3.** Comparison of ADHT with "sign-all" schemes for authenticating queries on data set of size $m$ over a network of size $n$, $m \leq n$. Expected complexity is denoted with $^*$ and amortized expected complexity is denoted with $^{**}$.

| | storage | signing cost | query cost | update cost | replay-safe |
|---|---|---|---|---|---|
| "Sign-all" | $O(m)$ | $O(m)$ | $O(\log n)^*$ | $O(\log n)^*$ | no |
| "Sign-all"-t | $O(m)$ | $O(m)$ | $O(\log n)^*$ | $O(m \log n)^*$ | yes |
| **ADHT** | $O(m \log m)$ | $O(1)$ | $O(\log n \log m)^*$ | $O(\log n \log m)^{**}$ | yes |
| **ADHT-c** | $O(m \log m)$ | $O(1)$ | $O(\log n + \log m)^*$ | $O(\log n + \log m)^{**}$ | yes |

**Distributed Authenticated Dictionary.** An immediate application of our ADHT is a *distributed authenticated dictionary*, where membership queries on a dynamic data set of key-value pairs are authenticated. Suppose that keys are drawn from a totally ordered universe. Our DMT is built on top of the data elements sorted according to their keys. To support authentication of negative answers, we use the technique of [17]: pairs of keys that are consecutive in the ordering used in the Merkle tree are inserted in the system for proving non-membership in the set. E.g., if $(k, v)$, $(k', v')$ are members of the set and $k'$ is the immediate successor key of $k$, then value $v$ contains also key $k'$. The DMT is again used also as a search tree. This scheme has asymptotically the same performance as the ADHT described above, given by Theorem 2.

**Load-balance issues.** Although our authentication structure achieves load balance with respect to data distribution over the p2p network (given the properties of the underlying DHT), as described, it does not achieve load balance with respect to network access. For instance, network nodes that store the tree root are accessed much more often than other network nodes. This turns out to be an important issue that appears to hold in general: all existing techniques for achieving authentication over DHTs that use signature amortization, including our technique or techniques based on self-certified data, introduce congestion at certain network nodes. The problem is challenging, since load-balancing and efficient content authentication in p2p systems correspond to contradictory design goals: signature amortization introduces heavily accessed points in the system, whereas for load-balancing we wish network nodes to be accessed with uniform, rather than skewed, distribution. However, we propose the following simple solution for load-balance in an amortized sense: after any structural update at node $u$ in the tree, we choose new tree identifiers for all nodes in the corresponding subtree $T_u$. Asymptotically no extra cost is incurred, since the structural update already propagates over $T_u$, thus the system can afford redistributing $T_u$ to new network nodes. Identifiers are chosen according to a random, well-defined and unpredictable way, such that no significant communication overhead is introduced in the structure. Effectively, over time, we expect to achieve smoother (closer to uniform) access patterns for network nodes.

## 5 Conclusions and Future Work

We consider the problem of data authentication in p2p storage networks. We introduce a new model for authenticating data in decentralized computing environments that extends the model of authenticated data structures and better captures the security needs of existing distributed systems. We design the first efficient implementation of a distributed Merkle tree (DMT) and show how it can be applied to the design of an authenticated distributed hash table that supports efficient verification of membership and update operations over dynamic data sets and eliminates the threat of replay attacks.

As future work, we plan to implement the DMT construction and experimentally test its efficiency and study load-balance, concurrency and other performance issues. We leave as open problems the design of authenticated schemes for more general queries, beyond set-membership, and the study of additional security issues in this new model, such as DoS attacks and Byzantine behavior.

### Acknowledgments

We would like to thank David Croston, Michael Goodrich, John Jannotti and John Nuber for useful discussions on the subject of this paper.

## References

[1] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. ISC*, pages 379–393, 2001.

[2] J. Aspnes and G. Shah. Skip graphs. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.

[3] M. Castro, P. Druschel, A. J. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured p2p overlay networks. In *Proc. OSDI*, pages 299–314, 2002.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. SOSP*, pages 202–215, 2001.

[5] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *Proc. CCS*, pages 136–145, 2001.

[6] 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.

[7] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proc. HOTOS*, page 75, 2001.

[8] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proc. of Symposium on Discrete Algorithms*, pages 1–23, 2002.

[9] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[10] M. T. Goodrich, M. J. Nelson, and J. Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proc. SODA*, pages 384–393, 2006.

[11] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proc. of RSA Conference*, pages 295–313, 2003.

[12] E. Hall and C. S. Julta. Parallelizable authentication trees. In Cryptology ePrint Archive, December 2002.

[13] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A scalable overlay network with practical locality properties. In *Proc. USENIX Symp. on Internet Technologies and Systems*, 2003.

[14] R. Huebsch, B. Chun, J. Hellerstein, B. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *Proc. of CIDR*, pages 28–43, 2005.

[15] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *Proc. VLDB*, pages 661–672, 2005.

[16] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. of 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[17] P. C. Kocher. On certificate revocation and validation. In *Proc. of International Conference on Financial Cryptography*, pages 172–177, 1998.

[18] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. of ACM SIGMOD*, pages 121–132, 2006.

[19] J. Li, K. Sollins, and D.-Y. Lim. Implementing aggregation and broadcast over distributed hash tables. *ACM SIGCOMM Computer Communication Review*, 35(1):81–92, 2005.

[20] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pages 127–140, 2003.

[21] 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.

[22] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, Germany, 1984.

[23] R. C. Merkle. A certified digital signature. In *Proc. CRYPTO*, pages 218–238, 1989.

[24] G. Miklau and D. Suciu. Implementing a tamper-evident database system. In *Proc. Asian Computing Science Conference, ASIAN*, pages 28–48, 2005.

[25] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Proc. of Network and Distr. System Security*, 2004.

[26] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. of USENIX Security Symposium*, pages 217–228, 1998.

[27] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proc. SPAA*, pages 50–59, 2002.

[28] G. Nuckolls. Verified query results from hybrid authentication trees. In *Proc. of Database Security*, pages 84–98, 2005.

[29] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. SPAA*, pages 311–320, 1997.

[30] S. Ramabhadran, J. Hellerstein, S. Ratnasamy, and S. Shenker. Prefix hash tree - an indexing data structure over distributed hash tables. In *Proc. of ACM PODC*, pages 368–368, 2004.

[31] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, pages 161–172, 2001.

[32] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proc. SIGCOMM*, pages 73–84, 2005.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *LNCS*, 2218:329, 2001.

[34] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[35] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of International Workshop on P2P Systems*, pages 261–269, 2002.

[36] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. SIGCOMM*, pages 149–160, 2001.

[37] R. Tamassia. Authenticated data structures. In *Proc. of European Symposium on Algorithms*, pages 2–5, 2003.

[38] R. Tamassia and N. Triandopoulos. Computational bounds on hierarchical data processing with applications to information security. In *Proc. ICALP*, pages 153–165, 2005.

[39] D. S. Wallach. A survey of peer-to-peer security issues. In *Proc. of International Symposium on Software Security*, 2002.

[40] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.