Super-Efficient Verification of Dynamic Outsourced Databases*

Michael T. Goodrich[1], Roberto Tamassia[2], and Nikos Triandopoulos[3]

[1] Dept. of Computer Science, UC Irvine, USA
goodrich@ics.uci.edu
[2] Dept. of Computer Science, Brown University, USA
rt@cs.brown.edu
[3] Dept. of Computer Science, University of Aarhus, Denmark
nikos@daimi.au.dk

Abstract. We develop new algorithmic and cryptographic techniques for authenticating the results of queries over databases that are outsourced to an untrusted responder. We depart from previous approaches by considering *super-efficient* answer verification, where answers to queries are validated in time asymptotically less that the time spent to produce them and using lightweight cryptographic operations. We achieve this property by adopting the decoupling of query answering and answer verification in a way designed for queries related to range search. Our techniques allow for efficient updates of the database and protect against replay attacks performed by the responder. One such technique uses an off-line audit mechanism: the data source and the user keep digests of the sequence of operations, yet are able to jointly audit the responder to determine if a replay attack has occurred since the last audit.

1 Introduction

Large databases are increasingly being outsourced to untrusted third parties (responders) and without some kind of verification mechanisms, users cannot trust the answers to queries. Thus, an important component of any outsourced database system is the security of its answer-verification process. Moreover, database outsourcing is typically realized for efficiency purposes in a distributed setting where clients are machines that have low computational power running applications that demand authentic responses of dynamic data at high rates. In this context, the cryptographic protocols for trustworthy answer verification

* Research supported in part by the U.S. National Science Foundation under grants IIS–0713403, IIS-0713046, CNS-0312760 and OCI–0724806, the Institute for Information Infrastructure Protection under an award from the Science and Technology Directorate at the U.S. Department of Homeland Security, and the Center for Algorithmic Game Theory at the University of Aarhus under an award from the Carlsberg Foundation. The views in this paper do not necessarily reflect the views of the sponsors.

T. Malkin (Ed.): CT-RSA 2008, LNCS 4964, pp. 407–424, 2008.

should incur small communication and computational overheads that ideally depend only on the answer size.

This paper studies protocols for authenticating the integrity of outsourced databases in ways that achieve high security and efficiency levels. Most database queries boil down to *one-dimensional range search* queries—asking to report those records having values of a certain field within a given interval—and most existing techniques for authenticating such queries have $O(\log n + t)$ communication and computational costs, where n is the total number of records in the database and t is the number of returned records. Instead, our goal is to design cryptographic techniques that allow *super-efficient* answer verification, that is, allow authentication of range search queries with only $O(t)$ associated costs, even when t is $o(\log n)$. Furthermore, we wish our protocols to involve lightweight cryptographic operations with, ideally, only $O(1)$ modular exponentiations performed during the answer-verification process.

Additionally, we seek authentication solutions that perform well even if the database evolves frequently over time. The main challenge in this context is that a malicious responder may perform a *replay attack*, i.e., provide verifiable (e.g., signed) but stale or currently invalid information (e.g., that was originated from the owner long in the past) to a client. But here is exactly where super-efficiency can hurt us, since we want to avoid a verification method that requires more than $O(t)$ work on the part of the client, and we want to avoid requiring the data owner to process (e.g., re-sign) all the records of the database with each update. Ideally, we would like a dynamic system that is super-efficient for the client, and immune to replay attacks launched by the responder, and that can process updates efficiently for the data owner and the responder.

Super-efficient verification is a theoretically interesting concept, since it advances the design of data authentication protocols by exploring the possibility of removing unnecessary computations at the verifier. But it is also a practically important property in database systems, since it provides trustworthy functionality in dynamic and highly distributed data dissemination models, where small mobile and computationally limited devices query continuously and at high rates data that is outsourced to untrusted, geographically dispersed, proxy machines.

Related Work. Extensive work exists on *authenticated data structures* [19, 25], which model secure data querying in adversarial environments, where data created by a trusted source becomes available to users through queries after it is replicated to an untrusted remote server. The general approach is to augment the data structures used by the source and the responder to support authentication protocols such that, along with an answer to a query, a cryptographic proof is provided to the user by the server that can be used to verify the authenticity of the answer. Research has mostly focused on hash-based authentication protocols, where extensions of Merkle's *hash tree* [16] are used for authenticating membership queries (e.g., [6, 11, 19, 26, 27]) or more general query types, such as basic operations on relational databases [9], pattern matching and orthogonal range searching [15], graph connectivity and geometric searching [13], XML queries [4, 8], and two-dimensional grid searching [1]. Many of these queries

essentially boil down to one-dimensional range search queries. General authentication techniques have been also proposed for certain query classes, including read-write operations on memory cells [5], queries on static data that are modeled as search DAGs [15], and decomposable queries over sequences and iterative searches over catalogs [13]. These schemes are not super-efficient as they involve answer proofs and verification times that asymptotically equal the complexity of answering queries. In [26], for hash-based authentication of set-membership queries, it is showed that for a set of size n, all costs related to authentication are at least logarithmic in n in the worst case. Related work on consistency and privacy of committed databases appears in [6, 17, 22]. Authenticated dictionaries in the two-party model, where the source keeps minimal state to check the integrity of its outsourced data, appear in [10, 24]. Finally, in [12] it is showed how to use the RSA accumulator [7] to realize a dynamic authenticated dictionary that achieves constant (thus super-efficient) verification costs at the client.

There has also been a growing body of work on authenticating queries in outsourced databases. The model is essentially the one of authenticated data structures, but now the data sets are relational databases residing in external memory and are queries through SQL queries which are founded on one-dimensional range search. In [9, 13] range queries are supported with $O(\log n+t)$ authentication costs. In [21], cryptographic hashing and accumulators are used in the first hash-based super-efficient, but static, verification scheme that achieves $O(\log t)$ communication cost and $O(t)$ verification cost, whereas in [23], static hash trees, where each tree node is individually signed, are used to authenticate range queries, incurring cost of $O(t)$ signature verifications. In [20] signature aggregation is used to accelerate the verification of the (individually signed) answer records. Both schemes achieve super-efficiency, but not coupled with both efficient updates and replay-attack safety. Finally, in [14] authentication techniques based on B-trees and aggregated signatures are studied experimentally.

Table 1. A summary of how our results are qualitatively compared with existing work

	[5, 13]	[21, 18]	[20]	**this work**
super-efficient		•	•	•
dynamic	•		•	•
replay safe	•	n.a.		•

Our Contributions. We provide the first super-efficient authentication techniques for one-dimensional range searching (or queries based on it), that are both *dynamic* and *replay safe*. Our schemes can support fast query time for the untrusted responder, super-efficient verification for clients, and fast update time for the data source. Our main technique for achieving these properties involves the use of an optimal authentication structure (employed separately by the source and the responder) that divides a hash tree in a recursive fashion so that it has $O(\log^* n)$ "special" levels (i.e., a number proportional to the inverse of the tower-of-twos function). The database owner needs to authenticate only the hash values of tree nodes that lie on the special levels, which significantly speeds

up data updates while also simplifying the means to achieve super-efficiency. Indeed, for all practical applications, there are only a constant number of special levels in our scheme. Table 1 summarizes the comparison of our work with the best existing methods for authentication of range searching in outsourced data.

To avoid the possibility of replay attacks, we provide two possible solutions. One solution involves the use of an RSA accumulator to allow clients to verify a single secure aggregation to check that the signed responses to a query are still valid even if some individual signatures are possibly quite old. We use a source-responder work trade-off to perform updates in $O(\sqrt{n})$ time, which is efficient for moderately large values of n. Our second solution provides a different trade-off, between the update cost at the source and responder and the immediacy in detecting a replay attack. We show how to build an off-line *auditing* mechanism to detect, and thereby deter, replay attacks through periodic audits of the responder. The key contribution here is that the auditor mechanism, based on an off-line memory-checking test introduced in [5], is implemented jointly but non-interactively by the source and the user and needs only store and process a constant-sized digest to check the responder (so that auditing is also a super-efficient computation), and that the responder cannot employ a replay attack without being caught by the auditing mechanism.

Section 2 describes our authentication model. Section 3 describes our approach for verifying answers to range queries by decoupling answer verification from query answering, and presents our core authentication structure designed to optimally support super-efficient verification. Section 4 describes a dynamic extension of our scheme that provides a trade-off in update and query costs, and Section 5 presents an augmentation of our scheme that realizes an efficient off-line auditing mechanism. Section 6 discusses extensions to support verification of other query types that are related to range searching, and also our final concluding remarks. Focused on one-dimensional range search and due to lack of space, this extended abstract omits some details of our design and proof techniques.

2 Authentication Model

We examine data authentication in the setting commonly used in today's Internet reality, where a database becomes available for queries at an intermediate entity that is distinct from the data owner and untrusted by the end user. In particular, we consider the following three-party data querying and authentication model. A data *source* $\mathcal{S}$ creates (and owns) a *dynamic data set* D, which evolves through update operations, and maintains an *authentication structure* for D, appropriately designed for a specific query type. Data set D is stored by a *responder* $\mathcal{R}$ who maintains the same authentication structure for D and answers queries issued by a *user* $\mathcal{U}$. Along with an answer a to a query q, $\mathcal{R}$ provides $\mathcal{U}$ with a cryptographic proof p that is computed using the authentication structure of D; p is used by a verification process ran by $\mathcal{U}$ to check the validity of answer a subject to query q. On any update for D issued by the source, D and the authentication structure are appropriately updated by $\mathcal{S}$ and $\mathcal{R}$.

The merits of this query model include scalability, decentralization and load-balance: by outsourcing D, $\mathcal{S}$ minimizes its operational costs by processing only data updates (e.g., it minimizes the time being on-line) and heavy query traffics of an unlimited population of users can be securely handled by one (or more) untrusted responders (e.g., proxy servers), *without* the need of creating or updating any trust relations, or installing any secure component at the server.

In this model, our goal is to design an authentication structure that allows trustworthy answer verification, that is, to check that the answer is as accurate as it would have been, had the answer come directly from $\mathcal{S}$. To achieve this, we use the following general approach. Using a PKI, we assume that $\mathcal{U}$ knows the public key of $\mathcal{S}$. The corresponding secret key is used by $\mathcal{S}$ in combination with some cryptographic primitives to produce one (or more) *authentication strings* (or digests) for data set D, which constitute short descriptions of D that capture structural information related to the type of queries of interest. Given any query q, $\mathcal{R}$ uses its authentication structure to produce a proof p for the answer a of q. On input a query-answer pair (q, a), a proof p, and the public key of $\mathcal{S}$, $\mathcal{U}$ runs a verification algorithm that either accepts a as valid or rejects it as invalid: p securely relates a, q to (some of) the authentication string(s), which are authenticated by $\mathcal{S}$ using a signature scheme. We call the set of authentication and communication protocols and verification process, an *authentication scheme*.

We now describe the security requirement that any authentication scheme must satisfy. Security is captured as two individual requirements, modeling the desired property: for all queries the verification process should be trustworthy, accepting an answer-proof pair if and only if the returned answer is the correct answer to the query. First, we require *completeness*, which ensures that for any query the authentication structure generates a correct corresponding answer-proof pair that the verification algorithm accepts. Second, we require *soundness*, which ensures that if, given a query q, an answer-proof pair (a, p) is accepted by the verification algorithm, then a is the correct answer to q. With respect to this requirement, we assume the following threat model. The user $\mathcal{U}$ trusts only the source $\mathcal{S}$, not the responder $\mathcal{R}$ which is modeled as an entity that is controlled by an adversary [1]. $\mathcal{R}$ can maliciously try to cheat, by providing an incorrect answer to a query and forging a false proof for this answer. Accordingly, the soundness requirement dictates that given any query issued by $\mathcal{U}$, no polynomial-time responder $\mathcal{R}$, having oracle access to the algorithm that the source runs to generate the authentication strings,[2] can come up with an answer-proof pair, such that the answer is incorrect, yet the verification algorithm accepts the answer as authentic. This definition implies safety against replay attacks.

In this work, we are interested in secure authentication schemes for verifying the results of range search queries that introduce low computational and communication overhead to the involved parties. In particular, we seek for

[1] We do not consider denial-of-service attacks but assume that $\mathcal{R}$ always participates in the communication protocol and interacts with $\mathcal{S}$ and $\mathcal{U}$.

[2] That is, $\mathcal{R}$ observes the authentication strings of D that are produced by $\mathcal{S}$ over time or selectively query for the authentication strings of specially chosen data sets.

authentication schemes that primarily incur low verification time, called *verification cost*. Other important secondary cost parameters are the *update cost* (for updating the authentication structure at $\mathcal{S}$ and $\mathcal{R}$ after updates) and the *query cost* (for producing the answer-proof pairs at $\mathcal{R}$ after queries), as well as the proof size. In the interest of super-efficient verification, we wish to design schemes that allow very fast answer verification, in time asymptotically less than the time needed for answer generation and tolerate reasonable trade-offs in the update and query costs or the update costs and the immediacy of replay-attack detections.

In our authentication schemes, we use standard cryptographic tools, such as collision-resistant hash functions and digital signatures, and the *dynamic RSA accumulator* [2, 3, 7]. Given a set X of size n, an accumulator can be used to incrementally and order-independently (through bivariate function $f(\cdot,\cdot)$) compute a constant-size *accumulation value* $A(X)$, with respect to which there exist (i) constant-size *witnesses* for all accumulated elements in X, and (ii) a constant-time computationally secure *verification test* that accepts witnesses of only elements existing in X. The RSA accumulator results by setting $f(a,x) = a^x \mod N$ as the result of accumulating new element x in the current accumulation value a, where x is a prime number in the appropriate range and N is an RSA modulo, thus, $A(X) = a_0^{\prod_{x\in X} x \mod \phi(N)}$, where $\phi(\cdot)$ is Euler's function and a_0 an initial (public) value. Membership of x with witness w in set X is tested as $w^x = A(X)$, which is a secure test: under the strong RSA assumption [3, 7], it is computationally infeasible to find items that are not accumulated in the set and corresponding fake witnesses that pass the test. In [12], it is showed how to use this primitive in our three-party authentication model for optimally verifying set membership, that is, how to update elements' witnesses *without* the trapdoor information $\phi(N)$, using $O(\sqrt{n})$ modular operations and multiplications.

3 A New Super-Efficient Authentication Structure

In this section, we present a new authentication structure that allows super-efficient answer verification of one-dimensional range search queries and that is the key component of the authentication schemes presented in the next sections.

Let $D \triangleq \{(k_1, v_1), \ldots, (k_n, v_n)\}$ be a set of n key-value pairs (k, v), where each key k is a distinct element of a totally ordered universe K, where, for simplicity and without loss of generality, $k_1 < \ldots < k_n$ and $n = 2^d$. A one-dimensional *range search* query $q = [q_L, q_R]$ on D is an interval with $q_L, q_R \in K \cup \{-\infty, +\infty\}$, and maps to answer $A_q \triangleq \{(k, v) \in D : q_L \leq k \leq q_R\}$, the subset of D consisting of all pairs whose keys are in $[q_L, q_R]$. We assume that answer A_q can be computed by the responder $\mathcal{R}$ in $O(\log n + t)$ time, using some optimal technique (e.g., searching in a balanced range tree), where $n = |D|$ and $t = |A_q|$.

Our approach for achieving super-efficient verification of range searching is to decouple the authentication structure from the search data structure in order to authenticate a collection of certain relations defined over D. Let the *successor relation* $\sigma(X)$, defined over a totally ordered set X with n elements,

be the set of size $n+1$ that consists of all ordered pairs of consecutive elements in X, augmented with pairs $(-\infty, x_1)$ and $(x_n, +\infty)$, where x_1 and x_n are the smallest and largest elements of X, respectively (e.g., $\sigma(\{1,5,2\}) = \{(-\infty,1)(1,2)(2,5)(5,+\infty)\}$). The successor relation of the keys of D is the essential information for verifying answers of range search queries on D.

Fact 1. *Let $q = (q_L, q_R)$ be a range search query on set D of key-value pairs, K_D be the set of keys in D and $A_q = \{(k_{i_1}, v_{i_1}), \ldots, (k_{i_t}, v_{i_t})\}$, $k_{i_1} < \ldots < k_{i_t}$, be a set of key-value pairs. Then $A = A_q$ if and only if there exist keys $k_{i_0}, k_{i_{t+1}} \in K_D$ such that: (1) $\{(k_{i_0}, k_{i_1}), (k_{i_1}, k_{i_2}), \ldots, (k_{i_{t-1}}, k_{i_t}), (k_{i_t}, k_{i_{t+1}})\} \subseteq \sigma(K_D)$ and $A \subseteq D$; and (2) $k_{i_0} < q_L \leq k_{i_1}$ and $k_{i_t} \leq q_R < k_{i_{t+1}}$.*

Indeed, keys $k_{i_0}, k_{i_{t+1}}$ correspond to the boundaries of the range interval, each one possibly coinciding with fictitious keys $-\infty$ or $+\infty$, with $(k_{i_0}, k_{i_{t+1}}) \in \sigma(K_D)$ if $A_q = \emptyset$. The first condition guarantees that the answer A consists of t consecutive key-value pairs of data set D, whereas the second that the query range is exactly covered by the answer range. Thus, in our formulation, answer correctness for range searching captures both inclusiveness (all returned pairs are in the query range) and completeness (all pairs in the query range are returned).

It follows that, if $A_q = \{(k_{i_1}, v_{i_1}), \ldots, (k_{i_t}, v_{i_t})\}$, $k_{i_1} < \ldots < k_{i_t}$, is the correct answer to query q, A_q can be authenticated by verifying (i) t pairs of the key-value relation, namely, that $(k_{i_j}, v_{i_j}) \in D$, $1 \leq j \leq t$, (ii) $t+1$ pairs of the successor relation on the keys, namely that $(k_{i_j}, k_{i_{j+1}}) \in K_D$, $0 \leq j \leq t$, where $k_{i_0} = -\infty$ if $k_{i_1} = k_1$, or $k_{i_0} = k_{i_1 - 1}$ otherwise, and, similarly, $k_{i_{t+1}} = +\infty$ if $k_{i_t} = k_n$, or $k_{i_{t+1}} = k_{i_t + 1}$ otherwise, and, finally, (iii) $t+4$ inequalities (i.e., the ordering of these pairs and that $k_{i_0} < q_L \leq k_{i_1}$, $k_{i_t} \leq q_R < k_{i_{t+1}}$). Assuming uniquely defined representations for the key-value and successor relations, we denote by $\theta(q)$ the resulting set of $2t+1$ pairs to be verified, i.e., $\theta(q) \triangleq \{(k_{i_1}, v_{i_1}), \ldots, (k_{i_t}, v_{i_t})\} \cup \{(k_{i_0}, k_{i_1}), \ldots, (k_{i_t}, k_{i_{t+1}})\}$.

By Fact 1, we have that the problem of authenticating any range search query q on a set D of size n is reduced to the problem of authenticating the membership of the relations of set $\theta(q)$ (of size $O(|A_q|) = O(t)$) in $D \cup \sigma(K_D)$ (the union of the key-value and successor relations defined by D, a set of size $O(n)$). We use this property to decouple the answer verification from the answer generation by designing an authentication structure that for any query q provides super-efficient verification of the corresponding special relations $\theta(q)$ over D. Our construction securely and compactly encodes and authenticates these special relations by associating, in a cryptographically sound manner, the answer A_q, a corresponding proof p and, overall, the relations in $\theta(q)$, with one or more authentication strings that are signed by the source. This structure is used both by $\mathcal{S}$, for computing and signing the authentication strings, and by $\mathcal{R}$, for producing the proofs that will allow $\mathcal{U}$ to verify the answer to queries.

Authentication Structure. Let D be the data set as before. Our authentication structure uses a hash tree built over D that essentially encodes the relations D and $\sigma(K_D)$. In particular, let h be a collision-resistant hash function. We build

a balanced hash tree T of depth d, storing at the leaves from left to right the hash values $h_1, \ldots, h_n$ defined as follows, where $\|$ denotes string concatenation:

- $h_i \triangleq h(h(k_i) \,\|\, h(v_i) \,\|\, h(k_{i+1}))$, $i = 2, \ldots, n-1$, and
- $h_1 \triangleq h(h(-\infty) \,\|\, h(k_1) \,\|\, h(v_1) \,\|\, h(k_2))$, $h_n \triangleq h(h(k_n) \,\|\, h(v_n) \,\|\, h(+\infty))$.

Thus, the hash values at the leaves encode information about various relations: for $2 \leq i \leq n-1$, h_i is the digest of the key-value relation (k_i, v_i) and successor relation (k_i, k_{i+1}), h_1 is the digest of relations (k_1, v_1), $(-\infty, k_1)$ and (k_1, k_2), and h_n is the digest of relations (k_n, v_n), $(k_n, +\infty)$. Internal nodes in T store the hash of the concatenation of the hash values stored at their children. So, any node v in T stores a hash value h_v that is the digest of the key-value and successor relations R_v that are associated with the leaves of the subtree T_v of T rooted at v. For instance, a hash value stored at the parent u of two sibling leaf nodes j and $j+1$ is the digest of the set of relations $R_u = \{(k_j, v_j), (k_{j+1}, v_{j+1}), (k_j, k_{j+1}), (k_{j+1}, k_{j+2})\}$, whereas the hash value h_r of the root r of T is the digest of all relations $R_r = \sigma(D) \cup D$ defined in T.

As we know, in order to authenticate answer A_q, it suffices to authenticate set $\theta(q)$; consequently, due to the collision resistance property of function h and the fact that h_v is the digest of relations R_v associated with the leaves of tree T_v, it suffices to (1) authenticate any set $S_q = \{h_{v_1}, \ldots, h_{v_m}\}$ of hash values stored at tree nodes $v_1, \ldots, v_m$ such that the set of relations $R(S_q) = R_{v_1} \cup \ldots \cup R_{v_m}$ strictly contain set $\theta(q)$, and (2) provide, as proof p, the collection of hash values and relations that associates A_q with S_q in T. Indeed, S_q contains digests that serve as a cryptographic commitment of $\theta(q)$ (computational binding by the collision resistance of h), thus when, given the answer A_q and the proof p, the authenticated hashes in S_q can be recomputed, then one can be assured—subject to the underlying security assumptions of the cryptographic primitives—that A_q is correct, simply by checking the validity of A_q with the test of Fact 1. Set S_q is not uniquely defined but corresponds to a specific query q. Our goal is to define a fixed collection of *special hash values* S such that any query q can be super-efficiently verified by authenticating membership of set $S_q \subset S$ in S. In the simplest case, $\mathcal{S}$ can authenticate S by separately signing its hash values; A_q is verified at hashing and signing costs proportional to $|R(S_q)|$ and $|S_q|$ respectively.

Super-efficient Verification. An efficient approach is to set $S = h_r$, i.e., to use as special hash value for all queries the root hash h_r. Then, for any query q, A_q $|A_q| = t$, can be efficiently associated with h_r, by considering as proof the $O(\log t)$ subtrees of total size $O(t)$ that *exactly cover* the leaves in T that the relations in $\theta(q)$ are associated with, along with the paths connecting these subtrees to r through $O(\log n)$ other tree nodes. The total verification cost is $O(\log n + t)$, which is not super-efficient whenever $t = o(\log n)$ (e.g., $t = O(\log \log n)$ or t is constant). We improve the verification cost as follows.

Suppose that we only query for answers of size $t < \log n$ (see Figure 1). We define the set S_1 of special hash values to contain the hashes $h^1_1, \ldots, h^1_{m_1}$, $m_1 = n/\log n$, at level $\ell_1 = \log \log n$ of the hash tree. It is easy to see that any answer of size t is covered by the subtrees of at most two nodes at level

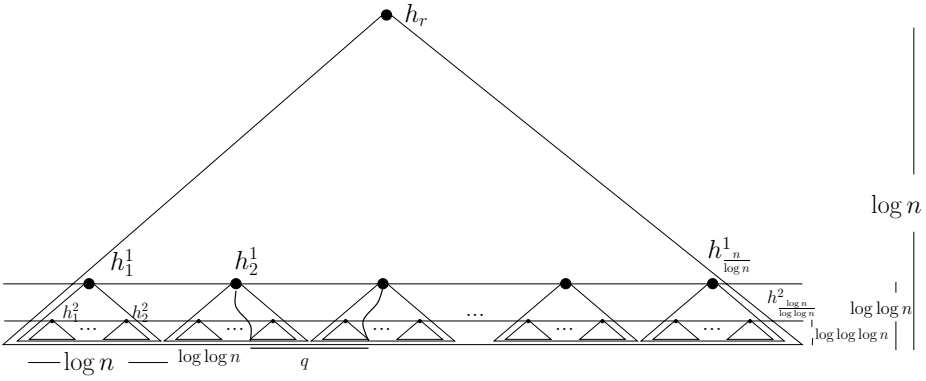

Fig. 1. Our new authentication structure. The set S of special hash values in the tree is defined recursively and consists of $\Theta(n)$ values residing at $\log^* n$ levels: h_r at level $\log n$, $\{h_1^1, ...\}$ at level $\log\log n$, $\{h_1^2, ...\}$ at level $\log\log\log n$, etc. Super-efficient verification is achieved: answer A_q of size at most $t = \log\log n$ to query q is verified by hashing along $O(\log t)$ nodes in the hash tree up to at most two special hash values and by optimally verifying that these hash values are indeed special, i.e., belong in S.

ℓ_1, thus can be verified at $O(\log\log n)$ cost and, if t is $o(\log n)$ and $\Omega(\log\log n)$ we have an improvement and optimal performance. To further improve the verification cost in the case where t is $o(\log\log n)$, we use the above technique to *recursively* define additional special hash values over the $n/\log n$ trees defined by the special hash values in S_1: we consider each one of the trees of size $\log n$ rooted at level ℓ_1 and apply the above technique, assuming that $t < \log\log n$. We define the set S_2 of special hash values to contain the hashes $h_1^2, \ldots, h_{m_2}^2$, $m_2 = m_1 \frac{\log n}{\log\log n}$, at level $\ell_2 = \log \ell_1 = \log\log\log n$ of the hash tree and answers of size t with $\log\log\log n < t < \log\log n$ can be authenticated *super-efficiently* at cost $O(\log\log\log n)$. We proceed as above: at the i-th step of the recursion we define the set S_i of special hash values, we stop before level $\log^* n$, effectively at the level 2 (or some other small constant) of T and set $S \triangleq h_r \cup S_1 \cup \ldots \cup S_{(\log^* n)-1}$ as the final set of special hash values, which is of $\Theta(n)$ size[3].

Our authentication structure lends itself to a first authentication scheme that achieves super-efficient verification: an answer of size t is verified with $O(\log t)$ hashing cost where $O(1)$ special hash values need be authenticated, essentially as being members of the set of special hash values S. In what follows, we consider the case where authentication in S is performed using a signatures scheme, i.e., each value in S is separately signed by the source $\mathcal{S}$. Updates on D are handled by appropriately updating the hash tree T (by hashing and restructuring T along a leaf-to-root path; see also Section 4), having $\mathcal{S}$ sign $O(\log^* n)$ updated special hashes. Replay attacks are eliminated by using *time-stamps* in the signed statements to check the freshness of a valid signature; this is a state-of-the-art solution (see, e.g., [13, 14, 19]) where time is partitioned into fixed and publicly

[3] In fact $|S| < n - 1$, thus, S has smaller size than the trivial solution of $S = T$.

known time-quanta, and verifiable signatures on digests are accepted only if their time-stamps belong in the current (at the time of verification), most recent, time-quantum. For *hash-based authentication*, i.e., in the most practical setting where only cryptographic hashing is used to produce the authentication strings, our authentication structure achieves *optimal performance* with respect to both the verification and the update costs. In particular, using the lower-bound framework of [26], we can show that in the worst case the source needs to authenticate a set S of $\Omega(n)$ special hash values in order to achieve verification costs that are independent of the size n of the database, and that, in this case, time-stamping and signature refreshing is an optimal technique against replay attacks. Thus:

Theorem 1. *There exists a super-efficient authentication scheme for range-search queries over a set of n key-value pairs with the following performance, where t denotes the number of pairs returned by a query: (i) a range query is answered in $O(\log n + t)$ time; (ii) the answer proof has size $O(\log t)$ and consists of two signatures, two keys, and $O(\log t)$ hash values; (iii) the answer to a range query is validated by performing $O(t)$ arithmetic computations, $O(t)$ hash operations, and $O(1)$ signature verifications; (iv) an update results in $O(\log n)$ hash operations (at both the source and the responder), $O(\log^* n)$ signature generations (at the source) and $O(n)$ signature renewals (at the source). This authentication scheme is secure with respect to data authentication, safe with respect to replay attacks, and optimal with respect to super-efficient verification in the hash-based data authentication model.*

4 Super-Efficient Dynamic Authentication Scheme

In this section, we propose an alternative technique that reduces the high update cost of the previous, optimal but less practical, hash-based authentication scheme to get the first super-efficient dynamic authentication scheme for range queries, which provides reasonable trade-offs between the update and query costs.

In Section 3 we constructed a hash tree for a set D of n key-value pairs that encodes information about the key-value and successor relations in D, and we defined a set S of $O(n)$ special hash values that are sufficient to support super-efficient answer verification, provided there is an optimal (in terms of verification) technique for authenticating set-membership queries. Recall that for any query, there are at most two special hash values, out of the total $O(n)$, that need to be verified as members of S, and note that only queries with positive answer need to be authenticated: a special hash value must be verified to be in set S.

We now describe our new authentication scheme. The main idea is to use a dynamic RSA accumulator for authenticating set membership queries for the set of special hash values S. This is performed as follows: the set S of special hash values is accumulated to accumulation value $\alpha = A(S)$ and α is signed by the source. Then, verifying that a special hash value belongs in S is performed in two steps, and still in optimal way ($O(1)$ verification cost): first, the hash value together with the membership witness are used to verify that the hash value was used by the accumulator in producing α and, second, the signature on α is

verified. For security reasons, only the source knows the trapdoor information of the accumulator; the responder does not know the trapdoor. It follows that the verification is (as in the construction of the previous section) super-efficient.

Let us briefly describe the dynamization of the authentication structure, i.e., how updates on the data set can be handled. Assume for simplicity that only values are updated, that is, no keys are inserted or deleted in D. After any update of this type in D, we end up rehashing over a unique-per update operation leaf-to-root path in the hash tree. Thus $O(\log^* n)$ special hash values change and we need to remove the old special hash values from the accumulation α and add the new ones into this, i.e., to perform $O(\log^* n)$ element deletions and insertions in S and update $A(S)$. Inserting and deleting elements in an accumulator involves some computational cost for updating the new accumulation but also for updating the set-membership witnesses of all the elements. Suppose that the witnesses of the $O(n)$ accumulated special hash values are explicitly maintained in the source and the responder. In a highly dynamic setting updates can be of cost $O(n)$: the reason is that after any update all n membership witnesses must be updated. The problem of the high update cost becomes more challenging for deletions, especially under the necessary restriction that the responder cannot use the trapdoor information, but using the RSA accumulator and certain algorithmic techniques [12] we can achieve reasonable update and query costs. We can show:

Theorem 2. *There exists a dynamic super-efficient authentication scheme for range search queries over a set of n key-value pairs with the following performance, where t denotes the size of the returned answer: (i) a range query is answered in $O(\log n + t)$ time; (ii) the answer proof has size $O(\log t)$ and consists of one signature, two field elements, two keys and $O(\log t)$ hash values; (iii) the answer to a range query is validated by performing $O(t)$ arithmetic computations, $O(t)$ hash operations and $O(1)$ modular exponentiation and verifying $O(1)$ signatures; (iv) an update results in $O(\log n)$ hash operations (at both the source and the responder), $O(\sqrt{n}\log^* n)$ modular operations and $O(1)$ signature generations (at the source). This authentication structure is secure with respect to authentication and safe with respect to replay attacks.*

5 Detection and Elimination of Replay Attacks

In Section 3, we presented an authentication structure for range search queries that provides super-efficient answer verification, asymptotically optimally in the hash-based data authentication model. In this section, we propose a new scheme in our three-party authentication model ($\mathcal{S}$, $\mathcal{R}$, $\mathcal{U}$) that achieves efficient update costs at $\mathcal{S}$ and $\mathcal{R}$ (only logarithmic in the database size) and super-efficient verification costs at $\mathcal{U}$ (as before), but uses an alternative solution to the replay-attack problem. In particular, we slightly relax the security requirement with respect to the time when replay attacks are detected and replayed data is rejected. As before, invalid answers are *immediately* rejected by $\mathcal{U}$, but answers are checked to be consistent with the update history in an *off-line* fashion. We introduce a technique which implements an *auditing* mechanism and provides

delayed consistency checking for detecting and effectively eliminating replay attacks. This mechanism augments the authentication scheme of Section 3, so that $\mathcal{U}$ can immediately check any received answer for correctness and at any later time check, in a batch, all received answers for freshness.

Delayed consistency checking is a useful property in application areas where the freshness of answers is not critical to be verified in real time. In many applications, risk management requires that invalid responses must be caught, but this determination does not always have to be immediate, as long as it is certain and sufficiently near-term. Indeed, such swift and sure justice is an ideal circumstance for risk management purposes. Additionally, delayed consistency checking is appropriate when consecutive queries occur sequentially in a short time window and share locality in risk management or equivalent trust relations.

In our auditing mechanism, the delayed consistency checking is performed by the user $\mathcal{U}$, collaboratively with the source $\mathcal{S}$ but without any direct interaction between the two, however. The auditing mechanism corresponds to securely, compactly and efficiently encoding a series of *transactions* with the responder $\mathcal{R}$, i.e., updates and queries over data set D issued by $\mathcal{S}$ and $\mathcal{U}$, respectively. In particular, $\mathcal{S}$ maintains an *update audit state* Σ_u, that encodes the history of updates, through information reported after update transactions with $\mathcal{R}$: for any update u performed on the data set D, an *update trail* T_u is provided to $\mathcal{S}$ by $\mathcal{R}$ that is used to update Σ_u through operation updU. Similarly, $\mathcal{U}$ maintains a *query audit state* Σ_q, that encodes the history of queries, through information reported after query transactions with $\mathcal{R}$: for any query q issued on D and returned answer-proof pair, a *query trail* T_q is provided to $\mathcal{U}$ by $\mathcal{R}$ that is used to update Σ_q through operation updQ. These trails correspond to "receipts" that the auditing mechanism collects (namely, the update and query trails that $\mathcal{S}$ and $\mathcal{U}$ receive). This series of updates of the states Σ_u and Σ_q corresponds to the *computation phase* of the auditing mechanism.

Verification of the consistency of the two transaction series (update and query) and, consequently, replay-attack detection are performed by $\mathcal{U}$ in the *audit phase*. At any point in time (predefined or decided instantly), $\mathcal{U}$ can invoke a request for checking the consistency of the reported transactions with the current set D that resides at $\mathcal{R}$. This is performed at $\mathcal{U}$ through operation audit, which receives as input the current audit query state Σ_q of $\mathcal{U}$ and the current audit update state Σ_u of $\mathcal{S}$, appropriately updated given the current data set D (provided to $\mathcal{S}$ by $\mathcal{R}$), and accepts or rejects its input, accordingly verifying the consistency of transactions. After an audit operation that accepts its input, the audit state remains unchanged and a new computation phase begins. If it rejects, the states are reset and the next computational phase starts for a new data set: in this case, the data source $\mathcal{S}$ is responsible for creating the new data set at $\mathcal{R}$. We call the triplet of algorithms $(\mathsf{updU}, \mathsf{updQ}, \mathsf{audit})$ along with the protocols for formatting the trails an *auditing scheme*.

An auditing scheme $(\mathsf{updU}, \mathsf{updQ}, \mathsf{audit})$ is secure if it satisfies the following property: operation audit accepts its input if and only if no malicious action has been performed by $\mathcal{R}$, i.e., all query-answer pairs verified by $\mathcal{U}$ are

consistent with the update history of D and the states computed using operations upd$\mathsf{U}$, upd$\mathsf{Q}$. In particular, (upd$\mathsf{U}$, upd$\mathsf{Q}$, audit) is secure if the following conditions hold: *completeness*, dictating that all valid update and query transactions yield (through upd$\mathsf{U}$ and upd$\mathsf{Q}$) audit states that when checked by audit with a valid (not corrupted by $\mathcal{R}$) data set D always result in accepting; and *soundness*, dictating that when audit accepts its inputs, then the audit states correspond to transactions of valid update/query operations subject to the current data set.

To detect and prevent replay attacks, we augment the authentication scheme of Section 3 with a secure auditing scheme (upd$\mathsf{U}$, upd$\mathsf{Q}$, audit) as follows. After updates, along with the update at $\mathcal{S}$ and $\mathcal{R}$ of the underlying authentication structure, $\mathcal{S}$ runs upd$\mathsf{U}$ to update its update audit state, but now no signature refreshing is performed: only $O(\log^* n)$ hash values are signed by $\mathcal{S}$. After queries, along with the answer verification, $\mathcal{U}$ also runs upd$\mathsf{Q}$ to update its query audit state. If $\mathcal{R}$ launches a replay attack at some point in time, it will be detected by $\mathcal{U}$ at the first audit phase occurring after the attack since, by the security property, audit will reject its input. So, a rejecting audit phase is equivalent to detecting a replay attack launched by $\mathcal{R}$, and a misbehaving $\mathcal{R}$ who performs replay attacks is always caught and exposed to its victim $\mathcal{U}$. Note that this technique provides only detection and cannot pinpoint which query-answer pairs were replayed.

To construct a secure auditing scheme, we use a simple cryptographic solution that is inspired from efficient and secure cryptographic mechanisms for off-line memory checking by Blum *et al.* [5]. In off-line memory checking, a trusted checker checks the correctness (or consistency) of an untrusted memory, where data is written in and read from the memory through operations load and store. The checker maintains some constant-size state and augments the data that is written into the untrusted memory with *time-stamps*, such that at any point in time, a check can be performed on the memory correctness. The idea is to use a cryptographic primitive A for generating and updating this state, as a short description of the memory history. A can produce short digests of large sets in an incremental fashion (i.e., elements are inserted in the set and the new digest is updated in $O(1)$ time without recomputing from scratch) and is used as follows. After any (augmented) load or store operation performed in the memory, a special encoding of the operation is created and securely enclosed in the state through A. In particular, two separate digests are maintained over two sets: a first set encodes the "load" history of the memory (i.e., reading operations); the second set encodes the "store" history of the memory (i.e., writing operations). An operation results in updating both sets (e.g., operation load(i) adds an item d_i in the "load" history and item d'_i with new time-stamp in the "store" history). The crucial property of the approach in [5] is that if the memory is correct, the encodings produce load and store digests that are the same when the check is performed. By choosing the cryptographic primitive A such that it is collision-resistant, meaning that its computationally infeasible to find distinct sets that produce the same digest, the memory checking problem is reduced to an equality testing problem (subject to an appropriate encoding for the operations in the

memory). Primitives A for incrementally computing collision-resistant digests of sets exist (e.g., ϵ-biased hash functions in the original work [5]).

We next design an efficient secure auditing scheme that is based on the above checking technique. The challenge in applying this idea in our three-party model is to implement the checking functionality collaboratively by $\mathcal{S}$ and $\mathcal{U}$ without destroying super-efficiency at $\mathcal{U}$. We use the RSA accumulator as a collision-resistance primitive A for incrementally computing digests over sets and use $A(S)$ to denote the digest of set S. Thus, given $A(S)$ and a new element x not in S, $A(S \cup x)$ can be computed in $O(1)$ time; also, it is hard to find sets $S \neq S'$ such that $A(S) = A(S')$. We use A to define the audit states Σ_u and Σ_q stored by $\mathcal{S}$ and $\mathcal{U}$. The main idea is as follows. We view the set S of special values defined over our super-efficient authentication structure of Section 3 as an untrusted memory: memory locations correspond to the unique identifiers of the tree nodes (according to a fixed ordering, e.g., in-order tree traversal) and memory items correspond to the special hash values and their signatures.

Every transaction (update or query) uniquely defines a subset of special hash values in the tree: for updates, the hashes in the $O(\log^* n)$ special tree levels in the corresponding leave-to-root path; for queries, the two hashes of the lowest special tree level that exactly covers the answer. These two subsets of special hashes respectively define the update trail T_u and the query trail T_q that are returned by $\mathcal{R}$. For each tree node v in a subset, the tuple $(id_v, h_v, \sigma_v, t_v)$ is included in the corresponding trail. Here, id_v is the identifier of v, h_v the hash value, σ_v the corresponding signature and t_v the associated timestamp. Algorithms updU and updQ process these trails to update the audit states $\Sigma_u = (A_{u,l}, A_{u,s})$ and $\Sigma_q = (A_{q,l}, A_{q,s})$. Each audit state is a pair of values, one for "load" history, one for "store"; $A_{u,l}$, $A_{u,s}$ are integer values and $A_{q,l}$,$A_{q,s}$ are accumulations. The tuple of v is encoded (according to fixed way) to a unique string x_v (e.g., by applying an one-way hash function) and for each tuple in the trails the states are updated to $\Sigma'_u = (A'_{u,l}, A'_{u,s})$ and $\Sigma'_q = (A'_{q,l}, A'_{q,s})$, as follows: $A'_{u,l} = A_{u,l} \cdot e(x_v) \mod \phi(N)$, $A'_{u,s} = A_{u,s} \cdot e(x'_v) \mod \phi(N)$, $A'_{q,l} = A_{q,l}^{e(x_v)} \mod N$, $A'_{q,s} = A_{q,s}^{e(x'_v)} \mod N$, where $e(\cdot)$ is a function for computing prime representative values, N is the RSA modulo, and x'_v is encoding x_v but with a fresh time-stamp (monotonically increasing, synchronized for all parties) and possibly with a new identifier, hash value and signature (only for updates).

The audit phase is as follows. First $\mathcal{R}$ forwards the request for the audit to $\mathcal{S}$, along with a final audit trail that contains a tuple for each special node in set S (final reading of memory). $\mathcal{S}$ updates its update audit state (only the "load" part), signs the final Σ_u and forwards it to $\mathcal{U}$, through $\mathcal{R}$. Given $(A_{u,l}, A_{u,s})$, $(A_{q,l}, A_{q,s})$, audit (run at $\mathcal{U}$) accepts if and only if: $A_{q,l}^{A_{u,l}} \equiv A_{q,s}^{A_{u,s}} \mod N$.

Theorem 3. *There exists a hash-based, dynamic, super-efficient and audited authentication scheme for range search queries over a set of size n with the following performance, where t denotes the number of data items returned by a query: (i) a query is answered in $O(\log n + t)$ time; an update results in $O(\log n)$ hash operations (at both the source and the responder), $O(\log^* n)$ signature generations*

(at the source); (ii) the answer proof has size $O(\log t)$ and consists of two signatures, two keys and $O(\log t)$ hash values; (iii) the answer to a query is validated by performing $O(t)$ hash operations and verifying $O(1)$ signatures; (iv) the auditing scheme stores $O(1)$ audit state, performs $O(\log n)$ work per update (at the source) and $O(1)$ work per query (at the user) during the computation phase and performs $O(n)$ work (at the source) and $O(1)$ work (at the user) during the audit phase; (v) replay attacks performed by the responder are always detectable by the user at the audit phase.

Proof. (Sketch.) The complexity for the queries and updates follow from Theorems 1 and 2, by observing that no signature refreshing is necessary after updates at $\mathcal{S}$. The update and query audit states are both of $O(1)$ size (a pair of values). At the computation phase, each update incurs $O(\log n)$ cost at $\mathcal{S}$ for updating the authentication structure (hashing along the update path and updating $O(\log^* n)$ signatures and the audit state with $O(\log^* n)$ exponent accumulations). Each query incurs $O(1)$ cost at $\mathcal{U}$ (at most two values are accumulated in the audit state). At the audit phase, the cost at $\mathcal{S}$ is $O(n)$, since $\mathcal{S}$ accumulates in the exponent all special hash values currently in the authentication structure; the cost at $\mathcal{U}$ is $O(1)$ as before. Security follows from the correctness of the checking mechanism of [5] and the collision-resistance property of the RSA accumulator. Recall that $\mathcal{R}$ does not know the trapdoor $\phi(N)$ of the accumulator. Regarding soundness, suppose that audit fails to detect a replay attack launched by $\mathcal{R}$. Either the provided by $\mathcal{R}$ update and query trails were correct or there existed one trail that was invalid. In the former case and given that the audit mechanism accepts, the memory checking technique is incorrect; in the latter case, there exist different sets S and S' that produce the same RSA-based accumulations $A(S) = A(S')$. We must conclude that either the $\mathcal{R}$ was able to compute the trapdoor $\phi(N)$ for the RSA modulo N (a task that is computationally equivalent to factoring N) or $\mathcal{R}$ was able, given $A_{q,l}^{A_{u,l}} \mod N$, to compute (through the query trails that $\mathcal{R}$ provided $\mathcal{U}$ with, which are distinct from the update trails) values $A_{q,s}$ and $A_{u,s}$ such that $A_{q,l}^{A_{u,l}} \mod N = A_{q,s}^{A_{u,s}} \mod N$ (a task that is computationally infeasible under the strong RSA assumption). □

6 Extensions and Concluding Remarks

Our authentication schemes are based on the authentication structure for range search queries of Section 3. Many other query types are related to range searching or consist of more complex search problems that eventually boil down to range searching. This suggests that our authentication schemes can be used as general design tools for achieving super-efficient authentication of other types of queries. Indeed, all that is needed is to consider a (different) hashing scheme over the data set D (computed along the hash tree), which should be appropriate for the target query type. Similar to the construction in Section 3, the hashing scheme over D should securely encode these relations that are sufficient for verifying the answers to the queries in consideration. Super-efficiency would then follow

simply by authenticating at most two special hash values at the appropriate special level of the tree, depending on the exact range defined by the query.

We briefly discuss two types of queries that fall into this category. Consider the class of queries that ask for any *associative* function over some field of data records that lie in a query range. The canonical members of this class are *aggregate* queries, e.g., SUM, MAX, AVG. An appropriate hashing scheme for these queries would be constructed such that it encodes the information (relations) about ranges, the corresponding aggregation values and the neighboring data records. In particular, the hash tree node v defining subtree T_v stores a hash value that encodes information about the aggregation value a_v computed over the records that correspond to the leaves of T_v, the left-most and right-most records in T_v and, also, their predecessor and successor records (not in T_v), respectively. Using this hashing scheme, these queries can be authenticated by considering the (at most two) allocation nodes that correspond to the query range and lie in some special tree level and *without* applying any associate operation. Similarly, we can use our schemes for the class of *path property* queries that are studied in [13]—all related to range searching. Our hashing scheme of Section 3 and, accordingly, all of our authentication schemes can be extended to these classes of queries (aggregation and path-property queries).

In conclusion, in this paper we study data authentication in a setting where critical information is queried at high rates from dynamic outsourced databases that reside in untrusted sites. We propose a new approach for query authentication, where, by decoupling the answer-generation and answer-verification procedures, super-efficient answer verification is enabled, a theoretically interesting and practically important property. We design the first authentication schemes for range searching that achieve super-efficiency (answers of size t are verified in time $O(t)$, using only $O(1)$ modular exponentiations), allow for efficient updates on the database and eliminate the replay attacks from the database responder. To prevent replay attacks on old invalid data, we design an authentication protocol that implements an off-line auditing mechanism, which checks the consistency of a dynamic database and reliably reports malicious actions of the responder. Open problems include further improving the update costs of our authentication schemes and extending our auditing scheme in a multi-user setting.

References

[1] Atallah, M.J., Cho, Y., Kundu, A.: Efficient data authentication in an environment of untrusted third-party distributors. In: Proceedings of International Conference on Data Engineering (ICDE) (to appear, 2008)

[2] Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)

[3] Benaloh, J., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Proceedings of Advances in Cryptology — EUROCRYPT, pp. 274–285 (1994)

[4] Bertino, E., Carminati, B., Ferrari, E., Thuraisingham, B., Gupta, A.: Selective and authentic third-party distribution of XML documents. IEEE Transactions on Knowledge and Data Engineering 16(10), 1263–1278 (2004)
[5] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12(2/3), 225–244 (1994)
[6] Buldas, A., Laud, P., Lipmaa, H.: Accountable certificate management using undeniable attestations. In: Proceedings of ACM Conference on Computer and Communications Security, pp. 9–18. ACM Press, New York (2000)
[7] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
[8] Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., Stubblebine, S.: Flexible authentication of XML documents. Journal of Computer Security 6, 841–864 (2004)
[9] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic data publication over the Internet. Journal of Computer Security 11(3), 291–314 (2003)
[10] Di Battista, G., Palazzi, B.: Authenticated relational tables and authenticated skip lists. In: Proc. Working Conference on Data and Applications Security (DBSEC), pp. 31–46 (2007)
[11] Gassko, I., Gemmell, P.S., MacKenzie, P.: Efficient and fresh certification. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 342–353. Springer, Heidelberg (2000)
[12] Goodrich, M.T., Tamassia, R., Hasic, J.: An efficient dynamic and distributed cryptographic accumulator. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 372–388. Springer, Heidelberg (2002)
[13] Goodrich, M.T., Tamassia, R., Triandopoulos, N., Cohen, R.: Authenticated data structures for graph and geometric searching. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 295–313. Springer, Heidelberg (2003)
[14] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 121–132 (2006)
[15] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures. Algorithmica 39(1), 21–41 (2004)
[16] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
[17] Micali, S., Rabin, M., Kilian, J.: Zero-Knowledge sets. In: Proceedings of Symposium of Foundations of Computer science (FOCS), pp. 80–91 (2003)
[18] Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. In: Proceeding of Network and Distributed System Security (NDSS) (2004)
[19] Naor, M., Nissim, K.: Certificate revocation and certificate update. In: Proceedings 7th USENIX Security Symposium, pp. 217–228 (1998)
[20] Narasimha, M., Tsudik, G.: Authentication of outsourced databases using signature aggregation and chaining. In: Proceedings of 11th International Conference on Database Systems for Advanced Applications, pp. 420–436 (2006)
[21] Nuckolls, G.: Verified query results from hybrid authentication trees. In: Proceedings of Data and Applications Security (DBSec), pp. 84–98 (2005)
[22] Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1041–1053. Springer, Heidelberg (2004)

[23] Pang, H., Jain, A., Ramamritham, K., Tan, K.-L.: Verifying completeness of relational query results in data publishing. In: Proceedings of ACM SIGMOD Int. Conference on Management of data, pp. 407–418 (2005)
[24] Papamanthou, C., Tamassia, R.: Time and space efficient algorithms for two party authenticated data structures. In: Qing, S., et al. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 1–15. Springer, Heidelberg (2007)
[25] Tamassia, R.: Authenticated data structures. In: Proceedings of European Symposium on Algorithms, pp. 2–5 (2003)
[26] Tamassia, R., Triandopoulos, N.: Computational bounds on hierarchical data processing with applications to information security. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 153–165. Springer, Heidelberg (2005)
[27] Tamassia, R., Triandopoulos, N.: Efficient content authentication in peer-to-peer networks. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 354–372. Springer, Heidelberg (2007)